



# SAS<sup>®</sup> Event Stream Processing

## 6.1: Connectors and Adapters

### General Overview

Connectors and adapters enable you to stream data into or out of an event 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 event stream processing engine. A single instance of a connector reads from or writes to a single window of an event 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 1 Connectors and Adapters*

Name	Description	Connector Available?	Adapter Available?
<a href="#">Adapter</a>	Functions as a wrapper around an adapter, enabling the ESP server to process the adapter as a connector.	Yes	No
<a href="#">BACNet</a>	Enable 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
<a href="#">BoardReader</a>	Enables you to stream data from a feed aggregator. The site <a href="#">boardreader.com</a> enables you to search message boards, websites, blogs, and other social media.	No	Yes
<a href="#">SAS Cloud Analytic Services</a>	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
<a href="#">Cassandra</a>	Enables you to stream data to and from Apache Cassandra databases.	No	Yes
<a href="#">Database</a>	Support publish and subscribe operations to a large number of databases.	Yes	Yes

Name	Description	Connector Available?	Adapter Available?
Event Stream Processor	Enables you to subscribe to a window and publish what it passes into another Source window.	No	Yes
File and Socket	Support both publish and subscribe operations on files or socket connections that stream a large number of data types.	Yes	Yes
HDAT Reader	SAS software provides a file format for Hadoop called SASHDAT, which makes files available for load to SAS LASR Analytic Server. The HDAT reader adapter converts each row in a SASHDAT file into an ESP event and injects event blocks into a Source window of an engine.	No	Yes
HDFS	Supports publish and subscribe operations to a Hadoop Distributed File System.	No	Yes
Java Message Service (JMS)	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
Kafka	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
Kinesis	Enable you to use Amazon Kinesis Data Streams to collect and process streams of data records in real time.	Yes	Yes
SAS LASR Analytic Server	Supports publish and subscribe operations on the SAS LASR Analytic Server.	No	Yes
Modbus	Enable you to stream data to and from Modbus, a serial communications protocol commonly used to connect industrial electronic devices.	Yes	Yes
MQTT	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
Nurego	Enables you to stream data from Nurego, a cloud-based analytics solution for subscription businesses. The Nurego connector is intended for use with a Metering window.	Yes	No
OPC-UA	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
PI	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
Project Publish	Enables you to subscribe to event blocks that are produced by a window from a different project within the event stream processing model.	Yes	No

Name	Description	Connector Available?	Adapter Available?
<a href="#">Pylon</a>	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
<a href="#">Rabbit MQ</a>	Enable you to stream data to and from RabbitMQ, a lightweight open-source message broker.	Yes	Yes
<a href="#">REST</a>	Supports subscribe operations to generate HTTP POST requests to a configured REST service.	No	Yes
<a href="#">SAS Data Set</a>	Connects to a SAS Workspace Server in order to stream data to and from a SAS data set.	No	Yes
<a href="#">SMTP</a>	Enable you to stream data through the Simple Mail Transfer Protocol, an internet standard for electronic mail (email) transmission.	Yes	Yes
<a href="#">Sniffer</a>	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
<a href="#">Solace</a>	Enable you to stream data to and from Solace, a message broker to establish event-driven interactions between applications and microservices.	Yes	Yes
<a href="#">Teradata</a>	Enable you to stream data to and from a Teradata server.	Yes	Yes
<a href="#">Teradata Listener</a>	Enable you to stream data from a Teradata Listener, which ingests and distributes extremely fast moving data streams throughout an analytical ecosystem,	Yes	Yes
<a href="#">Tervela</a>	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
<a href="#">Tibco Rendezvous</a>	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
<a href="#">Timer</a>	Generate and publish trigger events at regular intervals.	Yes	Yes
<a href="#">URL</a>	Consumes data into a project through a URL-based connection.	Yes	No
<a href="#">UVC</a>	Enable you to publish photos taken by a V4L2-compatible camera to an event stream processing model.	Yes	Yes
<a href="#">WebSocket</a>	Enables you to read data over a WebSocket-based connection and publish the data into a project.	Yes	No
<a href="#">IBM WebSphere</a>	Support the IBM WebSphere Message Queue Interface for publish and subscribe operations.	Yes	Yes

---

## Overview to Connectors

### How Do Connectors Work?

Connectors use the SAS Event Stream Processing 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 SAS Event Stream Processing. 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 SAS Event Stream Processing. 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:

```
<project-connectors>
  <connector-groups>
    <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);
```

## Connector Examples

### XML Examples

Examples of connectors specified in XML models are available in `$DFESP_HOME/examples/xml`:

- `db_connector_publisher_xml`

- `db_connector_subscriber_xml`
- `url_connector_new`
- `url_connector_weather`
- `ws_connector_json`
- `ws_connector_xml`
- `xml_connector_publisher_xml`
- `xml_connector_subscriber_xml`

## C++ Examples

Connector examples in C++ are available in `$DFESP_HOME/examples/cxx`. The `sample_connector` directory includes source code for a user-defined connector derived from the `dfESPconnector` base class. It also includes sample code that invokes a sample connector. For more information about how to write a connector and getting it loaded by the connector manager, see [“Writing and Integrating a Custom Connector”](#).

The remaining connector examples implement application code that invokes existing connectors. These connectors are loaded by the connector manager at initialization. Those examples are as follows:

- `db_connector_publisher`
- `db_connector_subscriber`
- `json_connector_publisher`
- `json_connector_subscriber`
- `socket_connector_publisher`
- `socket_connector_subscriber`
- `xml_connector_publisher`
- `xml_connector_subscriber`

---

## Overview to Adapters

### How Do Adapters Work?

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 SAS Event Stream Processing libraries. Therefore, you must install SAS Event Stream Processing on the computer system where an adapter is to be used. That system should not use a product license if it is not licensed to run an event stream processing engine.

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.

**Table 2** C++ Adapters

---

Bacnet Publisher  
Database  
Event Stream Processor  
File and Socket  
Kafka  
Modbus  
MQTT  
OPC-UA  
PI  
Pylon Publisher  
Rabbit MQ  
SMTP Subscriber  
Sniffer Publisher  
Solace Systems  
TD Listener Subscriber  
Teradata Subscriber  
Tervela Data Fabric  
Tibco Rendezvous (RV)  
Timer Publisher  
UVC Publisher  
IBM WebSphere MQ

---

**Table 3** Java Adapters

---

SAS Cloud Analytic Server  
Cassandra  
HDAT Reader  
HDFS (Hadoop Distributed File System)  
Java Message Service (JMS)  
SAS LASR Analytic Server  
REST Subscriber  
SAS Data Set

---

**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.

Table 4 Python Adapters

## BoardReader

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 in [the table here](#).

C++ and Java adapters can publish or subscribe using a Rabbit MQ server instead of a direct TCP/IP connection to the ESP 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 ESP 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 ESP 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”](#).

**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"
```

## Using the Adapter Connector

The Adapter Connector functions as a wrapper around an adapter, enabling the ESP 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”](#).)

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 ESP 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 5** Required Parameters for the Adapter Connector

Parameter	Description
<code>command</code>	<p>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 <code>command</code> parameter.</p> <p>Include the <code>-k</code> parameter for adapters that require it to specify the mode.</p> <p>Do not include <code>-h</code> in the <code>command</code> parameter definition here. It is overwritten by the URL specified by the <code>url</code> parameter or the generated default URL.</p>
<code>type</code>	<p>Specifies the mode of the adapter. The mode can be “pub” or “sub” and must match the mode of the running adapter.</p>

**Table 6** Optional Parameters for the Adapter Connector

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

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>
  </properties>
</connector>
```

```

    <property name='type'>pub</property>
    <property name='url'>-u dfESP://espserv1:59999/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.

---

## Using the Bacnet Connector and Adapter

### 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 event 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](#) configured in the event stream processing model.

**Table 7** 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 ESP event is built. If there is no matching field for a received value type, a fatal error is reported.

**Table 8** 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 9** 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 <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.config</code> (Windows) to parse for configuration parameters. Specify the value as <code>[configfilesection]</code> .
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 <code>opcode=Upsert</code> instead of <code>Insert</code> .
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 10** BACnet to SAS Event Stream Processing Data Type Mappings

BACnet type	SAS Event Stream Processing 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

## Using the BACnet Publisher Adapter

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

**Note:** Support for the BACnet adapter is available only on Linux platforms.

**dfesp\_bacnet\_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 11** Required Keys

Key-Value Pair	Description
<code>url=pubsubURL</code>	Specifies the standard URL in the form <b>dfESP://host:port/project/continuousquery/window</b> .
<code>bacnetbbmdaddress=BBMDipaddr</code>	Specifies the IP address of the BBMD.

Key-Value Pair	Description
<code>bacnetbbmdport=BBMDport</code>	Specifies the local port used by the adapter. The default port number is 47808.
<code>bacnetconfigfile=jsonfile</code>	Specifies the JSON configuration file containing BACnet device and object information.

**Table 12** *Optional Keys*

Key-Value Pair	Description
<code>gdconfig=gdconfigfile</code>	Specifies the guaranteed delivery configuration file.
<code>transport=native   solace   tervela   rabbitmq   kafka</code>	Specifies the transport type. If you specify <code>solace</code> , <code>tervela</code> , <code>rabbitmq</code> , or <code>kafka</code> transports instead of the default <code>native</code> transport, use the required client configuration files. These files are specified in the description of the C++ <code>C_dfESPpubsubSetPubsubLib()</code> API call.
<code>loglevel=trace   debug   info   warn   error   fatal   off</code>	Sets the logging level for the adapter. This is the same range of logging levels that you can set in the <code>C_dfESPpubsubInit()</code> <code>publish/subscribe</code> API call and in the engine <code>initialize()</code> call. The default level is <code>warn</code> .
<code>logconfigfile=logconfigfile</code>	Specifies the log configuration file.
<code>tokenlocation=location</code>	Specifies the location of the file in the local file system that contains the OAuth token required for authentication by the <code>publish/subscribe</code> server.
<code>configfilesection=[section]</code>	Specifies the name of the section in <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.config</code> (Windows) to parse for connection parameters.
<code>blocksize=evntblcksize</code>	Sets the event block size. The default value is 1.
<code>transactional=true   false</code>	When <code>true</code> , specifies that events are transactional.
<code>publishwithupsert=true   false</code>	When <code>true</code> , builds events with <code>opcode=Upsert</code> instead of <code>Insert</code> .
<code>restartonerror=true   false</code>	When <code>true</code> , restarts the adapter after a fatal error is reported.
<code>bacnetipport=localport</code>	Specifies the local port used by the adapter. The default port number is 47808.

Key-Value Pair	Description
<code>transportconfigfile=file</code>	Specifies the transport configuration file. The default value depends on the transport type specified. For <code>solace</code> , the default is <code>./solace.cfg</code> . For <code>tervela</code> , the default is <code>./client.config</code> . For <code>rabbitmq</code> , the default is <code>./rabbitmq.cfg</code> . For <code>kafka</code> , the default is <code>./kafka.cfg</code> . <b>Note:</b> No transport configuration file is required for native transport.
<code>ignoretimeouts=true   false</code>	When <code>true</code> , 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.
<code>maxevents=maxevents</code>	Specifies the maximum number of events to publish.
<code>quiesceproject=true   false</code>	When <code>maxevents</code> 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” in SAS Event Stream Processing: Implementing Failover](#).

## Using the BoardReader Publisher Adapters

The BoardReader application is a feed aggregator. The site [boardreader.com](http://boardreader.com) enables you to search message boards, websites, blogs, and other social media.

The BoardReader adapters provided by SAS Event Stream Processing 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/sas/viya/config/etc/SASEventStreamProcessingEngine/default/boardreader-**-config.txt` (Linux) or `%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\boardreader-**-config.txt` (Windows). Sample versions of these files are included in your SAS Event Stream Processing 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 event stream processing server 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” in *SAS Event Stream Processing: Publish/Subscribe API*. 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-smtp-host emailsmtp-host <-email-subject emailsubject >> <-email-to emailto > <-esp-date-format dateformat > <-esp-log-config-file logconfigfile > <-esp-log-level loglevel > <-esp-ignore-missing-schema-fields> <-esp-match-substrings substrings > <-esp-publish-with-upsert>
```

**Table 13** Parameters for the BoardReader Publisher Adapters

Parameter	Description
<code>-config-txt configtxt</code>	Specifies the name of a text file in the <a href="#">configuration directory</a> that contains BoardReader content source configuration parameters. See the sample files in your <a href="#">configuration directory</a> .
<code>-dev-mode-history-to-disk</code>	Provides additional debugging: writes SQLite database formatted documents to disk in your configuration directory unless you have configured the <code>work_dir</code> parameter.
<code>-dev-mode-resp-to-disk</code>	Provides additional debugging: writes responses to disk in your configuration directory unless you have configured the <code>work_dir</code> parameter.
<code>-email-from emailfrom</code>	Specifies the source address for email logging messages.

Parameter	Description
<code>-email-level emaillevel</code>	Specifies the email logging level. Permitted values are DEBUG, INFO, WARNING, ERROR, and CRITICAL. The default value is CRITICAL.
<code>-email-smtp host emailsmtphost</code>	Specifies the SMTP host for email logging messages.
<code>-email-subject emailssubject</code>	Specifies the subject line for email logging messages.
<code>-email-to emailto</code>	Specifies the destination address for email logging messages.
<code>-esp-dateformat dateformat</code>	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 <code>strftime</code> function.
<code>-esp-ignoremissingschemafields</code>	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.
<code>-esp-logconfigfile logconfigfile</code>	Specifies the name of the log configuration file.
<code>-esp-loglevel loglevel</code>	Specifies the logging level. Permitted values are TRACE, DEBUG, INFO, WARN, ERROR, or FATAL. The default value is INFO.
<code>-esp-matchsubstrings substrings</code>	Specifies a list of comma separated <code>fieldname:value</code> pairs, where input events that do not contain a value substring in the <code>fieldname</code> string field are dropped. The comparison is case-insensitive.
<code>-esp-publishwithupsert</code>	Specifies to publish events with <code>opcode = Upsert</code> . By default, events are published with <code>opcode = Insert</code> .
<code>-esp-url url</code>	Specifies the publisher standard URL in the form <code>"dfESP://host:port/project/continuousquery/window"</code>

For information about implementing hot failover for publisher adapters, see [“Publisher Adapter Failover with Kafka” in SAS Event Stream Processing: Implementing Failover](#).

## Using the SAS Cloud Analytic 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](#).

The CAS subscriber adapter enables you to append event stream processing 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 SAS Event Stream Processing 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 Viya: Data in Motion](#).

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 SAS Event Stream Processing 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. For more information, see [“Dual Authentication” in SAS Viya Administration: Authentication](#).

## 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 14** Required Keys

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

Table 15 Optional Keys

Key-Value Pairs	Description
<code>commitblocks=number</code>	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.
<code>buffersize=size</code>	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.
<code>configfilesection=section</code>	Specifies the name of the section in <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/javaadapters.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\javaadapters.config</code> (Windows) to parse for configuration parameters.
<code>pwdencrypted= true   false</code>	When true, the CAS password is encrypted. By default, it is not.
<code>gdconfigfile=filename</code>	Specifies the guaranteed delivery configuration file for the client.
<code>casessionid=id</code>	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.
<code>pubsublib= native   solace   tervela   rabbitmq   kafka</code>	Specifies the transport type. The default value is <code>native</code> .
<code>loglevel=severe   warning   info</code>	Specifies the application logging level. The default level is <code>warning</code> .
<code>caslib=libname</code>	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.
<code>casusername=username</code>	Specifies the user name used to authenticate the SAS Cloud Analytic Services session. Allowed only when <code>casoauthtoken</code> is not specified.
<code>tokenlocation=location</code>	Specifies the location of the file in the local file system that contains the OAuth token required for authentication by the publish/subscribe server.
<code>casoauthtoken=token</code>	Specifies the OAuth token that is used to authenticate the SAS Cloud Analytic Services session. Allowed only when <code>casusername</code> is not specified.
<code>caspassword=password</code>	Specifies the password used to authenticate the SAS Cloud Analytic Services session. When <code>caspassword</code> is not specified and <code>casusername</code> is specified, the password is extracted from that user's <code>authinfo/netrc</code> file.

Key-Value Pairs	Description
<code>columns=subset</code>	Specifies a subset of columns to read or write to and from the SAS Cloud Analytic Services table. Use the form <code>c1_name,...,cn_name</code> .  <b>Note:</b> When the column names in the SAS Cloud Analytic Services table match all the field names in the event stream processing window, you do not need to use this key. The event 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).
<code>restartonerror= true   false</code>	When true, restarts the adapter when a fatal error is reported.  <b>javaadapters.config</b> parameter name: <code>restartonerror</code>
<code>transportconfigfile=file</code>	Specifies the full path to the publish/subscribe transport configuration file. The default file depends on the transport type specified with through the <code>pubsublib</code> parameter:  For <code>solace</code> , the default file is <code>solace.cfg</code> . For <code>tervela</code> , the default file is <code>client.config</code> . For <code>rabbitmq</code> , the default file is <code>rabbitmq.cfg</code> . For <code>kafka</code> , the default file is <code>kafka.cfg</code> .  <b>Note:</b> No transport configuration file is required for native transport.
<code>maxnumrows=number</code>	Specifies the maximum number of table rows per <code>addTable</code> action. By default, the action is kept open indefinitely.
<code>periodicity=number</code>	Specifies the maximum number of seconds to keep the <code>addTable</code> 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 `casessionid` 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 16** Required Keys

Key-Value Pair	Description
<code>url=url</code>	Specifies the event stream processing standard URL in the following format:  <code>"dfESP://host:port/project/contquery/window"</code> .
<code>cashostport=url</code>	Specifies the CAS server host name and port in the form <code>"host:port"</code>
<code>type=pub</code>	Specifies a publisher adapter.

Key-Value Pair	Description
<code>castable=table</code>	Specifies the full name of the CAS table.

Table 17 Optional Keys

Key-Value Pair	Description
<code>blocksize=size</code>	Specifies the number of events to be flushed to the server. The default value is 1.
<code>configfilesection=section</code>	Specifies the name of the section in <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/javaadapters.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\javaadapters.config</code> (Windows) to parse for configuration parameters.
<code>quiesceproject=true   false</code>	When true, specifies to quiesce the project after all events are injected into the Source window. <b>javaadapters.config</b> parameter name: <code>quiesceproject</code>
<code>pwdencrypted= true   false</code>	When true, the CAS password is encrypted. By default, it is not.
<code>transactional= true   false</code>	When true, event blocks are transactional. By default, they are normal.
<code>gdconfigfile=filename</code>	Specifies the guaranteed delivery configuration file for the client.
<code>casessionid=id</code>	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.
<code>pubsublib= native   solace   tervela   rabbitmq   kafka</code>	Specifies the transport type. The default value is <code>native</code> .
<code>loglevel=severe   warning   info</code>	Specifies the application logging level. The default level is <code>warning</code> .
<code>caslib=libname</code>	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.
<code>casusername=username</code>	Specifies the user name used to authenticate the SAS Cloud Analytic Services session. Allowed only when <code>casoauthtoken</code> is not specified.
<code>tokenlocation=location</code>	Specifies the location of the file in the local file system that contains the OAuth token required for authentication by the publish/subscribe server.
<code>casoauthtoken=token</code>	Specifies the OAuth token that is used to authenticate the SAS Cloud Analytic Services session. Allowed only when <code>casusername</code> is not specified.

Key-Value Pair	Description
<code>caspassword=password</code>	Specifies the password used to authenticate the SAS Cloud Analytic Services session. When <code>caspassword</code> is not specified and <code>casusername</code> is specified, the password is extracted from that user's <code>authinfo/netrc</code> file.
<code>columns=subset</code>	Specifies a subset of columns to read or write to and from the SAS Cloud Analytic Services table. Use the form <code>c1_name,...cn_name</code> . <b>Note:</b> When the column names in the SAS Cloud Analytic Services table match all the field names in the event stream processing window, you do not need to use this key. The event 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).
<code>publishwithupsert=true   false</code>	When true, builds events with <code>opcode=Upsert</code> . By default, events are built with <code>opcode=Insert</code> . <b>javaadapters.config</b> parameter name: <code>publishwithupsert</code>
<code>restartonerror= true   false</code>	When true, restarts the adapter when a fatal error is reported. <b>javaadapters.config</b> parameter name: <code>restartonerror</code>
<code>maxevents=number</code>	Specifies the maximum number of events to publish.
<code>transportconfigfile=file</code>	Specifies the full path to the publish/subscribe transport configuration file. The default file depends on the transport type specified with through the <code>pubsublib</code> parameter: For <code>solace</code> , the default file is <code>solace.cfg</code> . For <code>tervela</code> , the default file is <code>client.config</code> . For <code>rabbitmq</code> , the default file is <code>rabbitmq.cfg</code> . For <code>kafka</code> , the default file is <code>kafka.cfg</code> . <b>Note:</b> No transport configuration file is required for native transport.

**Note:** When the SAS Cloud Analytic Services table contains fewer columns than the event stream processing window, use `columns` to choose the subset of event 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 Stream Processing data type mappings are as follows:

SAS Cloud Analytic Services type	SAS Event Stream Processing data type
INT32	INT32
INT64	INT64

SAS Cloud Analytic Services type	SAS Event Stream Processing data type
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” in SAS Event Stream Processing: Implementing Failover](#).

---

## Using the Cassandra Adapter

### Overview

The Cassandra adapter is engineered to handle data to and from [Apache Cassandra databases](#). The adapter is available in `dfx-esp-cassandra-adapter.jar`, which bundles the Java publisher and subscriber SAS Event Stream Processing clients. The subscriber client receives SAS Event Stream Processing 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 event stream processing 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 driver, 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 18** 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 event stream processing standard URL in the following format: "dfESP://host:port/project/contquery/window?snapshot=true   false<? collapse=true   false>".

**Table 19** 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 <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/javaadapters.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\javaadapters.config</code> (Windows) to parse for configuration parameters.
gdconfigfile=filename	Specifies the Guaranteed Delivery configuration file for this publisher.

Key-Value Pair	Description
<code>pubsublib= native   solace   tervela   rabbitmq   kafka</code>	Specifies the transport type. When you specify the <code>solace</code> , <code>tervela</code> , <code>rabbitmq</code> , or <code>kafka</code> transports instead of the default <code>native</code> transport, use the required client configuration files specified in the description of the C++ <code>C_dfESPpubsubSetPubsubLib()</code> API call.
<code>compression=type</code>	Specifies Cassandra transport compression, valid values are "NONE", "LZ4", "SNAPPY", default is "NONE".
<code>username=name</code>	Specifies the user name to log on to the Cassandra host.
<code>loglevel=severe   warning   info</code>	Specifies the application logging level. The default value is <code>warning</code> .
<code>tokenlocation=location</code>	Specifies the location of the file in the local file system that contains the OAuth token required for authentication by the publish/subscribe server.
<code>password=password</code>	Specifies the password to use with the specified user name.
<code>ssl=true   false</code>	When true, enables Transport Layer Security (TLS) on the Cassandra connection, using the default platform JSSE options.
<code>batchsecs=seconds</code>	Specifies the maximum number of seconds to hold an incomplete batch. The default is 0.
<code>restartonerror=true   false</code>	When true, restarts the adapter if a fatal error is reported.
<code>port=port</code>	Specifies the Cassandra cluster node port. The default value is 9042.
<code>retrypolicy=policy</code>	Specifies a Cassandra retry policy, with optional comma-separated wrapped policies, default is "Default".
<code>transportconfigfile=file</code>	<p>Specifies the full path to the publish/subscribe transport configuration file. The default file depends on the transport type specified with through the <code>pubsublib</code> parameter:</p> <ul style="list-style-type: none"> <li>For <code>solace</code>, the default file is <code>solace.cfg</code>.</li> <li>For <code>tervela</code>, the default file is <code>client.config</code>.</li> <li>For <code>rabbitmq</code>, the default file is <code>rabbitmq.cfg</code>.</li> <li>For <code>kafka</code>, the default file is <code>kafka.cfg</code>.</li> </ul> <p><b>Note:</b> No transport configuration file is required for native transport.</p>
<code>unloggedbatches= true   false</code>	Specifies whether Cassandra batch statement are unlogged. The default value is <code>false</code> .

## 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 20** Required Keys

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

**Table 21** Optional Keys

Key-Value Pair	Description
<code>loadbalancingpolicy= policy</code>	Specifies a Cassandra load balancing policy, with optional comma-separated wrapped policies. The default value is "TokenAware, DCAwareRoundRobin".
<code>blocksize=number</code>	Specifies the number of events per event block. The default value is 1.
<code>configfilesection=section</code>	Specifies the name of the section in <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/javaadapters.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\javaadapters.config</code> (Windows) to parse for configuration parameters.
<code>gdconfigfile=filename</code>	Specifies the guaranteed delivery configuration file for the client.
<code>pubsublib= native   solace   tervela   rabbitmq   kafka</code>	Specifies the transport type. When you specify the <code>solace</code> , <code>tervela</code> , <code>rabbitmq</code> , or <code>kafka</code> transports instead of the default <code>native</code> transport, use the required client configuration files specified in the description of the C++ <code>C_dfESPpubsubSetPubsubLib()</code> API call.
<code>compression=value</code>	Specifies Cassandra transport compression, valid values are "NONE", "LZ4", "SNAPPY", default is "NONE".
<code>username=name</code>	Specifies the user name to log on to the Cassandra host.
<code>loglevel=severe   warning   info</code>	Specifies the application logging level. The default value is warning.

Key-Value Pair	Description
<code>tokenlocation=location</code>	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.
<code>password=password</code>	Specifies the password to use with the specified user name.
<code>quiesceproject=true   false</code>	When true, quiesces the project after all events are injected into the Source window.
<code>transactional=true   false</code>	When true, specifies that event blocks are transactional. The default event block type is normal.
<code>ssl=true   false</code>	When true, enables TLS on the Cassandra connection, using the default platform JSSE options.
<code>publishwithupsert=true   false</code>	When true, builds events with <code>opcode=Upsert</code> instead of <code>Insert</code> .
<code>restartonerror=true   false</code>	When true, restarts the adapter if a fatal error is reported.
<code>maxevents=number</code>	Specifies the maximum number of events to publish.
<code>port=number</code>	Specifies the Cassandra cluster node port. The default value is 9042.
<code>retrypolicy=policy</code>	Specifies a Cassandra retry policy, with optional comma-separated wrapped policies, default is "Default".
<code>transportconfigfile=file</code>	<p>Specifies the full path to the publish/subscribe transport configuration file. The default file depends on the transport type specified with through the <code>pubsublib</code> parameter:</p> <ul style="list-style-type: none"> <li>For <code>solace</code>, the default file is <code>solace.cfg</code>.</li> <li>For <code>tervela</code>, the default file is <code>client.config</code>.</li> <li>For <code>rabbitmq</code>, the default file is <code>rabbitmq.cfg</code>.</li> <li>For <code>kafka</code>, the default file is <code>kafka.cfg</code>.</li> </ul> <p><b>Note:</b> No transport configuration file is required for native transport.</p>

## Publisher Failover

For information about implementing hot failover for publisher adapters, see [“Publisher Adapter Failover with Kafka” in SAS Event Stream Processing: Implementing Failover](#).

---

## Using the Database Connector and Adapter

### Using the Database Connector

#### Overview

The database connector supports both publish and subscribe operations. To determine which drivers the connector uses, set the `sas` connector configuration parameter. When the `sas` is set to `true`, it uses the SAS threaded kernel drivers. When the `sas` parameter is set to `false` or is left at the default value, the database connector uses the DataDirect ODBC drivers.

#### Using the Database Connector with SAS Threaded Kernel Drivers

The connector configuration parameter `connectstring` requires a `driver=` statement that specifies the database type. Supported driver values are:

- aster
- db2
- db2zos
- hawq
- impala
- mysql
- netezza
- odbc
- oracle
- postgres
- redshift
- saphana
- mssqlsvr
- sapiq
- teradata
- vertica

No ODBCINI environment variable, `odbc.ini` file, or DSN definition is required. All corresponding parameters are defined directly in the `connectstring` parameter, following the driver specification. Some drivers also require additional installation of a client library.

Note that when you specify `driver=odbc`, you can install and use any standard ODBC-compliant driver. In this case, the driver might have other system requirements, such as `odbc.ini`. When using `driver=odbc` on Linux installations, be sure to include `dm_unicode=UCS2` when specifying the `connectstring` parameter. This ensures that the driver correctly uses UTF-16/UCS-2 when it calls the unixODBC Driver Manager.

For more information about driver-specific parameters, see the “Driver Reference for SAS Federation Server” in the *SAS Federation Server: Administrator’s Guide* and “Data Source Support” in the [SAS Cloud Analytic Services: CASL Reference](#).

## Using the Database Connector with DataDirect Drivers

Currently, the DataDirect ODBC drivers are certified for the following databases:

- Oracle
- MySQL
- IBM DB2
- Greenplum
- Hive
- PostgreSQL
- SAP Sybase ASE
- Teradata
- Microsoft SQL Server
- IBM Informix
- Sybase IQ

When using the DataDirect driver, the connector requires that you make database connectivity available through a system data source name (DSN). This DSN and the associated database user credentials are required configuration parameters for the connector.

For SAS Event Stream Processing installations that are not on Windows, the DataDirect drivers are located in `$DFESP_HOME/lib`. The DSN required for your specific database connection is configured in an `odbc.ini` file that is pointed to by the `ODBCINI` environment variable. A default `odbc.ini` template file is available in your [configuration directory](#). To enable database drivers to locate the connector software, replace all instances of `$DFESP_HOME` in the template file with the installation location of the engine.

**Note:** Some DataDirect drivers require that you load a library that contains SSL Crypto resources. The needed library is located in `$DFESP_HOME/ssl/lib`. To use the drivers that have this requirement, you must add that path to `LD_LIBRARY_PATH`.

## Configuring the Database Connector

Regardless of driver type, some drivers require the definition of a DSN. To define a DSN, ensure that the optional ODBC component is installed. Then you can configure a DSN using the Windows ODBC Data Source Administrator. This application is located in the **Windows Control Panel** under **Administrative Tools**. Beginning with Windows 8, the icon is named ODBC Data Sources. On 64-bit operating systems, there are 32-bit and 64-bit versions.

Perform the following steps to create a DSN in a Windows environment:

- 1 Enter `odbc` in the Windows search window. The default ODBC Data Source Administrator is displayed.
- 2 Select the **System DSN** tab and click **Add**.
- 3 Select the appropriate driver from the list and click **Finish**.
- 4 Enter your information in the Driver Setup dialog box.
- 5 Click **OK** when finished.

This DSN is supplied in the connect string.

Regardless of driver type, the connector publisher obtains result sets from the database using a single SQL statement configured by the user. Additional result sets can be obtained by stopping and restarting the connector. The connector subscriber writes window output events to the database table configured by the user.

If required, you can configure the `pwd` field in the `connectstring` parameter with an encrypted password. You can generate the encrypted version of the password using OpenSSL, which must be installed on your system. When you installed the SAS Event Stream Processing 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 "pwd in connectstring" | openssl enc -e -aes-256-cbc -a -salt
-pass pass:"espDBconnectorUsedByUser=uid in connectstring"
```

Then copy the encrypted password into your `connectstring` parameter and enable the `pwdencrypted` parameter.

Use the following parameters with database connectors, regardless of driver type.

**Note:** To improve performance on the Teradata platform, a separately packaged connector and adapter use the Teradata Parallel Transporter.

**Table 22** Required Parameters for Subscriber Database Connectors

Parameter	Description
<code>connectstring</code>	<p>When using DataDirect drivers, specifies the database DSN and user credentials. Use the following format:</p> <pre>DSN=dsn[;uid=userid] [;pwd=password] .</pre> <p>When using SAS/TKTS drivers, specifies the driver type and other driver-specific parameters.</p>
<code>desttablename</code>	Specifies the target table name.
<code>snapshot</code>	<p>Specifies whether to send snapshot data.</p> <p>When <code>true</code>, 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.</p>
<code>type</code>	Specifies to subscribe. Must be "sub".

**Table 23** Required Parameters for Publisher Database Connectors

Parameter	Description
<code>connectstring</code>	<p>When using DataDirect drivers, specifies the database DSN and user credentials. Use the following format:</p> <pre>DSN=dsn[;uid=userid] [;pwd=password] .</pre> <p>When using SAS/TKTS drivers, specifies the driver type and other driver-specific parameters.</p>
<code>type</code>	Specifies to publish. Must be "pub".

**Table 24** *Optional Parameters for Subscriber Database Connectors*

Parameter	Description
configfilesection	Specifies the name of the section in <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.config</code> (Windows) to parse for configuration parameters. Specify the value as <code>[configfilesection]</code> .
commitrows	Specifies the minimum number of output rows to buffer.
commitsecs	Specifies the maximum number of seconds to hold onto an incomplete commit buffer.
ignoresqlerrors	Enables the connector to continue to write Inserts, Updates, and Deletes to the database table despite an error in a previous Insert, Update, or Delete.
maxcolbinding	Specifies the maximum supported width of string columns. The default value is 4096.
pwdencrypted	Specifies that the <code>pwd</code> field in <code>connectstring</code> is encrypted.
rmretdel	Specifies to remove all delete events from event blocks received by a subscriber that were introduced by a window retention policy.
sas	When <code>true</code> , specifies to use the SAS threaded kernel drivers instead of the DataDirect drivers. The default value is <code>false</code> .

**Table 25** *Optional Parameters for Publisher Database Connectors*

Parameter	Description
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 <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.config</code> (Windows) to parse for configuration parameters. Specify the value as <code>[configfilesection]</code> .
greenplumlogminer	Enables Greenplum log miner mode. <a href="#">“Using Log Miner Modes”</a> .
logminerdbname	Specifies the <code>gpperfmon</code> database that contains the <code>queries_history</code> table for Greenplum log miner mode. Use the following format: <code>dd-mmm-yy hh:mm:ss</code> .
logminerschemaowner	Specifies the schema owner when using Oracle or Greenplum log miner mode. For more information, see <a href="#">“Using Log Miner Modes”</a> .

Parameter	Description
logminerstartdatetime	Specifies the start date time when using Oracle or Greenplum log miner mode. For more information, see <a href="#">“Using Log Miner Modes”</a> .
logminertablename	Specifies the table name when using Oracle or Greenplum log miner mode. For more information, see <a href="#">“Using Log Miner Modes”</a> .
maxcolbinding	Specifies the maximum supported width of string columns. The default value is 4096.
maxevents	Specifies the maximum number of events to publish.
oraclelogminer	Enables Oracle log miner mode. <a href="#">“Using Log Miner Modes”</a> .
publishwithupsert	Builds events with opcode=Upsert instead of Insert.
pwdencrypted	Specifies that the pwd field in connectstring is encrypted.
selectstatement	Specifies the SQL statement to be executed on the source database. Required when oraclelogminer and greenplumlogminer are not enabled.
transactional	Sets the event block type to transactional. The default event block type is normal.
sas	When <b>true</b> specifies to the SAS threaded kernel drivers instead of the DataDirect drivers . The default value is <b>false</b> .

The number of columns in the source or target database table and their data types must be compatible with the schema of the involved event stream processor window.

### Subscriber Event Stream Processor to SQL Data Type Mappings

Subscriber Event Stream Processor Data Type	SQL Data Type
ESP_UTF8STR	SQL_CHAR, SQL_VARCHAR, SQL_LONGVARCHAR, SQL_WCHAR, SQL_WVARCHAR, SQL_WLONGVARCHAR, SQL_BINARY, SQL_VARBINARY, SQL_LONGVARBINARY, SQL_GUID
ESP_INT32	SQL_INTEGER, SQL_BIGINT, SQL_DECIMAL, SQL_BIT, SQL_TINYINT, SQL_SMALLINT
ESP_INT64	SQL_BIGINT, SQL_DECIMAL, SQL_BIT, SQL_TINYINT, SQL_SMALLINT
ESP_DOUBLE	SQL_DOUBLE, SQL_FLOAT, SQL_REAL, SQL_NUMERIC, SQL_DECIMAL
ESP_MONEY	SQL_DOUBLE (converted to ESP_DOUBLE), SQL_FLOAT, SQL_REAL, SQL_NUMERIC (converted to SQL_NUMERIC), SQL_DECIMAL (converted to SQL_NUMERIC)

Subscriber Event Stream Processor Data Type	SQL Data Type
ESP_DATETIME	SQL_TYPE_DATE (sets only year/month/day), SQL_TYPE_TIME (sets only hours/minutes/seconds), SQL_TYPE_TIMESTAMP (sets fractional seconds = 0)
ESP_TIMESTAMP	SQL_TYPE_TIMESTAMP

### Publisher SQL to Event Stream Processor Data Type Mappings

Publisher SQL Data Type	Event Stream Processor Data Types
SQL_CHAR	ESP_UTF8STR
SQL_VARCHAR	ESP_UTF8STR
SQL_LONGVARCHAR	ESP_UTF8STR
SQL_WCHAR	ESP_UTF8STR
SQL_WVARCHAR	ESP_UTF8STR
SQL_WLONGVARCHAR	ESP_UTF8STR
SQL_BIT	ESP_INT32, ESP_INT64
SQL_TINYINT	ESP_INT32, ESP_INT64
SQL_SMALLINT	ESP_INT32, ESP_INT64
SQL_INTEGER	ESP_INT32, ESP_INT64
SQL_BIGINT	ESP_INT64
SQL_DOUBLE	ESP_DOUBLE, ESP_MONEY (upcast from ESP_DOUBLE)
SQL_FLOAT	ESP_DOUBLE, ESP_MONEY (upcast from ESP_DOUBLE)
SQL_REAL	ESP_DOUBLE
SQL_TYPE_DATE	ESP_DATETIME (sets only year/month/day)
SQL_TYPE_TIME	ESP_DATETIME (sets only hours/minutes/seconds)
SQL_TYPE_TIMESTAMP	ESP_TIMESTAMP, ESP_DATETIME
SQL_DECIMAL	ESP_INT32 (only if scale = 0, and precision must be <= 10), ESP_INT64 (only if scale = 0, and precision must be <= 20), ESP_DOUBLE
SQL_NUMERIC	ESP_INT32 (only if scale = 0, and precision must be <= 10), ESP_INT64 (only if scale = 0, and precision must be <= 20), ESP_DOUBLE, ESP_MONEY (converted from SQL_NUMERIC)

Publisher SQL Data Type	Event Stream Processor Data Types
SQL_BINARY	ESP_UTF8STR
SQL_VARBINARY	ESP_UTF8STR
SQL_LONGVARBINARY	ESP_UTF8STR

## Using Log Miner Modes

### Overview

The database connector can run in one of two special log miner modes: Oracle or Greenplum. These modes apply to a publisher only. They are designed to fetch rows from an Oracle Log Miner database or from a `queries_history` table in a Greenplum `gpperfmon` database. After rows are fetched, an event per row is published into a source window with a fixed, well-known subset of fields.

You configure log miner mode by enabling the appropriate parameter: `oraclelogminer` or `greenplumlogminer`.

For information about Oracle requirements, see the Oracle Administration guide: [http://docs.oracle.com/cd/B28359\\_01/server.111/b28319/logminer.htm](http://docs.oracle.com/cd/B28359_01/server.111/b28319/logminer.htm).

To summarize: the database must be in archive log mode with supplemental logging enabled. To achieve this, use the following SQL command:

```
ALTER DATABASE ADD SUPPLEMENTAL LOG DATA.
```

Whoever reads the logs must have `SELECT_CATALOG_ROLE` and `SELECT ANY TRANSACTION` privileges.

### Using Oracle Log Miner Mode

The user name that you configure through the `logminerschemaowner` parameter must have the following privileges on the Oracle Log Miner database: `SELECT_CATALOG_ROLE`, `EXECUTE_CATALOG_ROLE`, `SELECT_ANY_TRANSACTION`. The connector then repeatedly executes a select operation to pull records from the log, given a user-provided start time. Every subsequent select specifies a start time that is equal to timestamp in the most recently received `SQL_REDO` response.

Specifically, the connector defines the initial start and end times with the following statement:

```
Declare startdate date := to_date('logminerstartdatetime', 'DD-MON-YYYY
HH24:MI:SS');begin execute immediate 'ALTER SESSION SET NLS_DATE_FORMAT = ' ||
chr(39) || 'DD-MON-YYYY HH24:MI:SS' || chr(39);dbms_logmnr.start_logmnr(OPTIONS=>
DBMS_LOGMNR.DICT_FROM_ONLINE_CATALOG
+DBMS_LOGMNR.CONTINUOUS_MINE,STARTTIME=>startdate,ENDTIME=>SYSDATE-(1/86400));end;
```

The connector runs the repeated query with the following statement:

```
Select OPERATION, TIMESTAMP, replace (SQL_REDO,chr(0),' ') SQL_REDO, CSF from v
$logmnr_contents where SEG_OWNER='logminerschemaowner' and
SEG_NAME='logminertablename'and OPERATION in ('INSERT','UPDATE','DELETE');
```

The connector then processes `SQL_REDO` responses that contain Insert/Update/Delete operations. The start date value for every subsequent query is set to the value in the `sql_timestamp` column in the current response plus one second.

The statements are converted to events and injected into the Source window, whose schema must begin with the following fields:

```
"index*:int64,ts:stamp,sql_operation:string,sql_timestamp:stamp,schema:string,table
name:string,sql_where:string"
```

The connector completes these fields as follows:

- `index`: a unique incrementing value
- `ts`: the timestamp when the SQL\_REDO was received
- `sql_operation`: INSERT/UPDATE/DELETE
- `sql_timestamp`: the timestamp in the SQL\_REDO response
- `schema`: the `logminerschemaowner` configured on the connector
- `tablename`: the `logminertablename` configured on the connector
- `sql_where`: the contents of the WHERE clause in Update and Delete statements, excluding the ROWID value. Null for Insert.

The remainder of the schema must contain string fields named after columns returned in the SQL\_REDO Insert/Update/Delete responses. The values in these columns are copied into the corresponding source window schema fields.

For Update and Delete events, values from the WHERE clause are copied first. For Update events, values from the set clause are copied next, overwriting any values that were copied from the WHERE clause.

### Using Greenplum Log Miner Mode

In this mode, the connector first checks for the presence of a work table named `cq_logminerstartdatetime`. If the table exists, it is truncated, else it is created with a single column: (`last_ctime timestamp(0)` without time zone). The connector then adds a row to the table that contains the value of `logminerstartdatetime`. Then the connector repeatedly executes the following SQL code in one-second intervals against the table `queries_history` until a nonzero row count is returned:

```
SELECT COUNT(1) FROM queries_history WHERE ctime > (select max(last_ctime) FROM
"cq_logminerstartdatetime") AND db = 'logminerdbname' AND username=
'logminerschemaowner' AND query_text LIKE '%logminertablename%';
```

When a nonzero row count is returned, the work table is updated as follows:

```
INSERT INTO "cq_logminerstartdatetime" SELECT max(ctime) FROM queries_history
WHERE ctime > (SELECT max(last_ctime) FROM "cq_logminerstartdatetime");
```

Then rows are fetched from table `queries_history` as follows:

```
SELECT ctime, query_text FROM queries_history WHERE ctime > (SELECT max(last_ctime)
FROM "cq_logminerstartdatetime" WHERE last_ctime < (SELECT max(last_ctime) FROM
"cq_logminerstartdatetime")) AND db = 'logminerdbname' AND username =
'logminerschemaowner' AND query_text LIKE '%logminertablename%';
```

After all rows are fetched, the connector loops back and begins looking for another nonzero row count in table `queries_history`. Fetched rows are converted to events and injected into the Source window.

The Source window schema requirements and field contents are the same as for SQL\_REDO response processing in Oracle Log Miner mode.

## Using the Database Adapter

### Overview

See [“Using the Database Connector”](#) for functional description and configuration requirements.

```
dfesp_db_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 26** Required Keys

Key-Value Pairs	Description
<code>type=sub</code>	Specifies a subscriber adapter.
<code>url=pubsubURL</code>	<p>Specifies the standard URL in the form <code>dfESP://host:port/project/continuousquery/window</code>.</p> <p>Append <code>?snapshot=true   false</code> for subscribers.</p> <p>When <code>?snapshot=true</code>, 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.</p> <p>Append <code>?rmretdel=true   false</code> for subscribers if needed.</p>
<code>connectstring=string</code>	<p>When using the DataDirect drivers, <i>string</i> specifies the database DSN and user credentials. Use the following format:</p> <p><code>DSN=dsn[;uid=userid][;pwd=password]</code>.</p> <p>When using the SAS threaded kernel drivers, <i>string</i> specifies the driver type and other driver-specific parameters.</p>
<code>desttablename=table</code>	Specifies the subscriber target database table.

**Table 27** Optional Keys

Key-Value Pair	Description
<code>gdconfig=gdconfigfile</code>	Specifies the guaranteed delivery configuration file.
<code>transport=native   solace   tervela   rabbitmq   kafka</code>	Specifies the transport type. If you specify <code>solace</code> , <code>tervela</code> , <code>rabbitmq</code> , or <code>kafka</code> transports instead of the default <code>native</code> transport, then use the required client configuration files specified in the description of the C++ <code>C_dfESPpubsubSetPubsubLib()</code> API call.
<code>loglevel=trace   debug   info   warn   error   fatal   off</code>	Sets the logging level for the adapter. This is the same range of logging levels that you can set in the <code>C_dfESPpubsubInit()</code> <code>publish/subscribe</code> API call and in the engine <code>initialize()</code> call. The default level is <code>warn</code> .
<code>logconfigfile=logconfigfile</code>	Specifies the log configuration file.
<code>tokenlocation=location</code>	Specifies the location of the file in the local filesystem that contains the OAuth token required for authentication by the <code>publish/subscribe</code> server.
<code>maxcolbinding=maxwidth</code>	Specifies the maximum supported width of string columns. The default value is 4096.

Key-Value Pair	Description
<code>configfilesection=[section]</code>	Specifies the name of the section in <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.config</code> (Windows) to parse for configuration parameters. Specify the value as <code>[configfilesection]</code> .
<code>pwdencrypted=true   false</code>	When <code>true</code> , the password in <code>connectstring</code> is encrypted.
<code>restartonerror= true   false</code>	When <code>true</code> , restarts the adapter if a fatal error is reported.
<code>transportconfigfile=file</code>	Specifies the publish/subscribe transport configuration file. For <code>solace</code> , the default is <code>./solace.cfg</code> . For <code>tervela</code> , the default is <code>./client.config</code> . For <code>rabbitmq</code> , the default is <code>./rabbitmq.cfg</code> . For <code>kafka</code> , the default is <code>./kafka.cfg</code> . <b>Note:</b> No transport configuration file is required for native transport.
<code>commitrows=minrows</code>	Specifies the minimum number of output rows to buffer.
<code>commitsecs=maxsecs</code>	Specifies the maximum number of seconds to hold onto an incomplete commit buffer.
<code>ignoresqlerrors=true   false</code>	When <code>true</code> , ignores errors when executing SQL Inserts, Updates, or Deletes.
<code>SAS=true   false</code>	When <code>true</code> , specifies to use the SAS threaded kernel drivers instead of the DataDirect drivers. The default value is <code>false</code> .

## Publisher Usage

**Table 28** Required Keys

Key-Value Pairs	Description
<code>type=pub</code>	Specifies a publisher.
<code>url=pubsubURL</code>	Specifies the standard URL in the form <code>dfESP://host:port/project/continuousquery/window</code> .
<code>connectstring=string</code>	When using the DataDirect drivers, <code>string</code> specifies the database DSN and user credentials. Use the following format: <code>DSN=dsn[;uid=userid] [;pwd=password]</code> . When using the SAS threaded kernel drivers, <code>string</code> specifies the driver type and other driver-specific parameters.

Table 29 Optional Keys

Key-Value Pairs	Description
<code>gdconfig=gdconfigfile</code>	Specifies the guaranteed delivery configuration file.
<code>transport=native   solace   tervela   rabbitmq   kafka</code>	Specifies the transport type. If you specify <code>solace</code> , <code>tervela</code> , <code>rabbitmq</code> , or <code>kafka</code> transports instead of the default <code>native</code> transport, use the required client configuration files specified in the description of the C++ <code>C_dfESPpubsubSetPubsubLib()</code> API call.
<code>loglevel=trace   debug   info   warn   error   fatal   off</code>	Sets the logging level for the adapter. This is the same range of logging levels that you can set in the <code>C_dfESPpubsubInit()</code> <code>publish/subscribe</code> API call and in the engine <code>initialize()</code> call. The default level is <code>warn</code> .
<code>logconfigfile=logconfigfile</code>	Specifies the log configuration file.
<code>tokenlocation=location</code>	Specifies the location of the file in the local filesystem that contains the OAuth token required for authentication by the <code>publish/subscribe</code> server.
<code>maxcolbinding=maxwidth</code>	Specifies the maximum supported width of string columns. The default value is 4096.
<code>configfilesection=[section]</code>	Specifies the name of the section in <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.config</code> (Windows) to parse for configuration parameters. Specify the value as <code>[configfilesection]</code> .
<code>pwdencrypted=true   false</code>	When <code>true</code> , the password specified in <code>connectstring</code> is encrypted.
<code>restartonerror=true   false</code>	When <code>true</code> , restarts the adapter if a fatal error is reported.
<code>transportconfigfile=file</code>	Specifies the <code>publish/subscribe</code> transport configuration file. For <code>solace</code> , the default is <code>./solace.cfg</code> . For <code>tervela</code> , the default is <code>./client.config</code> . For <code>rabbitmq</code> , the default is <code>./rabbitmq.cfg</code> . For <code>kafka</code> , the default is <code>./kafka.cfg</code> . <b>Note:</b> No transport configuration file is required for native transport.
<code>selectstatement=statement</code>	Specifies the publisher source database SQL statement.
<code>blocksize=size</code>	Specifies the event block size. The default value is 1.
<code>transactional=true   false</code>	When <code>true</code> , specifies that events are transactional.
<code>quiesceproject=true   false</code>	When <code>true</code> , quiesces the project after all events are injected into the Source window.

Key-Value Pairs	Description
<code>logminerschemaowner=owner</code>	Specifies the schema owner for Oracle or Greenplum log miner mode.
<code>logminerdbname=database</code>	Specifies the <code>gpperfmon</code> database that contains the <code>queries_history</code> table.
<code>logminertablename=tablename</code>	Specifies the table name for Oracle or Greenplum log miner mode.
<code>logminerstartdatetime=datetime</code>	Specifies the start date time for Oracle or Greenplum log miner mode. Use the following format: "dd-mmm-yyyy hh:mm:ss"
<code>oraclelogminer=true   false</code>	When <code>true</code> , enables Oracle log miner mode.
<code>greenplumlogminer=true   false</code>	When <code>true</code> , enables Greenplum log miner mode.
<code>publishwithupsert=true   false</code>	When <code>true</code> , builds events with <code>opcode=Upsert</code> instead of <code>Insert</code> .
<code>maxevents=maxevents</code>	Specifies the maximum number of events to publish.
<code>SAS=true   false</code>	When <code>true</code> , specifies to use the SAS threaded kernel drivers instead of the the DataDirect drivers. The default value is <code>false</code> .

## Publisher Failover

For information about implementing hot failover for publisher adapters, see [“Publisher Adapter Failover with Kafka” in SAS Event Stream Processing: Implementing Failover](#).

## 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>`

*Table 30 Parameters for the Event Stream Processor Adapter*

Parameter	Description
<code>-p url</code>	Specifies the publish standard URL in the form <code>dfESP://host:port/project/contquery/window</code>

Parameter	Description
-s url	<p>Specifies the subscribe standard URL in the form "dfESP://host:port/project/contquery/window?snapshot=true   false"</p> <p>When <code>?snapshot=true</code>, 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.</p> <p>Append the following if needed:  <code>?collapse=true   false"</code></p>
-E tokenlocation	<p>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.</p>

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” in SAS Event Stream Processing: Implementing Failover](#).

## Using the File and Socket Connector and Adapter

### Using File and Socket Connectors

#### Overview to 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 `f$name`. 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 `f$name`, the connector implements the client. The configured host and port specify the network peer implementing the server side of the connection. However, when `host` is blank (that is, when `f$name` 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 31** Required Parameters for File and Socket Connectors

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

**Table 32** Required Parameters for Subscriber File and Socket Connectors

Parameter	Description
<code>snapshot</code>	Specifies whether to send snapshot data.  When <code>true</code> , 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 33** Optional Parameters for Subscriber File and Socket Connectors

Parameter	Description
<code>collapse</code>	Converts <code>UPDATE_BLOCK</code> events to <code>UPDATE</code> events in order to make subscriber output publishable.
<code>configfilesection</code>	Specifies the name of the section in <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.config</code> (Windows) to parse for configuration parameters. Specify the value as <code>[configfilesection]</code> .
<code>dateformat</code>	Specifies the format of <code>DATE</code> and <code>STAMP</code> fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds ( <code>DATE</code> ) or microseconds ( <code>STAMP</code> ) since epoch. The <code>dateformat</code> parameter accepts any time format that is supported by the UNIX <code>strptime</code> function.
<code>doubleprecision</code>	Specifies the number of fractional digits in the ASCII representation of a double. The default value is 6.
<code>hdatcashostport</code>	Specifies the CAS server host and port. Applies only to the <code>hdat</code> connector.
<code>hdatcaspassword</code>	Specifies the CAS server password. Applies only to the <code>hdat</code> connector.
<code>hdatcasusername</code>	Specifies the CAS server user name. Applies only to the <code>hdat</code> connector.

Parameter	Description
<code>hdatfilename</code>	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 <code>fspath</code> parameter. Do not include the <code>.sashdat</code> extension. Applies only to and is required for the <code>hdat</code> connector.
<code>hdatlasrhostport</code>	Specifies the SAS LASR Analytic Server host and port. Applies only to the <code>hdat</code> connector.
<code>hdatlasrkey</code>	Specifies the path to <code>tklasrkey.sh</code> . By default, this file is located in <code>\$DFESP_HOME/bin</code> . Applies only to the <code>hdat</code> connector.
<code>hdatmaxdatanodes</code>	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 <code>hdatnumthreads</code> $\leq$ 1. Applies only to the <code>hdat</code> connector.
<code>hdatmaxstringlength</code>	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 <code>hdat</code> 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.
<code>hdatnumthreads</code>	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 <code>hdat</code> connector.
<code>hdfsblocksize</code>	Specifies in Mbytes the block size used to write an Objective Analysis Package Data (HDAT) file. Applies only to and is required for the <code>hdat</code> connector.
<code>hdfsnumreplicas</code>	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 <code>hdat</code> connector.
<code>header</code>	<code>false</code>   <code>true</code>   <code>full</code>  For a CSV subscriber, specifies to write a header row that shows comma-separated fields. Set the value to <code>full</code> to include <code>opcode</code> , <code>flags</code> , in the header line.
<code>maxfilesize</code>	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 <code>fstype</code> other than <code>hdat</code> .
<code>periodicity</code>	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 <code>fstype</code> other than <code>hdat</code> .
<code>rate</code>	When latency mode is enabled, transmit at this specified rate to generated output files.

Parameter	Description
<code>rmretdel</code>	Specifies to remove all delete events from event blocks received by a subscriber that were introduced by a window retention policy.
<code>unbufferedoutputstreams</code>	Specifies to create an unbuffered stream when writing to a file or socket. By default, streams are buffered.
<code>hdatsecure</code>	Specifies to use SSH to connect to secure Hadoop plugins. The default value is <b>false</b> . Applies only to the <b>hdat</b> connector.
<code>hdatsecuresshloginname</code>	When <code>hdatsecure=true</code> , specifies the SSH login name. Invalid when <code>hdatsecuresshidentityfile</code> is not configured. The default value is the current user. Applies only to the <b>hdat</b> connector.
<code>hdatsecuresshidentityfile</code>	When <code>hdatsecure=true</code> , specifies the SSH identity file. Invalid when <code>hdatsecuresshloginname</code> is not configured. The default value is none. Applies only to the <b>hdat</b> connector.
<code>hdatsecurehadoopcmdpath</code>	When <code>hdatsecure=true</code> , specifies the full Hadoop command path on the remote name and data nodes. The default value is <code>"/opt/hadoop/bin/hadoop"</code> . Applies only to the <b>hdat</b> connector.

**Table 34** Optional Parameters for Publisher File and Socket Connectors

Parameter	Description
<code>addcsvflags</code>	Specifies the event type to insert into input CSV events (with a comma). Valid values are <code>normal</code> and <code>partialupdate</code> .
<code>addcsvopcode</code>	Prepends an opcode and comma to incoming CSV events. The opcode is <code>Insert unless publishwithupsert</code> is enabled.
<code>blocksize</code>	Specifies the number of events to include in a published event block. The default value is 1.
<code>cefsyslogprefix</code>	When <code>fstype=cef</code> , specifies that CEF events contain the syslog prefix.
<code>configfilesection</code>	Specifies the name of the section in <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.config</code> (Windows) to parse for configuration parameters. Specify the value as <code>[configfilesection]</code> .
<code>csvfielddelimiter</code>	Specifies the character delimiter for field data in input CSV events. The default delimiter is the <code>,</code> character.
<code>dateformat</code>	Specifies the format of <code>DATE</code> and <code>STAMP</code> fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds ( <code>DATE</code> ) or microseconds ( <code>STAMP</code> ) since epoch. The <code>dateformat</code> parameter accepts any time format that is supported by the UNIX <code>strftime</code> function.

Parameter	Description
<code>growinginputfile</code>	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.
<code>header</code>	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.
<code>ignorecsvparseerrors</code>	Specifies that when a field in an input CSV event cannot be parsed, the event is dropped, an error is logged, and publishing continues.
<code>maxevents</code>	Specifies the maximum number of events to publish.
<code>noautogenfield</code>	Specifies that input events are missing the key field that is autogenerated by the source window.
<code>prebuffer</code>	Controls whether event blocks are buffered to an event block vector before doing any injects. The default value is "false". Not valid with <code>growinginputfile</code> or for a socket connector
<code>publishwithupsert</code>	Build events with <code>opcode=Upsert</code> instead of <code>opcode=Insert</code> .
<code>rampupsecs</code>	Specifies the number of seconds to linearly ramp up from 0 to the transmit rate that you request with the <code>rate</code> parameter. The default value is 0.
<code>rate</code>	Specifies the requested transmit rate in events per second.
<code>repeatcount</code>	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.
<code>transactional</code>	Sets the event block type to transactional. The default event block type is normal.
<code>stripnullsfromcsv</code>	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:string,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,devicevendor:string,deviceproduct:string,deviceversion:string,signatureid:string,name:string,severity: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 event 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.java: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” in SAS Event Stream Processing: Publish/Subscribe API](#) 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 is supported only for publisher operations. It collects syslog events, converts them into ESP event blocks, and injects them into one or more Source windows in a running ESP 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='update' 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
<b>i</b>		Insert
<b>u</b>		Update
<b>p</b>		Upsert
<b>d</b>		Delete
<b>s</b>		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.

## 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 35** Required Keys

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

**Table 36** Optional Keys

Key-Value Pair	Description
<code>dateformat=format</code>	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 <code>dateformat</code> parameter accepts any time format that is supported by the UNIX <code>strftime</code> function.
<code>rate=transmitrate</code>	Specifies the rate to write files when latency mode is enabled.
<code>rampupsecs=seconds</code>	Specifies the number of seconds to linearly ramp up from 0 to the transmit rate that you request with <code>rate</code> . The default is 0.
<code>gdconfig=gdconfigfile</code>	Specifies the guaranteed delivery configuration file.
<code>transport=native   solace   tervela   rabbitmq   kafka</code>	Specifies the transport type. If you specify <code>solace</code> , <code>tervela</code> , <code>rabbitmq</code> , or <code>kafka</code> transports instead of the default <code>native</code> transport, then use the required client configuration files specified in the description of the C++ <code>C_dfESPpubsubSetPubsubLib()</code> API call.
<code>loglevel=trace   debug   info   warn   error   fatal   off</code>	Sets the logging level for the adapter. This is the same range of logging levels that you can set in the <code>C_dfESPpubsubInit()</code> <code>publish/subscribe</code> API call and in the engine <code>initialize()</code> call. The default level is <code>warn</code> .
<code>logconfigfile=logconfigfile</code>	Specifies the log configuration file.

Key-Value Pair	Description
<code>tokenlocation=location</code>	Specifies the location of the file in the local filesystem that contains the OAuth token required for authentication by the publish/subscribe server.
<code>configfilesection=[section]</code>	Specifies the name of the section in <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.config</code> (Windows) to parse for configuration parameters. Specify the value as <code>[configfilesection]</code> .
<code>latencymode=true   false</code>	When <code>true</code> , specifies latency mode.
<code>restartonerror= true   false</code>	When <code>true</code> , restarts the adapter if a fatal error is reported.
<code>transportconfigfile=file</code>	Specifies the publish/subscribe transport configuration file. For <code>solace</code> , the default is <code>./solace.cfg</code> . For <code>tervela</code> , the default is <code>./client.config</code> . For <code>rabbitmq</code> , the default is <code>./rabbitmq.cfg</code> . For <code>kafka</code> , the default is <code>./kafka.cfg</code> . <b>Note:</b> No transport configuration file is required for native transport.
<code>periodicity=seconds</code>	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
<code>maxfilesize=bytes</code>	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.
<code>aggrsize=statsforblocks</code>	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.
<code>hdatfilename=filename</code>	Specifies the filename to write to Hadoop Distributed File System (HDFS).
<code>hdatmaxdatanodes=numnodes</code>	Specifies the maximum number of data nodes. The default value is the number of live data nodes.
<code>hdfsblocksize=mbytes</code>	Specifies the Hadoop Distributed File System (HDFS) block size in MB.
<code>hdfsnumreplicas=number</code>	Specifies the Hadoop Distributed File System (HDFS) number of replicas. The default value is 1.
<code>hdatnumthreads=numthreads</code>	Specifies the adapter thread pool size. A value $\leq 1$ means that no data nodes are used.
<code>hdatmaxstringlength=size</code>	Specifies the fixed size to use for string fields in HDAT records. The value must be a multiple of 8.

Key-Value Pair	Description
<code>hdatlasrhostport=hostport</code>	Specifies the LASR server host and port.
<code>hdatlasrkey=path</code>	Specifies the path to <code>tklasrkey.sh</code> . By default, this file is located in <code>\$DFESP_HOME/bin</code>
<code>hdatcashostport=hostport</code>	Specifies the CAS server host and port.
<code>hdatcasusername=username</code>	Specifies the CAS server user name.
<code>hdatcaspassword=password</code>	Specifies the CAS server password.
<code>latencyblksize=numevents</code>	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 <code>latencymode</code> .
<code>unbufferedoutputstreams=true   false</code>	When <code>true</code> , specifies to create an unbuffered stream when writing to a file or socket. By default, streams are buffered.
<code>header=headerrow</code>	For a CSV subscriber, writes a header row that shows comma-separated field names. Set to "full" to include "opcode, flags," in header line.
<code>doubleprecision=number</code>	Specifies the number of fractional digits in the ASCII representation of a double. The default value is 6.
<code>hdatsecure=true   false</code>	When <code>true</code> , specifies to use SSH to connect to secure Hadoop plugins.
<code>hdatsecursshloginname=name</code>	When <code>hdatsecure=true</code> , specifies the SSH login name. Invalid if <code>hdatsecursshidentityfile</code> is not configured.
<code>hdatsecursshidentityfile=file</code>	When <code>hdatsecure=true</code> , specifies the SSH identity file. Invalid if <code>hdatsecursshloginname</code> is not configured.
<code>hdatsecurehadoopcmdpath=path</code>	When <code>hdatsecure=true</code> , specifies the full Hadoop command path on the remote name and data nodes. The default value is <code>"/opt/hadoop/bin/hadoop"</code> .

## Publisher Usage

Table 37 Required Keys

Key-Value Pairs	Description
<code>type=pub</code>	Specifies a publisher adapter.
<code>url=pubsubURL</code>	Specifies the standard URL in the form <code>dfESP://host:port/project/continuousquery/window</code> .
<code>fsname=output</code>	Specifies the publisher input file or socket "host:port". Leave <code>host</code> blank to implement a server.

Key-Value Pairs	Description
<code>fstype=binary   csv   xml   json   hdat   cef</code>	Specifies the file system type.

**Table 38** *Optional Keys*

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

Key-Value Pairs	Description
transportconfigfile=file	Specifies the publish/subscribe transport configuration file. For <code>solace</code> , the default is <code>./solace.cfg</code> . For <code>tervela</code> , the default is <code>./client.config</code> . For <code>rabbitmq</code> , the default is <code>./rabbitmq.cfg</code> . For <code>kafka</code> , the default is <code>./kafka.cfg</code> . <b>Note:</b> 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 <code>true</code> , specifies that events are transactional.
growinginputfile=true   false	When <code>true</code> , read from the growing file
prebuffer=true   false	When <code>true</code> , buffer all event blocks prior to publishing.
header=numlines	For CSV publisher, the number of header lines to ignore.
quiesceproject=true   false	When <code>true</code> , quiesce project after all events are injected into the Source window.
ignorecsvparseerrors=true   false	When <code>true</code> , drop event when a field fails CSV parsing and continue.
csvfielddelimiter=character	Specifies the character that delimits CSV fields, default = ,
noautogenfield=true   false	When <code>true</code> , input events are missing the key field autogenerated by the Source window.
addcsvopcode=true   false	When <code>true</code> , prepend opcode to input CSV events.
addcsvflags=none   normal   partialupdate	Specifies the event type to insert into input CSV event.
publishwithupsert=true   false	When <code>true</code> , build events with opcode = Upsert instead of insert.
cefsyslogprefix=true   false	When <code>true</code> , 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 <code>true</code> , strip all nulls from input CSV events.

## Publisher Failover

For information about implementing hot failover for publisher adapters, see [“Publisher Adapter Failover with Kafka”](#) in *SAS Event Stream Processing: Implementing Failover*.

## Using the HDAT Reader Adapter

SAS software provides a file format for Hadoop called SASHDAT, which makes files available for load to SAS LASR Analytic Server. The HDAT reader adapter converts each row in a SASHDAT file into an ESP event and injects event blocks into a Source window of an engine. Each event is built with an Insert opcode unless you specify the `publishwithupsert` key.

The adapter resides in `dfx-esp-hdatreader-adapter.jar`, which bundles the Java publisher SAS Event Stream Processing client.

The number of fields in the target Source window schema must match the number of columns in the SASHDAT row data. Also, all SASHDAT column types must be numeric, except for columns that correspond to an ESP field of type UTF8STR. In that case, the column must contain character data.

The source Hadoop Distributed File System (HDFS) and the name of the file within the file system are passed as required parameters to the adapter. The client target platform must define the environment variable `DFESP_HDFS_JARS`. This specifies the location of the Hadoop JAR files.

List the JAR files in this order: `hadoop-common-*.jar`, `hadoop-hdfs-*.jar`, *common hadoop JARs*.

For example, in Linux you would run this command to properly set the environment variable:

```
$ 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/*
```

### dfesp\_hdat\_publisher

`-C key1=val1,key2=val2,key3=val3,...`

**Table 39** Required Keys

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

**Table 40** Optional Keys

Key-Value Pair	Description
<code>blocksize=size</code>	Specifies the number of events per event block. The default value is 1.

Key-Value Pair	Description
<code>configfilesection=section</code>	Specifies the name of the section in <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/javaadapters.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\javaadapters.config</code> (Windows) to parse for configuration parameters.
<code>gdconfigfile=file</code>	Specifies the guaranteed delivery configuration file.
<code>pubsublib= native   solace   tervela   rabbitmq   kafka</code>	Specifies the transport type. The default is <code>native</code> .
<code>loglevel=severe   warning   info</code>	Specifies the application logging level. The default value is <code>warning</code> .
<code>tokenlocation=location</code>	Specifies the location of the file in the local file system that contains the OAuth token required for authentication by the publish/subscribe server.
<code>quiesceproject=true   false</code>	When true, quiesce the project after all events are injected into the Source window.
<code>publishwithupsert=true   false</code>	When true, build events with <code>opcode=Upsert</code> . By default, events are built with <code>opcode=Insert</code> . <b>javaadapters.config</b> parameter name: <code>publishwithupsert</code>
<code>transactional=true   false</code>	When true, event blocks are transactional. By default, they are normal.
<code>restartonerror=true   false</code>	When true, restarts the adapter when a fatal error is reported. <b>javaadapters.config</b> parameter name: <code>restartonerror</code>
<code>maxevents=number</code>	Specifies the maximum number of events to publish.
<code>transportconfigfile=file</code>	Specifies the full path to the publish/subscribe transport configuration file. The default file depends on the transport type specified with through the <code>pubsublib</code> parameter: For <code>solace</code> , the default file is <code>solace.cfg</code> . For <code>tervela</code> , the default file is <code>client.config</code> . For <code>rabbitmq</code> , the default file is <code>rabbitmq.cfg</code> . For <code>kafka</code> , the default file is <code>kafka.cfg</code> . <b>Note:</b> No transport configuration file is required for native transport.

For information about implementing hot failover for publisher adapters, see [“Publisher Adapter Failover with Kafka” in SAS Event Stream Processing: Implementing Failover](#).

## 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 SAS Event Stream Processing 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 41** Required Keys

Key-Value Pair	Description
<code>hdfs=target</code>	Specifies the target file system, in the form " <code>hdfs://host:port</code> ". Specifies the file system that is normally configured in property <code>fs.defaultFS</code> in <code>core-site.xml</code> .
<code>outputfile=file</code>	Specifies the output CSV file, in the form " <code>/path/filename.csv</code> ".
<code>url=url</code>	Specifies the event stream processing standard URL in the following format: " <code>dfESP://host:port/project/contquery/window&lt;?collapse=true   false&gt;</code> ".

Table 42 Optional Keys

Key-Value Pair	Description
<code>hdfsblocksize=size</code>	Specifies the HDFS block size in MB. The default is 64MB.
<code>configfilesection=section</code>	Specifies the name of the section in <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/javaadapters.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\javaadapters.config</code> (Windows) to parse for configuration parameters.
<code>dateformat=format</code>	Specifies the format of <code>DATE</code> and <code>STAMP</code> fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds ( <code>DATE</code> ) or microseconds ( <code>STAMP</code> ) since epoch. The <code>dateformat</code> parameter accepts any time format that is supported by the UNIX <code>strftime</code> function.
<code>gdconfigfile=file</code>	Specifies the guaranteed delivery configuration file for the client.
<code>jaasconf=file</code>	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.
<code>krb5conf=file</code>	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 <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/krb5.conf</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\krb5.conf</code> (Windows).
<code>pubsublib= native   solace   tervela   rabbitmq   kafka</code>	Specifies the transport type. The default value is <code>native</code> .
<code>maxfilesize=size</code>	Specifies the output file periodicity in bytes.
<code>hdfsnumreplicas=number</code>	Specifies the HDFS number of replicas. The default value is 1.
<code>loglevel=severe   warning   info</code>	Specifies the application logging level. The default value is <code>warning</code> .
<code>tokenlocation=location</code>	Specifies the location of the file in the local file system that contains the OAuth token required for authentication by the publish/subscribe server.
<code>periodicity=seconds</code>	Specifies the output file periodicity in seconds. Invalid if data is not insert-only.
<code>opaquestringfield=field</code>	Specifies the subscribed window field from which to extract the output line.
<code>restartonerror=true   false</code>	When true, restarts the adapter if a fatal error is reported.

Key-Value Pair	Description
<code>transportconfigfile=file</code>	<p>Specifies the full path to the publish/subscribe transport configuration file. The default file depends on the transport type specified with through the <code>pubsublib</code> parameter:</p> <p>For <code>solace</code>, the default file is <code>solace.cfg</code>.</p> <p>For <code>tervela</code>, the default file is <code>client.config</code>.</p> <p>For <code>rabbitmq</code>, the default file is <code>rabbitmq.cfg</code>.</p> <p>For <code>kafka</code>, the default file is <code>kafka.cfg</code>.</p> <p><b>Note:</b> No transport configuration file is required for native transport.</p>

Publisher usage:

**dfesp\_hdfs\_publisher** -C *key1=val1, key2=val2, key3=val3, ...*

**Table 43** Required Keys

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

**Table 44** Optional Keys

Key-Value Pair	Description
<code>addcsvopcode=true   false</code>	<p>When true, prepends an opcode and comma to input CSV events. The opcode is <code>Insert</code> unless <code>publishwithupsert</code> is true.</p> <p><b>javaadapters.config</b> parameter name: <code>addcsvopcode</code></p>
<code>addcsvflags=type</code>	Specifies the event type in insert into input CSV events. Valid values are <code>normal</code> and <code>partialupdate</code> .
<code>tokenlocation=location</code>	Specifies the location of the file in the local file system that contains the OAuth token required for authentication by the publish/subscribe server.
<code>quiesceproject=true   false</code>	When true, quiesce the project after all events are injected into the Source window.
<code>restartonerror=true   false</code>	<p>When true, restarts the adapter when a fatal error is reported.</p> <p><b>javaadapters.config</b> parameter name: <code>restartonerror</code></p>

Key-Value Pair	Description
<code>maxevents=number</code>	Specifies the maximum number of events to publish.
<code>transportconfigfile=file</code>	<p>Specifies the full path to the publish/subscribe transport configuration file. The default file depends on the transport type specified with through the <code>pubsublib</code> parameter:</p> <p>For <code>solace</code>, the default file is <code>solace.cfg</code>.</p> <p>For <code>tervela</code>, the default file is <code>client.config</code>.</p> <p>For <code>rabbitmq</code>, the default file is <code>rabbitmq.cfg</code>.</p> <p>For <code>kafka</code>, the default file is <code>kafka.cfg</code>.</p> <p><b>Note:</b> No transport configuration file is required for native transport.</p>
<code>blocksize=size</code>	Specifies the number of events per event block. The default value is 1.
<code>configfilesection=section</code>	Specifies the name of the section in <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/javaadapters.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\javaadapters.config</code> (Windows) to parse for configuration parameters.
<code>dateformat=format</code>	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 <code>dateformat</code> parameter accepts any time format that is supported by the UNIX <code>strftime</code> function.
<code>ignorecsvparseerrors=true   false</code>	<p>When true, drops an event when a field in an input CSV event cannot be parsed. An error is logged and publishing continues.</p> <p><b>javaadapters.config</b> parameter name: <code>ignorecsvparseerrors</code></p>
<code>gdconfigfile=file</code>	Specifies the guaranteed delivery configuration file.
<code>jaasconf=file</code>	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.
<code>krb5conf=file</code>	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 <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/krb5.conf</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\krb5.conf</code> (Windows).
<code>pubsublib= native   solace   tervela   rabbitmq   kafka</code>	Specifies the transport type. The default is <code>native</code> .
<code>csvfielddelimiter=delimiter</code>	Specifies the character delimiter for field data in input CSV events. The default delimiter is the <code>,</code> character.

Key-Value Pair	Description
<code>noautogenfield=true   false</code>	When true, input events are missing the key field that is autogenerated by the Source window. By default, events contain all schema fields.
<code>publishwithupsert=true   false</code>	When true, build events with <code>opcode=Upsert</code> . By default, events are built with <code>opcode=Insert</code> .  <code>javaadapters.config</code> parameter name: <code>publishwithupsert</code>
<code>transactional=true   false</code>	When true, event blocks are transactional. By default, they are normal.
<code>stripnullsfromcsv=true   false</code>	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”](#) in *SAS Event Stream Processing: Implementing Failover*.

## 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 Event Stream Processing 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 SAS Event Stream Processing 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 45** Required Keys

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

**Table 46** Optional Keys

Key-Value Pair	Description
<code>jndifactname=name</code>	Specifies the JNDI name to look up for the JMS connection factory. The default value is "ConnectionFactory". This option overrides settings in <code>jndi.properties</code> .
<code>configfilesection=section</code>	Specifies the name of the section in <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/javaadapters.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\javaadapters.config</code> (Windows) to parse for configuration parameters.
<code>dateformat=format</code>	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 <code>dateformat</code> parameter accepts any time format that is supported by the UNIX <code>strftime</code> function.
<code>pwdencrypted=true   false</code>	When true, the JMS password is encrypted. By default, it is not.
<code>protofile=file</code>	Specifies the .proto file that contains the message used for Google Protocol buffer support.
<code>gdconfigfile=filename</code>	Specifies the Guaranteed Delivery configuration file.
<code>jmsuserid=userid</code>	Specifies the user ID for the JMS destination.

Key-Value Pair	Description
<code>jndidestname=name</code>	Specifies the JNDI name to look up for the JMS destination. The default value is <code>"jmsDestination"</code> . This option overrides settings in <code>jndi.properties</code> .
<code>pubsublib= native   solace   tervela   rabbitmq   kafka</code>	Specifies the transport type. The default is <code>native</code> .
<code>csvmsgpereventblock= true   false</code>	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 <code>"line.separator"</code> system property.
<code>csvmsgperevent= true   false</code>	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 <code>"line.separator"</code> system property.
<code>loglevel=severe   warning   info</code>	Specifies the application logging level. The default value is <code>warning</code> .
<code>tokenlocation=location</code>	Specifies the location of the file in the local file system that contains the OAuth token required for authentication by the publish/subscribe server.
<code>jmspassword=password</code>	Specifies the password for the JMS destination.
<code>opaquestringfield=field</code>	Specifies the subscribed window field from which to extract the JMS <code>TextMessage</code> .
<code>restartonerror=true   false</code>	When true, restarts the adapter if a fatal error is reported.
<code>xmllib="axeda"   "turkcell"</code>	Specifies the JAR file derived from a third-party <code>.xsd</code> file that defines the XML contained in a JMS <code>TextMessage</code> . By default, the <code>TextMessage</code> contains CSV.
<code>transportconfigfile=file</code>	Specifies the full path to the publish/subscribe transport configuration file. The default file depends on the transport type specified with through the <code>pubsublib</code> parameter: For <code>solace</code> , the default file is <code>solace.cfg</code> . For <code>tervela</code> , the default file is <code>client.config</code> . For <code>rabbitmq</code> , the default file is <code>rabbitmq.cfg</code> . For <code>kafka</code> , the default file is <code>kafka.cfg</code> . <b>Note:</b> No transport configuration file is required for native transport.

#### Publisher Usage:

**dfesp\_jms\_publisher** -C *key1=val1, key2=val2, key3=val3...*

Table 47 Required Keys

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

Table 48 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 <code>solace.cfg</code> . For tervela, the default file is <code>client.config</code> . For rabbitmq, the default file is <code>rabbitmq.cfg</code> . For kafka, the default file is <code>kafka.cfg</code> . <b>Note:</b> 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 <code>jndi.properties</code> .
blocksize=number	Specifies the number of events per event block. The default value is 1.

Key-Value Pair	Description
<code>configfilesection=section</code>	Specifies the name of the section in <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/javaadapters.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\javaadapters.config</code> (Windows) to parse for configuration parameters.
<code>dateformat=format</code>	For messages of type <code>TextMessage</code> , specifies the format of <code>datetime</code> and <code>timestamp</code> fields. Specify a Java <code>SimpleDateFormat</code> pattern in quotation marks.
<code>ignorecsvparseerrors=true   false</code>	When true, drop an event and continue whenever a field fails CSV parsing.
<code>protofile=file</code>	Specifies the <code>.proto</code> file that contains the message used for Google Protocol buffer support.
<code>gdconfigfile=filename</code>	Specifies the Guaranteed Delivery configuration file for this publisher.
<code>jmsuserid=userid</code>	Specifies the user ID for the JMS destination.
<code>jndidestname=name</code>	Specifies the JNDI name to look up for the JMS destination. The default value is <code>"jmsDestination"</code> . This option overrides settings in <code>jndi.properties</code> .
<code>pubsublib= native   solace   tervela   rabbitmq   kafka</code>	Specifies the transport type. The default is <code>native</code> .
<code>csvfielddelimiter=delimiter</code>	Specifies the character delimiter for field data in input CSV events. The default delimiter is the <code>,</code> character.
<code>noautogenfield=true   false</code>	When true, input events are missing the key field that is autogenerated by the source window. By default, events contain all schema fields.
<code>loglevel=severe   warning   info</code>	Specifies the application logging level. The default value is <code>warning</code> .
<code>jmspassword=password</code>	Specifies the password for the JMS destination.
<code>opaquestring=true   false</code>	When true, do not parse <code>TextMessage</code> . Instead, copy complete message to a <code>ESP_UTF8STR</code> field.
<code>protomsg=message</code>	Specifies the message itself in the <code>.proto</code> file that is specified by the <code>protofile</code> parameter.
<code>publishwithupsert=true   false</code>	When true, build events with <code>opcode = upsert</code> . By default, <code>opcode=insert</code> .
<code>transactional=true   false</code>	When true, event blocks are transactional. By default, event blocks are normal.
<code>stripnullsfromcsv=true   false</code>	When true, strip all nulls from input CSV events.

Key-Value Pair	Description
<code>xmlib="axeda"   "turkcell"</code>	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” in SAS Event Stream Processing: Implementing Failover](#).

## Using the Kafka Connector and Adapter

### Overview

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.

### Using the Kafka Connector for Kafka 0.9 and MapR Streams

#### 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 or writing 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 SAS Event Stream Processing 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>.

**Table 49** Valid Library Versions

Library Version	SAS Event Stream Processing Release
1.0.0	6.1
0.9.3	4.3, 5.1, 5.2
0.9.2	4.2

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/>.

Make sure the LibrdKafka and Zookeeper libraries are built on the machine where you run the connector. This 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 SAS Event Stream Processing.

**Note:** The ESP 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 LibrdKafka 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 event stream processing publish/subscribe client plug-in libraries are available for C and Java publish/subscribe clients. These libraries enable a standard event stream processing publish/subscribe client application to exchange event blocks with an event stream processing server 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 event 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 event stream processing servers in the hot failover group. Thus, a standby connector becomes active when the active connector fails. All event stream processing servers 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 50 Required Parameters for the Kafka Subscriber Connector*

Parameter	Description
<code>type</code>	Specifies to subscribe. Must be "sub".

Parameter	Description
kafkahostport	Specifies one or more Kafka brokers in the following form: host:port,host:port,...
kafkatopic	Specifies the Kafka topic.
kafkpartition	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 <b>binary</b> , <b>csv</b> , <b>json</b> , <b>protobuf</b> , <b>avro</b> , 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 <i>false</i> . Subscriber clients reading messages sent by this subscriber can control their snapshot by configuring an appropriate initial offset into the Kafka partition.

**Table 51** Optional Parameters for the Kafka 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.
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 <b>.proto</b> 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 <b>protobuf</b> .
protomsg	Specifies the name of a Google Protocol Buffers message in the <b>.proto</b> 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 <b>protobuf</b> .

Parameter	Description
csvincludeschema	When <code>kafkatype=CSV</code> , specifies when to prepend output CSV data with the window's serialized schema. Valid values are <code>never</code> , <code>once</code> , and <code>preventblock</code> . 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 <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.config</code> (Windows) to parse for configuration parameters. Specify the value as <code>[configfilesection]</code> .
zookeeperhostport	Specifies the Zookeeper server in the form <code>host:port</code> .
kafkaglobalconfig	Specifies a semicolon-separated list of <code>key=value</code> strings to configure <code>librdkafka</code> global configuration values.
kafkatopicconfig	Specifies a semicolon-separated list of <code>key=value</code> strings to configure <code>librdkafka</code> 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 <code>kafkatype</code> is not set to <code>avro</code> .  See also " <a href="#">Publish/Subscribe API Support for Apache Avro Messaging</a> " in <i>SAS Event Stream Processing: Publish/Subscribe API</i> .
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 <code>avroschemaregistryurl</code> parameter. The default value is an Apache Avro schema that represents the subscribed window schema plus an <code>opcode</code> field. This definition is also copied to the schema registry. This parameter is ignored when <code>kafkatype</code> is not set to <code>avro</code> .
avroschemaname	Specifies the name of an Apache Avro schema to copy from the schema registry that is configured in the <code>avroschemaregistryurl</code> parameter. The default value is an Apache Avro schema that represents the subscribed window schema plus an <code>opcode</code> field. This name is copied to the schema registry. This parameter is ignored when <code>kafkatype</code> is not set to <code>avro</code> .
fieldtokafkakey	Specifies to copy the contents of the specified window string field to the Kafka message key. This parameter is ignored when <code>kafkatype</code> is not set to <code>protobuf</code> .

## Parameters for the Kafka Publisher Connector

**Table 52** Required Parameters for the Kafka Publisher Connector

Parameter	Description
<code>type</code>	Specifies to publish. Must be "pub".
<code>kafkahostport</code>	Specifies one or more Kafka brokers in the following form: <code>host:port,host:port,...</code>
<code>kafkatopic</code>	Specifies the Kafka topic.
<code>kafkpartition</code>	Specifies the Kafka partition. Specify -1 to use all partitions in the topic. This value is not supported when hot failover is enabled.
<code>kafkatype</code>	Specifies <b>binary</b> , <b>csv</b> , <b>json</b> , <b>protobuf</b> , <b>avro</b> , or <b>opaquestring</b> .
<code>urlhostport</code>	Specifies the <code>host:port</code> 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 53** Optional Parameters for the Kafka Publisher Connector

Parameter	Description
<code>transactional</code>	When <code>kafkatype=csv</code> or <code>kafkatype=opaquestring</code> , sets the event block type to transactional. The default event block type is normal.
<code>blocksize</code>	When <code>kafkatype=csv</code> or <code>kafkatype=opaquestring</code> , specifies the number of events to include in a published event block. The default value is 1.
<code>dateformat</code>	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 <code>dateformat</code> parameter accepts any time format that is supported by the UNIX <code>strftime</code> function.
<code>ignorecsvparseerrors</code>	Specifies that when a field in an input CSV event cannot be parsed, the event is dropped, an error is logged, and publishing continues.
<code>protofile</code>	Specifies the <code>.proto</code> 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 <code>protomsg</code> parameter.  This parameter is ignored when <code>kafkatype</code> is not set to <b>protobuf</b> .
<code>protomsg</code>	Specifies the name of a Google Protocol Buffers message in the <code>.proto</code> file that you specified with the <code>protofile</code> parameter. Event blocks are converted into this message.  This parameter is ignored when <code>kafkatype</code> is not set to <b>protobuf</b> .

Parameter	Description
configfilesection	Specifies the name of the section in <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.config</code> (Windows) to parse for configuration parameters. Specify the value as <code>[configfilesection]</code> .
csvfielddelimiter	Specifies the character delimiter for field data in input CSV events. The default delimiter is the <code>,</code> character.
noautogenfield	Specifies that input events are missing the key field that is autogenerated by the source window.
publishwithupsert	Specifies to build events with <code>opcode = Upsert</code> instead of <code>opcode = Insert</code> .
kafkainitialoffset	Specifies the offset from which to begin consuming messages from the Kafka topic and partition. Valid values are <code>smallest</code> , <code>largest</code> , or an integer. The default value is <code>smallest</code> .
addcsvopcode	Prepends an opcode and comma to write CSV events. The opcode is <code>Insert</code> unless <code>publishwithupsert</code> is enabled.
addcsvflags	Specifies the event type to insert into input CSV events (with a comma). Valid values are <code>normal</code> and <code>partialupdate</code> .
kafkaglobalconfig	Specifies a semicolon-separated list of <code>key=value</code> strings to configure <code>librdkafka</code> global configuration values.
kafkatopicconfig	Specifies a semicolon-separated list of <code>key=value</code> strings to configure <code>librdkafka</code> 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.
avroschemaregistryurl	Specifies the URL of the Apache Avro schema registry. This parameter is ignored when <code>kafkatype</code> is not set to <code>avro</code> .
keyfromkafkakey	Specifies to copy the Kafka message key to the window key field. This field must be <code>type=string</code> . This parameter is ignored when <code>kafkatype</code> is not set to <code>protobuf</code> .

## Using the Kafka Adapter for Kafka 0.9 and MapR Streams

### Overview

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

**Note:** The Kafka adapter is certified to work with the MapR converged data platform for publishing and subscribing.

**dfesp\_kafka\_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 54 Required Keys*

Key-Value Pair	Description
<code>type=sub</code>	Specifies a subscriber adapter.
<code>url=pubsubURL</code>	<p>Specifies the standard URL in the form <b>dfESP://host:port/project/continuousquery/window</b>.</p> <p>Append the following for subscribers: <code>?snapshot=false</code>.</p> <p><b>Note:</b> <code>?snapshot=true</code> is invalid.</p> <p>When <code>?snapshot=true</code>, 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.</p> <p>Append <code>?collapse=true   false</code> or <code>?rmretdel=true   false</code> for subscribers if needed.</p>
<code>kafkahostport=broker</code>	Specifies one or more Kafka brokers in the following form: <code>"host:port,host:port,..."</code> .
<code>kafkatopic=topic</code>	Specifies the Kafka topic.
<code>kafkpartition=partition</code>	Specifies the Kafka partition. Specify -1 to use all partitions in the topic. This value is not supported when hot failover is enabled.
<code>urlhostport=field</code>	Specifies the <code>host:port</code> field in the metadata topic to which the adapter subscribes (replace <code>:</code> with <code>'_'</code> )
<code>numbufferedmsgs=number</code>	Specifies the maximum number of messages buffered by a standby subscriber connector.
<code>kafkatype= binary   csv   json   protobuf   avro</code>	Specifies the message format. For subscribers, the name of a string field in the subscribed window schema is also supported.

Table 55 Optional Keys

Key-Value Pair	Description
<code>dateformat=format</code>	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 <code>dateformat</code> parameter accepts any time format that is supported by the UNIX <code>strftime</code> function.
<code>protofile=file</code>	Specifies the <code>.proto</code> file to be used for Google protocol buffer support. This key is ignored when <code>kafkatype</code> is not set to <code>protobuf</code> .
<code>protomsg=message</code>	Specifies the message itself in the <code>.proto</code> file that is specified by the <code>protofile</code> parameter. This key is ignored when <code>kafkatype</code> is not set to <code>protobuf</code> .
<code>gdconfig=file</code>	Specifies the guaranteed delivery configuration file.
<code>transport=native   solace   tervela   rabbitmq   kafka</code>	Specifies the transport type. If you specify <code>solace</code> , <code>tervela</code> , <code>rabbitmq</code> , or <code>kafka</code> transports instead of the default <code>native</code> transport, use the required client configuration files specified in the description of the C++ <code>C_dfESPpubsubSetPubsubLib()</code> API call.
<code>loglevel=trace   debug   info   warn   error   fatal   off</code>	Sets the logging level for the adapter. This is the same range of logging levels that you can set in the <code>C_dfESPpubsubInit()</code> <code>publish/subscribe</code> API call and in the engine <code>initialize()</code> call. The default level is <code>warn</code> .
<code>logconfigfile=file</code>	Specifies the log configuration file.
<code>tokenlocation=location</code>	Specifies the location of the file in the local filesystem that contains the OAuth token required for authentication by the <code>publish/subscribe</code> server.
<code>configfilesection=[section]</code>	Specifies the name of the section in <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.config</code> (Windows) to parse for configuration parameters. Specify the value as <code>[configfilesection]</code> .
<code>kafkaglobalconfig=list</code>	Specifies a semicolon-separated list of "key=value" strings to configure <code>librdkafka</code> global configuration values.
<code>kafkatopicconfig=list</code>	Specifies a semicolon-separated list of "key=value" strings to configure <code>librdkafka</code> topic configuration values.
<code>restartonerror=true   false</code>	When <code>true</code> , specifies to restart the adapter if a fatal error is reported.

Key-Value Pair	Description
<code>transportconfigfile=file</code>	Specifies the publish/subscribe transport configuration file. For <code>solace</code> , the default is <code>./solace.cfg</code> . For <code>tervela</code> , the default is <code>./client.config</code> . For <code>rabbitmq</code> , the default is <code>./rabbitmq.cfg</code> . For <code>kafka</code> , the default is <code>./kafka.cfg</code> . <b>Note:</b> No transport configuration file is required for native transport.
<code>csvincludeschema=never   once   pereventblock</code>	When <code>kafkatype=CSV</code> , specifies when to prepend output CSV data with the window's serialized schema. The default value is "never".
<code>csvmsgperevent=true   false</code>	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.
<code>csvmsgpereventblock=true   false</code>	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.
<code>doubleprecision=number</code>	Specifies the number of fractional digits in the ASCII representation of a double. The default value is 6.
<code>avroschemaregistryurl=url</code>	Specifies the URL of the Apache Avro schema registry. This parameter is ignored when <code>kafkatype</code> is not set to <code>avro</code> .
<code>avroschemadefinition=schema</code>	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 <code>avroschemaregistryurl</code> 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 <code>kafkatype</code> is not set to <code>avro</code> .
<code>avroschemaname=name</code>	Specifies the name of an Apache Avro schema to copy from the schema registry configured in the <code>avroschemaregistryurl</code> 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 <code>kafkatype</code> is not set to <code>avro</code> .
<code>fieldtokafkakey=fieldname</code>	Specifies to copy the contents of the specified window string field to the Kafka message key. This key is ignored when <code>kafkatype</code> is not set to <code>protobuf</code> .

## Publisher Usage

Table 56 Required Keys

Key-Value Pair	Description
<code>type=pub</code>	Specifies a publisher adapter

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

**Table 57** *Optional Keys*

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

Key-Value Pair	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 <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.config</code> (Windows) to parse for configuration parameters. Specify the value as <code>[configfilesection]</code> .
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 <code>./solace.cfg</code> . For tervela, the default is <code>./client.config</code> . For rabbitmq, the default is <code>./rabbitmq.cfg</code> . For kafka, the default is <code>./kafka.cfg</code> . <b>Note:</b> 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 <code>,</code> 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 <code>publishwithupsert</code> is true.
addcsvflags=none   normal   partialupdate	Specifies the event type to Insert into input CSV events (with comma).

Key-Value Pair	Description
<code>maxevents=number</code>	Specifies the maximum number of events to publish.
<code>quiesceproject=true   false</code>	When <code>true</code> , quiesces the project after all events are injected into the Source window.
<code>stripnullsfromcsv=true   false</code>	When <code>true</code> , strip all nulls from input CSV events.
<code>avroschemaregistryurl=url</code>	Specifies the URL of the Apache Avro schema registry. This parameter is ignored when <code>kafkatype</code> is not set to <code>avro</code> . See also <a href="#">“Functions to Support JSON Objects” in SAS Event Stream Processing: Publish/Subscribe API</a> .
<code>keyfromkafkakey=true   false</code>	When set to <code>true</code> , specifies to copy the Kafka message key to the window key field. This field must be <code>type=string</code> . This key is ignored when <code>kafkatype</code> is not set to <code>protobuf</code> .

## Publisher Failover

For information about implementing hot failover for publisher adapters, see [“Publisher Adapter Failover with Kafka” in SAS Event Stream Processing: Implementing Failover](#).

---

## Using the Kinesis Connector and Adapter

### 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 event stream processing 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 throughput. 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 58** Required Parameters for the Kinesis Subscriber Connector

Parameter	Description
<code>type</code>	Specifies to subscribe. Must be <code>sub</code> .
<code>kinesistype</code>	Specifies <code>binary</code> , <code>csv</code> , <code>json</code> , <code>protobuf</code> , <code>avro</code> , or the name of a string field in the subscribed window schema.
<code>snapshot</code>	Specifies whether to send snapshot data. When <code>true</code> , 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.
<code>streamname</code>	Specifies the Amazon Kinesis Data Stream name.
<code>awsregion</code>	Specifies the Amazon AWS region.
<code>partitionkey</code>	Specifies a string used to map output records to a specific shard. Configure <code>partitionkeyrange</code> to write to multiple shards.

**Table 59** Required Parameters for the Kinesis Publisher Connector

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

**Table 60** Optional Parameters for the Kinesis Subscriber Connector

Parameter	Description
<code>collapse</code>	Enables conversion of <code>UPDATE_BLOCK</code> 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 <code>.proto</code> 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 <code>protomsg</code> parameter. This parameter is ignored when <code>kinesistype</code> is not set to <code>protobuf</code> .
protomsg	Specifies the name of a Google Protocol Buffers message in the <code>.proto</code> file that you specified with the <code>protofile</code> parameter. Event blocks are converted into this message. This parameter is ignored when <code>kinesistype</code> is not set to <code>protobuf</code> .
csvincludeschema	When <code>kinesistype=csv</code> , specifies when to prepend output CSV data with the window's serialized schema. Valid values are <code>never</code> , <code>once</code> , and <code>pereventblock</code> . The default value is <code>never</code> .
configfilesection	Specifies the name of the section in <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.config</code> (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 <code>kinesistype</code> is not set to <code>avro</code> . See also <a href="#">"Functions to Support Apache Avro Objects" in SAS Event Stream Processing: Publish/Subscribe API</a> .
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 <code>avroschemaregistryurl</code> 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 <code>kinesistype</code> is not set to <code>avro</code> .
avroschemaname	Specifies the name of an Apache Avro schema to copy from the schema registry that is configured in the <code>avroschemaregistryurl</code> 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 <code>kinesistype</code> is not set to <code>avro</code> .
awsaccesskeyid	Specifies the AWS access key id credential. Required when <code>awssecretkey</code> is configured.

Parameter	Description
<code>awssecretkey</code>	Specifies the AWS secret key credential. Required when <code>awsaccesskeyid</code> is configured.
<code>partitionkeyrange</code>	Specifies the number of partition keys used when writing output records. The default value is 1.

**Table 61** Optional Parameters for the Kinesis Publisher Connector

Parameter	Description
<code>transactional</code>	When <code>kinesistype=csv</code> , sets the event block type to transactional. The default event block type is normal.
<code>blocksize</code>	When <code>kinesistype=csv</code> , specifies the number of events to include in a published event block. The default value is 1.
<code>dateformat</code>	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 <code>dateformat</code> parameter accepts any time format that is supported by the UNIX <code>strftime</code> function.
<code>ignorecsvparseerrors</code>	Specifies that when a field in an input CSV event cannot be parsed, the event is dropped, an error is logged, and publishing continues.
<code>protofile</code>	Specifies the <code>.proto</code> 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 <code>protomsg</code> parameter. This parameter is ignored when <code>kinesistype</code> is not set to <code>protobuf</code> .
<code>protomsg</code>	Specifies the name of a Google Protocol Buffers message in the <code>.proto</code> file that you specified with the <code>protofile</code> parameter. Event blocks are converted into this message. This parameter is ignored when <code>kinesistype</code> is not set to <code>protobuf</code> .
<code>configfilesection</code>	Specifies the name of the section in <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.config</code> (Windows) to parse for configuration parameters.
<code>csvfielddelimiter</code>	Specifies the character delimiter for field data in input CSV events. The default delimiter is the <code>,</code> character.
<code>noautogenfield</code>	Specifies that input events are missing the key field that is autogenerated by the source window.
<code>publishwithupsert</code>	Specifies to build events with <code>opcode = Upsert</code> instead of <code>opcode = Insert</code> .
<code>addcsvopcode</code>	Prepends an opcode and comma to write CSV events. The opcode is <code>Insert</code> unless <code>publishwithupsert</code> is enabled.
<code>addcsvflags</code>	Specifies the event type to insert into input CSV events (with a comma). Valid values are <code>normal</code> and <code>partialupdate</code> .

Parameter	Description
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 <code>kinesistype</code> is not set to <code>avro</code> . See also <a href="#">“Functions to Support Apache Avro Objects” in SAS Event Stream Processing: Publish/Subscribe API</a> .
awsaccesskeyid	Specifies the AWS access key id credential. Required when <code>awssecretkey</code> is configured.
awssecretkey	Specifies the AWS secret key credential. Required when <code>awsaccesskeyid</code> is configured.
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 <code>sharditeratortype</code> is set to <code>atsequencenumber</code> or <code>aftersequencenumber</code> .
shardtimestamp	Specifies the timestamp to use when <code>sharditeratortype</code> is set to <code>attimestamp</code> . See the <code>dateformat</code> parameter description for the required format.
maxrecordspershard	Specifies the maximum number of records to read from the shard per <code>GetRecords()</code> call. The default value is to read all available records.

## 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\"`).

Note:

### Subscriber Usage

Table 62 Required Keys

Key-Value Pair	Description
<code>type=sub</code>	Specifies a subscriber adapter.

Key-Value Pair	Description
<code>url=pubsubURL</code>	<p>Specifies the standard URL in the form <code>dfESP://host:port/project/continuousquery/window</code>.</p> <p>Append the following for subscribers: <code>?snapshot=true   false</code>. When <code>?snapshot=true</code>, 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.</p> <p>Append <code>?collapse=true   false</code> or <code>?rmretdel=true   false</code> for subscribers if needed.</p>
<code>kinesistype= binary   csv   json   protobuf   avro</code>	Specifies the message format. For subscribers, the name of a string field in the subscribed window schema is also supported.
<code>streamname=name</code>	Specifies the Amazon Kinesis Data Stream name.
<code>awsregion=region</code>	Specifies the Amazon AWS region.
<code>partitionkey=key</code>	Specifies a string used to map output records to a specific shard. Configure <code>partitionkeyrange</code> to write to multiple shards.

**Table 63** *Optional Keys*

Key-Value Pair	Description
<code>dateformat=format</code>	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 <code>dateformat</code> parameter accepts any time format that is supported by the UNIX <code>strftime</code> function.
<code>protofile=file</code>	Specifies the <code>.proto</code> file to be used for Google protocol buffer support. This key is ignored when <code>kinesistype</code> is not set to <code>protobuf</code> .
<code>protomsg=message</code>	Specifies the message itself in the <code>.proto</code> file that is specified by the <code>protofile</code> parameter. This key is ignored when <code>kinesistype</code> is not set to <code>protobuf</code> .
<code>gdconfig=file</code>	Specifies the guaranteed delivery configuration file.
<code>transport=native   solace   tervela   rabbitmq   kafka</code>	Specifies the transport type. If you specify <code>solace</code> , <code>tervela</code> , <code>rabbitmq</code> , or <code>kafka</code> transports instead of the default native transport, use the required client configuration files specified in the description of the C++ <code>C_dfESPpubsubSetPubsubLib()</code> API call.
<code>loglevel=trace   debug   info   warn   error   fatal   off</code>	Sets the logging level for the adapter. This is the same range of logging levels that you can set in the <code>C_dfESPpubsubInit()</code> publish/subscribe API call and in the engine <code>initialize()</code> call. The default level is <code>warn</code> .
<code>logconfigfile=file</code>	Specifies the log configuration file.
<code>tokenlocation=location</code>	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 Pair	Description
configfilesection=[section]	Specifies the name of the section in <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.config</code> (Windows) to parse for configuration parameters.
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 <code>./solace.config</code> . For Tervela, the default is <code>./client.config</code> . For RabbitMQ, the default is <code>./rabbitmq.config</code> . For Kafka, the default is <code>./kafka.config</code> . <b>Note:</b> No transport configuration file is required for native transport.
csvincludeschema=never   once   pereventblock	When <code>kinesistype=csv</code> , specifies when to prepend output CSV data with the window's serialized schema. The default value is <code>never</code> .
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 <code>kinesistype</code> is not set to <code>avro</code> . See also " <a href="#">Functions to Support Apache Avro Objects</a> " in <i>SAS Event Stream Processing: Publish/Subscribe API</i> .
awsaccesskeyid=id	Specifies the AWS access key ID credential. Required when <code>awssecretkey</code> is configured.
awssecretkey=key	Specifies the AWS secret key credential. Required when <code>awsaccesskeyid</code> 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 <code>avroschemaregistryurl</code> 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 <code>kinesistype</code> is not set to <code>avro</code> .
avroschemaname=name	Specifies the name of an Apache Avro schema to copy from the schema registry configured in the <code>avroschemaregistryurl</code> 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 <code>kinesistype</code> is not set to <code>avro</code> .
partitionkeyrange=number	Specifies the number of partition keys used when writing output records. The default value is 1.

## Publisher Usage

Table 64 Required Keys

Key-Value Pair	Description
type=pub	Specifies a publisher adapter
url=pubsubURL	Specifies the standard URL in the form <code>dfESP://host:port/project/continuousquery/window</code> .
kinesistype=binary   csv   json   protobuf   avro   opaquestring	Specifies the message format. <b>opaquestring</b> is valid only for publishers. For <b>opaquestring</b> , 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=trimhORIZON   latest   atsequencenumber   aftersequencenumber   attimestamp	Specifies the Amazon Kinesis shard iterator type.

Table 65 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 <code>dateformat</code> parameter accepts any time format that is supported by the UNIX <code>strftime</code> function.
protofile=file	Specifies the <code>.proto</code> file to be used for Google protocol buffer support. This key is ignored when <code>kinesistype</code> is not set to <b>protobuf</b> .
protomsg=message	Specifies the message itself in the <code>.proto</code> file that is specified by the <code>protofile</code> parameter. This key is ignored when <code>kinesistype</code> is not set to <b>protobuf</b> .
gdconfig=file	Specifies the guaranteed delivery configuration file.
transport=native   solace   tervela   rabbitmq   kafka	Specifies the transport type. If you specify <b>solace</b> , <b>tervela</b> , <b>rabbitmq</b> , or <b>kafka</b> transports instead of the default native transport, then use the required client configuration files specified in the description of the C++ <code>C_dfESPpubsubSetPubsubLib()</code> 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 <code>C_dfESPpubsubInit()</code> publish/subscribe API call and in the engine <code>initialize()</code> call. The default level is <b>warn</b> .
logconfigfile=file	Specifies the log configuration file.

Key-Value Pair	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 <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.config</code> (Windows) to parse for configuration parameters.
restartonerror=true   false	When <b>true</b> , 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 <code>./solace.cfg</code> . For Tervela, the default is <code>./client.config</code> . For RabbitMQ, the default is <code>./rabbitmq.cfg</code> . For Kafka, the default is <code>./kafka.cfg</code> . <b>Note:</b> No transport configuration file is required for native transport.
awsaccesskeyid=id	Specifies the AWS access key ID credential. Required when <code>awssecretkey</code> is configured.
awssecretkey=key	Specifies the AWS secret key credential. Required when <code>awsaccesskeyid</code> 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 <code>,</code> character.
noautogenfield=true   false	When <b>true</b> , specifies that input events are missing the key field that is autogenerated by the Source window.
publishwithupsert=true   false	When <b>true</b> , build events with <code>opcode = Upsert</code> instead of <code>Insert</code> .
addcsvopcode=true   false	When <b>true</b> , prepends an opcode and comma to write CSV events. The opcode is <code>Insert</code> unless <code>publishwithupsert</code> is <b>true</b> .
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 <b>true</b> , quiesces the project after all events are injected into the Source window.

Key-Value Pair	Description
<code>stripnullsfromcsv=true</code>   <code>false</code>	When <code>true</code> , strip all nulls from input CSV events.
<code>avroschemaregistryurl=url</code>	Specifies the URL of the Apache Avro schema registry. This parameter is ignored when <code>kinesistype</code> is not set to <code>avro</code> . See also <a href="#">“Functions to Support Apache Avro Objects” in SAS Event Stream Processing: Publish/Subscribe API</a> .
<code>shardids=list</code>	Specifies a comma separated list of shard IDs to read from. The default value is to read from all shards in the stream.
<code>shardsequencenumber=number</code>	Specifies the sequence number to use when <code>sharditeratortype</code> is set to <code>atsequencenumber</code> or <code>aftersequencenumber</code> .
<code>shardtimestamp=timestamp</code>	Specifies the timestamp to use when <code>sharditeratortype</code> is set to <code>attimestamp</code> . See the <code>dateformat</code> parameter description for the required format.
<code>maxrecordspershard=number</code>	Specifies the maximum number of records to read from the shard per <code>GetRecords()</code> call. The default value is to read all available records.

## Using the SAS LASR Analytic Server Adapter

The SAS LASR Analytic Server adapter supports publish and subscribe operations on the SAS LASR Analytic Server. To authenticate to the SAS LASR Analytic server, you can do one of the following:

- configure the adapter with a path to a script that invokes password-less SSH to the server
- configure the adapter to access SAS LASR Analytic Server authentication services

**Note:** If you do not configure the adapter to access SAS LASR Analytic Server authentication services:

- Running this adapter in a UNIX environment requires the installation of an SSH client.
- Running this adapter in a Microsoft Windows environment requires the installation of the SAS Event Stream Processing System Encryption and Authentication Overlay.
- The LASR adapter must be running on the same computer system as the SAS LASR Analytic Server under the following circumstances:
  - when the SAS LASR Analytic Server is running in SMP mode without SSH support
  - when the SAS LASR Analytic Server is running in SMP mode without SAS LASR Analytic Server authentication services support

When you start the adapter, you must be logged in as a user with permission to access the SAS LASR Analytic Server.

- The user running the adapter must be the same user that started the SAS LASR Analytic Server.

**Note:** Datetime and timestamp values that are mapped to a numeric column in a LASR table are automatically converted to and from SAS format (seconds since 1/1/1960).

**Note:** LASR table column names cannot exceed 32 characters. Ensure that your window schema complies with this restriction.

**Note:** To visualize data written by the subscriber adapter in Visual Analytics, you must register the LASR table in Visual Analytics after it has been created.

The subscriber writes rows to the LASR table by implementing an instance of the `com.sas.lasr.LASRPreparedAppender` class. The appender is flushed when the adapter is shut down, so the LASR table can be updated with additional rows at that time. While the adapter is running, there are 2 configuration parameters that control when the appender sends rows to the SAS LASR Analytic Server: `blocksize` and `autocommit`. The default values for these are `blocksize=0` and `autocommit=0` (disabled). So by default, every row is sent to the SAS LASR Analytic Server in real time. However, rows are not committed to the table until the appender is closed by the adapter. If the subscribed window is producing insert-only events (that is, no updates or deletes), the appender is not closed until the adapter is closed. Alternatively, you can configure `autocommit` to commit rows to the LASR table periodically based on number of seconds or number of rows.

If the SAS LASR Analytic Server adapter is configured to access LASR authentication services, then specify an `authpassword` in non-encrypted form. The encrypted version of the password can be generated using OpenSSL. The OpenSSL executable is included in the SAS Event Stream Processing System Encryption and Authentication Overlay installation. Use the following command on the console to use OpenSSL to display your encrypted password:

```
echo "authpassword" | openssl enc -e -aes-256-cbc -a -salt -pass pass:"espLASRadapterUsedByUser=authusername"
```

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

Subscriber Usage:

**dfesp\_lasr\_adapter** -C *key1=val1,key2=val2,key3=val3,...*

**Table 66** Required Keys

Key-Value Pair	Description
<code>lasrurl=url</code>	Specifies the SAS LASR Analytic Server connection URL in the form " <i>host:port</i> "
<code>url=url</code>	Specifies the event stream processing standard URL in the following format: " <code>dfESP://host:port/project/contquery/window?snapshot=true   false&lt;?collapse=true   false&gt;&lt;?rmretdel=true   false&gt;</code> ". <b>Note:</b> Multiple URLs, separated by commas, are permitted.
<code>type=sub</code>	Specifies a subscriber adapter.
<code>table=table</code>	Specifies the full name of the SAS LASR Analytic Server table. You must locate the server tag in the metadata properties for the table.

**Table 67** Optional Keys

Key-Value Pair	Description
<code>autocommit=number</code>	Specifies the number of observations to commit to the server. A value <code>&lt; 0</code> specifies the time in seconds between auto-commit operations. A value <code>&gt; 0</code> specifies the number of records that, after reached, forces an auto-commit operation. A value of 0 specifies no auto-commit operation. The default value is 0.

Key-Value Pair	Description
<code>configfilesection=section</code>	Specifies the name of the section in <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/javaadapters.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\javaadapters.config</code> (Windows) to parse for configuration parameters.
<code>pubsublib= native   solace   tervela   rabbitmq   kafka</code>	Specifies the transport type. The default value is <code>native</code> .
<code>pwdencrypted= true   false</code>	When true, the SAS LASR Analytic Server password is encrypted. By default, it is not.
<code>tokenlocation=location</code>	Specifies the location of the file in the local file system that contains the OAuth token required for authentication by the publish/subscribe server.
<code>nostrictschema= true   false</code>	When true, the adapter does not observe strict SAS LASR Analytic Server adapter schema. By default, strict schema is observed. <b>javaadapters.config</b> parameter name: <code>nostrictschema</code>
<code>stimeout=seconds</code>	<code>stimeout=seconds</code>
<code>authurl=url</code>	Specifies the URL for SAS LASR Analytic Server authentication services.
<code>restartonerror= true   false</code>	When true, restarts the adapter when a fatal error is reported. <b>javaadapters.config</b> parameter name: <code>restartonerror</code>
<code>tklasrkey=path</code>	Specifies the path to <code>tklasrkey.sh</code> for Linux or <code>tklasrkey.bat</code> for Microsoft Windows. By default, these files are located in <code>\$DFESP_HOME/bin</code> .
<code>authusername=username</code>	Specifies the user name for SAS LASR Analytic Server authentication services.
<code>authpassword=password</code>	Specifies the password for SAS LASR Analytic Server authentication services.
<code>obstoanalyze=number</code>	Specifies the number of observations to analyze in order to define the output SAS LASR Analytic Server structure. The default value is 4096. When you specify <code>-s</code> , this option is ignored.
<code>blocksize=size</code>	Specifies the buffer size (number of observations) to be flushed to the server.
<code>dateformat=format</code>	Specifies the format of <code>DATE</code> and <code>STAMP</code> fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds ( <code>DATE</code> ) or microseconds ( <code>STAMP</code> ) since epoch. The <code>dateformat</code> parameter accepts any time format that is supported by the UNIX <code>strftime</code> function.
<code>gdconfigfile=filename</code>	Specifies the guaranteed delivery configuration file for the client.

Key-Value Pair	Description
loglevel=severe   warning   info	Specifies the application logging level. The default level is warning.
newtable= true   false	When true, the adapter creates a new LASR table. By default, the adapter assumes a LASR table already exists. <b>javaadapters.config</b> parameter name: newtable
schema=schema	Specifies the explicit SAS LASR Analytic Server schema in the form <i>rowname1:sastype1:sasformat1:label1,...,rownameN:sastypeN:sasformatN:labelN</i> . <b>Note:</b> When you omit the <i>sasformat</i> , no SAS format is applied to the row. When you omit the <i>label</i> , no label is applied to the row. If you previously specified <code>nostrictschema=true</code> , this key is ignored.
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 <code>pubsublib</code> parameter: For <code>solace</code> , the default file is <code>solace.cfg</code> . For <code>tervela</code> , the default file is <code>client.config</code> . For <code>rabbitmq</code> , the default file is <code>rabbitmq.cfg</code> . For <code>kafka</code> , the default file is <code>kafka.cfg</code> . <b>Note:</b> No transport configuration file is required for native transport.
bufsize=size	Specifies the socket buffer size. By default, the size is 16MB.

#### Publisher Usage:

**dfesp\_lasr\_adapter** -C *key1=val1, key2=val2, key3=val3, ...*

**Table 68** Required Keys

Key-Value Pair	Description
lasrurl=url	Specifies the SAS LASR Analytic Server connection URL in the form " <i>host:port</i> "
url=url	Specifies the event stream processing standard URL in the following format: " <i>dfESP://host:port/project/contquery/window</i> ". <b>Note:</b> Multiple URLs, separated by commas, are permitted.
type=pub	Specifies a publisher adapter.
table=table	Specifies the full name of the SAS LASR Analytic Server table. You must locate the server tag in the metadata properties for the table.

Table 69 Optional Keys

Key-Value Pair	Description
<code>configfilesection=name</code>	Specifies the name of the section in <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/javaadapters.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\javaadapters.config</code> (Windows) to parse for configuration parameters.
<code>fetchallcached=true   false</code>	When true, enables caching as data is fetched from a table. By default, caching is disabled. <b>javaadapters.config</b> parameter name: <code>fetchallcached</code>
<code>pubsublib= native   solace   tervela   rabbitmq   kafka</code>	Specifies the transport type. By default, the type is <code>native</code> .
<code>pwdencrypted= true   false</code>	When true, the SAS LASR Analytic Server password is encrypted. By default, it is not.
<code>tokenlocation=location</code>	Specifies the location of the file in the local file system that contains the OAuth token required for authentication by the publish/subscribe server.
<code>cachedaction="action"</code>	Specifies the action to get cached results from the LASR table. For example, specify <code>cachedaction="fetch field1 / to=100 where field2=2"</code> .
<code>nostrictschema= true   false</code>	When true, the adapter does not observe strict SAS LASR Analytic Server adapter schema. By default, strict schema is observed. <b>javaadapters.config</b> parameter name: <code>nostrictschema</code>
<code>stimeout=seconds</code>	<code>stimeout=seconds</code>
<code>authurl=url</code>	Specifies the URL for SAS LASR Analytic Server authentication services.
<code>restartonerror= true   false</code>	When true, restarts the adapter when a fatal error is reported. <b>javaadapters.config</b> parameter name: <code>restartonerror</code>
<code>maxevents=number</code>	Specifies the maximum number of events to publish.
<code>tklasrkey=path</code>	Specifies the path to <code>tklasrkey.sh</code> for Linux or <code>tklasrkey.bat</code> for Microsoft Windows. By default, these files are located in <code>\$DFESP_HOME/bin</code> .
<code>authusername=username</code>	Specifies the user name for SAS LASR Analytic Server authentication services.
<code>authpassword=password</code>	Specifies the password for SAS LASR Analytic Server authentication services.
<code>blocksize=size</code>	Specifies the buffer size (number of observations) to be flushed to the server.

Key-Value Pair	Description
<code>quiesceproject=true   false</code>	When true, quiesces the project after all events are injected into the Source window. <b>javaadapters.config</b> parameter name: <code>quiesceproject</code>
<code>dateformat=format</code>	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 <code>dateformat</code> parameter accepts any time format that is supported by the UNIX <code>strftime</code> function.
<code>transactional=true   false</code>	When true, specifies that event blocks are transactional. By default, event blocks are normal. <b>javaadapters.config</b> parameter name: <code>transactional</code>
<code>gdconfigfile=filename</code>	Specifies the guaranteed delivery configuration file for the client.
<code>loglevel=severe   warning   info</code>	Specifies the application logging level. The default level is warning.
<code>action="action"</code>	Specifies the action to get cached results from the LASR table. For example, specify <code>action="fetch field1 / to=100 where field2=2"</code> .
<code>publishwithupsert=true   false</code>	When true, builds events with <code>opcode=Upsert</code> . By default, events are built with <code>opcode=Insert</code> . <b>javaadapters.config</b> parameter name: <code>publishwithupsert</code>
<code>transportconfigfile=file</code>	Specifies the full path to the publish/subscribe transport configuration file. The default file depends on the transport type specified with through the <code>pubsublib</code> parameter: For <code>solace</code> , the default file is <code>solace.cfg</code> . For <code>tervela</code> , the default file is <code>client.config</code> . For <code>rabbitmq</code> , the default file is <code>rabbitmq.cfg</code> . For <code>kafka</code> , the default file is <code>kafka.cfg</code> . <b>Note:</b> No transport configuration file is required for native transport.
<code>bufsize=size</code>	Specifies the socket buffer size. By default, the size is 16MB.

For information about implementing hot failover for publisher adapters, see [“Publisher Adapter Failover with Kafka” in SAS Event Stream Processing: Implementing Failover](#).

## Using the Modbus Connector and Adapter

### 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 SAS Event Stream Processing 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 70 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 SAS Event Stream Processing. The Modbus subscriber connector reads data from SAS Event Stream Processing 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/>.

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/>.

The Modbus publisher connector polls the Modbus server for data and publishes the values to a Source window on a running ESP 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 ESP 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 SAS Event Stream Processing, 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 71** Required Parameters for the Modbus Publisher Connector

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

**Table 72** Optional Parameters for the Modbus Publisher Connector

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

**Table 73** Required Parameters for the Modbus Subscriber Connector

Parameter	Description
<code>type</code>	Specifies to subscribe. Must be <code>sub</code> .
<code>modbus</code>	Specifies the Modbus server host. Instead of a host name, you can specify a port with the form <code>host:port</code> . If no port is specified, the default is 502.

Parameter	Description
snapshot	If true, pulls a snapshot from the window at start-up.

**Table 74** Optional Parameters for the Modbus Subscriber Connector

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

## Using the Modbus Adapter

### Overview

The Modbus adapter supports publish and subscribe operations between SAS Event Stream Processing and a Modbus server.

In publisher mode, the adapter reads data from Modbus and publishes it to a Source window on a running ESP 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 ESP 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 75** Required Keys

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

Table 76 Optional Keys

Key-Value Pair	Description
<code>restartonerror=true   false</code>	When <code>true</code> , restarts the adapter if a fatal error is reported.
<code>gdconfig=gdconfigfile</code>	Specifies the guaranteed delivery configuration file.
<code>transport=native   solace   tervela   rabbitmq   kafka</code>	Specifies the transport type. If you specify <code>solace</code> , <code>tervela</code> , <code>rabbitmq</code> , or <code>kafka</code> transports instead of the default <code>native</code> transport, use the required client configuration files specified in the description of the C++ <code>C_dfESPpubsubSetPubsubLib()</code> API call.
<code>transportconfigfile=file</code>	Specifies the publish/subscribe transport configuration file. For <code>solace</code> , the default is <code>./solace.cfg</code> . For <code>tervela</code> , the default is <code>./client.config</code> . For <code>rabbitmq</code> , the default is <code>./rabbitmq.cfg</code> . For <code>kafka</code> , the default is <code>./kafka.cfg</code> . <b>Note:</b> No transport configuration file is required for native transport.
<code>tokenlocation=location</code>	Specifies the location of the file in the local filesystem that contains the OAuth token required for authentication by the publish/subscribe server.
<code>configfilesection=[section]</code>	Specifies the name of the section in <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.config</code> (Windows) to parse for configuration parameters. Specify the value as <code>[configfilesection]</code> .

## Publisher Usage

Table 77 Required Keys

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

Table 78 Optional Keys

Key-Value Pairs	Description
<code>restartonerror=true   false</code>	When <code>true</code> , restarts the adapter if a fatal error is reported.
<code>gdconfig=gdconfigfile</code>	Specifies the guaranteed delivery configuration file.
<code>transport=native   solace   tervela   rabbitmq   kafka</code>	Specifies the transport type. If you specify <code>solace</code> , <code>tervela</code> , <code>rabbitmq</code> or <code>kafka</code> transports instead of the default <code>native</code> transport, then use the required client configuration files specified in the description of the C++ <code>C_dfESPpubsubSetPubsubLib()</code> API call.
<code>transportconfigfile=file</code>	Specifies the publish/subscribe transport configuration file. For <code>solace</code> , the default is <code>./solace.cfg</code> . For <code>tervela</code> , the default is <code>./client.config</code> . For <code>rabbitmq</code> , the default is <code>./rabbitmq.cfg</code> . For <code>kafka</code> , the default is <code>./kafka.cfg</code> . <b>Note:</b> No transport configuration file is required for native transport.
<code>tokenlocation=location</code>	Specifies the location of the file in the local filesystem that contains the OAuth token required for authentication by the publish/subscribe server.
<code>configfilesection=[section]</code>	Specifies the name of the section in <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.config</code> (Windows) to parse for configuration parameters. Specify the value as <code>[configfilesection]</code> .
<code>blocksize=size</code>	Specifies the event block size. The default value is 1.
<code>maxevents=maxevents</code>	Specifies the maximum number of events to publish.
<code>interval=seconds</code>	Specifies the Modbus polling interval, in seconds. The default is 10.
<code>coils=coils</code>	Specifies the Modbus coils to read.
<code>inputs=inputs</code>	Specifies the Modbus inputs to read.
<code>inputRegisters=registers</code>	Specifies the Modbus input registers to read.
<code>holdingRegisters=hregisters</code>	Specifies the Modbus holding registers to read.

## Using the MQTT Connector and Adapter

### 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 ESP 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 SAS Event Stream Processing 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 79** Required Parameters for Subscriber MQTT Connectors

Parameter	Description
<code>type</code>	Specifies to subscribe. Must be “sub”.

Parameter	Description
snapshot	Specifies whether to send snapshot data. When <code>true</code> , 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.
mqtttthost	Specifies the MQTT server host name.
mqtttclientid	Specifies the string to use as the MQTT Client ID. If NULL, a random client ID is generated. If NULL, <code>mqtttdonotcleansession</code> must be <code>false</code> . Must be unique among all clients connected to the MQTT server.
mqttttopic	Specifies the string to use as an MQTT topic to publish events to.
mqtttqos	Specifies the requested Quality of Service. Values can be 0, 1 or 2.
mqtttmsgtype	Specifies <code>binary</code> , <code>csv</code> , <code>json</code> , <code>protobuf</code> , or the name of a string field in the subscribed window schema.

**Table 80** Required Parameters for Publisher MQTT Connectors

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

**Table 81** Optional Parameters for Publisher MQTT Connectors

Parameter	Description
mqtttuserid	Specifies the user name required to authenticate the connector's session with the MQTT server.
mqtttpassword	Specifies the password associated with <code>mqtttuserid</code> .

Parameter	Description
<code>mqttport</code>	Specifies the MQTT server port. Default is 1883.
<code>mqttacceptretainedmsg</code>	Sets to true to accept to receive retained message. The default is "false".
<code>mqttcleansession</code>	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".
<code>mqttkeepaliveinterval</code>	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.
<code>publishwithupsert</code>	Builds events with <code>opcode=Upsert</code> instead of <code>Insert</code> .
<code>transactional</code>	When <code>mqttmsgtype=CSV</code> , sets the event block type to transactional. The default event block type is normal.
<code>blocksize</code>	When <code>mqttmsgtype=CSV</code> , specifies the number of events to include in a published event block. The default value is 1.
<code>ignorecsvparseerrors</code>	Specifies that when a field in an input CSV event cannot be parsed, the event is dropped, an error is logged, and publishing continues.
<code>csvfielddelimiter</code>	Specifies the character delimiter for field data in input CSV events. The default delimiter is the <code>,</code> character.
<code>noautogenfield</code>	Specifies that input events are missing the key field that is automatically generated by the Source window.
<code>dateformat</code>	Specifies the format of <code>DATE</code> and <code>STAMP</code> fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds ( <code>DATE</code> ) or microseconds ( <code>STAMP</code> ) since epoch. The <code>dateformat</code> parameter accepts any time format that is supported by the UNIX <code>strftime</code> function.
<code>protofile</code>	Specifies the <code>.proto</code> file that contains the Google Protocol Buffers message definition used to convert event blocks to <code>protobuf</code> messages. When you specify this parameter, you must also specify the <code>protomsg</code> parameter.  This parameter is ignored when <code>mqttmsgtype</code> is not set to <code>protobuf</code> .
<code>protomsg</code>	Specifies the name of a Google Protocol Buffers message in the <code>.proto</code> file that you specified with the <code>protofile</code> parameter. Event blocks are converted into this message.  This parameter is ignored when <code>mqttmsgtype</code> is not set to <code>protobuf</code> .
<code>mqttssl</code>	Specifies to use TLS to connect to the MQTT broker. The default is "false". In order to use TLS, the SAS Event Stream Processing encryption overlay must be installed.
<code>mqttsslcafile</code>	If <code>mqttssl=true</code> , specifies the path to a file containing the PEM encoded trusted CA certificate files. Either <code>mqttsslcafile</code> or <code>mqttsslcapath</code> must be specified.

Parameter	Description
<code>mqtsslcapath</code>	If <code>mqtssl=true</code> , specifies the path to a directory containing the PEM encoded trusted CA certificate files. See <code>mosquitto.conf</code> for more details about configuring this directory. Either <code>mqtsslcafile</code> or <code>mqtsslcapath</code> must be specified.
<code>mqtsslcertfile</code>	If <code>mqtssl=true</code> , specifies the path to a file containing the PEM encoded certificate file for this client. Both <code>mqtsslcertfile</code> and <code>mqtsslkeyfile</code> must be provided if one of them is.
<code>mqtsslkeyfile</code>	If <code>mqtssl=true</code> , specifies the path to a file containing the PEM encoded private key for this client. Both <code>mqtsslcertfile</code> and <code>mqtsslkeyfile</code> must be provided if one of them is.
<code>mqtsslpassword</code>	If <code>mqtssl=true</code> , and if key file is encrypted, specifies the password for decryption.
<code>configfilesection</code>	Specifies the name of the section in <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.config</code> (Windows) to parse for configuration parameters. Specify the value as <code>[configfilesection]</code> .
<code>mqtpasswordencrypted</code>	Specifies that <code>mqtpassword</code> is encrypted.
<code>maxevents</code>	Specifies the maximum number of events to publish.
<code>stripnullsfromcsv</code>	Strip all nulls from input CSV events.

**Table 82** *Optional Parameters for Subscriber MQTT Connectors*

Parameter	Description
<code>mqtuserid</code>	Specifies the user name required to authenticate the connector's session with the MQTT server.
<code>mqtpassword</code>	Specifies the password associated with <code>mqtuserid</code> .
<code>mqtport</code>	Specifies the MQTT server port. The default is 1883.
<code>mqtretainmsg</code>	Sets to true to make the published message retained in the MQTT Server. The default is "false".
<code>mqttdonotcleansession</code>	Instructs the MQTT Server to keep all messages and subscriptions on disconnect, instead of keeping them. The default is "false".
<code>mqtkeepaliveinterval</code>	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.
<code>mqtmsgmaxdeliveryattempts</code>	Specifies the number of times the connector tries to resend the message in case of failure. The default is 20.

Parameter	Description
<code>mqtmsgdelaydeliveryattempts</code>	Specifies the delay in milliseconds between delivery attempts specified with <code>mqtmsgmaxdeliveryattempts</code> . The default is 500.
<code>mqtmsgwaitbeforere retry</code>	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.
<code>mqtmaxinflightmsg</code>	Specifies the number of QoS 1 and 2 messages that can be simultaneously in flight. The default is 20.
<code>collapse</code>	Enables conversion of UPDATE_BLOCK events to make subscriber output publishable.
<code>rmretdel</code>	Specifies to remove all delete events from event blocks received by a subscriber that were introduced by a window retention policy.
<code>dateformat</code>	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 <code>dateformat</code> parameter accepts any time format that is supported by the UNIX <code>strptime</code> function.
<code>protofile</code>	Specifies the <code>.proto</code> file that contains the Google Protocol Buffers message definition used to convert event blocks to <code>protobuf</code> messages. When you specify this parameter, you must also specify the <code>protomsg</code> parameter.  This parameter is ignored when <code>mqtmsgtype</code> is not set to <code>protobuf</code> .
<code>protomsg</code>	Specifies the name of a Google Protocol Buffers message in the <code>.proto</code> file that you specified with the <code>protofile</code> parameter. Event blocks are converted into this message.  This parameter is ignored when <code>mqtmsgtype</code> is not set to <code>protobuf</code> .
<code>mqtssl</code>	Specifies to use TLS to connect to the MQTT broker. The default is "false". In order to use TLS, the SAS Event Stream Processing encryption overlay must be installed.
<code>mqtsslcafile</code>	If <code>mqtssl=true</code> , specifies the path to a file containing the PEM encoded trusted CA certificate files. Either <code>mqtsslcafile</code> or <code>mqtsslcapath</code> must be specified.
<code>mqtsslcapath</code>	If <code>mqtssl=true</code> , specifies the path to a directory containing the PEM encoded trusted CA certificate files. See <code>mosquitto.conf</code> for more details about configuring this directory. Either <code>mqtsslcafile</code> or <code>mqtsslcapath</code> must be specified.
<code>mqtsslcertfile</code>	If <code>mqtssl=true</code> , specifies the path to a file containing the PEM encoded certificate file for this client. Both <code>mqtsslcertfile</code> and <code>mqtsslkeyfile</code> must be provided if one of them is.
<code>mqtsslkeyfile</code>	If <code>mqtssl=true</code> , specifies the path to a file containing the PEM encoded private key for this client. Both <code>mqtsslcertfile</code> and <code>mqtsslkeyfile</code> must be provided if one of them is.

Parameter	Description
<code>mqttsslpassword</code>	If <code>mqttssl=true</code> , and if key file is encrypted, specifies the password for decryption.
<code>csvincludeschema</code>	Specifies <code>never</code> , <code>once</code> , or <code>preventblock</code> . The default value is <code>never</code> . When <code>mqttmsgtype = CSV</code> , prepend output CSV with the window's serialized schema.
<code>configfilesection</code>	Specifies the name of the section in <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.config</code> (Windows) to parse for configuration parameters. Specify the value as <code>[configfilesection]</code> .
<code>mqttpasswordencrypted</code>	Specifies that <code>mqttpassword</code> is encrypted.
<code>addcsvopcode</code>	Prepends an opcode and comma to input CSV events. The opcode is <code>insert</code> unless <code>publishwithupsert</code> is enabled.
<code>addcsvflags</code>	Specifies the event type to insert into input CSV events (with a comma). Valid values are <code>normal</code> and <code>partialupdate</code> .
<code>csvmsgperevent</code>	For CSV, specifies to send one message per event. The default is one message per transactional event block or else one message per event.
<code>csvmsgpereventblock</code>	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.
<code>doubleprecision</code>	Specifies the number of fractional digits in the ASCII representation of a double. The default value is 6.

## 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\"`).

**Note:** The MQTT adapter is not supported on Windows systems.

## Subscriber Usage

Table 83 Required Keys

Key-Value Pair	Description
<code>type=sub</code>	Specifies a subscriber adapter.
<code>url=pubsubURL</code>	Specifies the standard URL in the form <code>dfESP://host:port/project/continuousquery/window</code> . Append <code>?snapshot=true</code>   <code>false</code> for subscribers. Append <code>?collapse=true</code>   <code>false</code> or <code>?rmretdel=true</code>   <code>false</code> or both for subscribers if needed.
<code>mqtttthost=hostname</code>	Specifies the MQTT server host name.
<code>mqttttopic=topic</code>	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).
<code>mqtttclientid=string</code>	Specifies the string to use as the MQTT Client ID. If NULL, a random client ID is generated. If NULL, <code>mqtttdonotcleansession</code> must be <code>false</code> . Must be unique among all clients connected to the MQTT server.
<code>mqtttqos=0</code>   <code>1</code>   <code>2</code>	Specifies the requested Quality of Service.
<code>mqtttmsgtype=binary</code>   <code>csv</code>   <code>json</code>   <code>protobuf</code>   <code>opaquestringfield</code>	Specifies the message format or the name of a string field in the subscribed window schema.

Table 84 Optional Keys

Key-Value Pair	Description
<code>mqtttuserid=username</code>	Specifies the user name required to authenticate the session with the MQTT server.
<code>mqtttpassword=password</code>	Specifies the password associated with <code>mqtttuserid</code> .
<code>mqtttport=port</code>	Specifies the MQTT server port. The default is 1883.
<code>mqtttdonotcleansession=true</code>   <code>false</code>	When <code>true</code> , instructs the MQTT Server to keep all messages and subscriptions on disconnect (instead of cleaning them). The default is "false".
<code>mqtttkeepaliveinterval=seconds</code>	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.
<code>mqtttssl=true</code>   <code>false</code>	When <code>true</code> , uses Transport Layer Security (TLS) to connect to the MQTT broker. The default is "false". In order to use TLS, the SAS ESP encryption overlay must be installed.

Key-Value Pair	Description
<code>mqttslcafile=path</code>	When <code>mqttsl=true</code> , specifies the path to a file containing the PEM encoded trusted CA certificate files. Either <code>mqttslcafile</code> or <code>mqttslcapath</code> must be specified.
<code>mqttslcapath=path</code>	When <code>mqttsl=true</code> , specifies the path to a file containing the PEM encoded trusted CA certificate files. Either <code>mqttslcafile</code> or <code>mqttslcapath</code> must be specified.  See <code>mosquitto.conf</code> for more information about configuring this directory.
<code>mqttslcertfile=path</code>	When <code>mqttsl=true</code> , specifies the path to a file containing the encoded certificate file for this client. Both <code>mqttslcertfile</code> and <code>mqttslkeyfile</code> must be provided if one of them is.
<code>mqttslkeyfile=path</code>	When <code>mqttsl=true</code> , specifies the path to a file containing the encoded private key for this client. Both <code>mqttslcertfile</code> and <code>mqttslkeyfile</code> must be provided if one of them is.
<code>mqttslpassword=password</code>	When <code>mqttsl=true</code> and <code>mqttslkeyfile</code> is encrypted, specifies the password for decryption.
<code>dateformat=format</code>	Specifies the format of <code>DATE</code> and <code>STAMP</code> fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds ( <code>DATE</code> ) or microseconds ( <code>STAMP</code> ) since epoch. The <code>dateformat</code> parameter accepts any time format that is supported by the UNIX <code>strftime</code> function.
<code>protofile=file</code>	Specifies the <code>.proto</code> file to be used for Google protocol buffer support.  This key is ignored when <code>mqtmsgtype</code> is not set to <code>protobuf</code> .
<code>protomsg=message</code>	Specifies the message itself in the <code>.proto</code> file that is specified by the <code>protofile</code> parameter.  This key is ignored when <code>mqtmsgtype</code> is not set to <code>protobuf</code> .
<code>gdconfig=file</code>	Specifies the guaranteed delivery configuration file.
<code>transport=native   solace   tervela   rabbitmq   kafka</code>	Specifies the transport type. If you specify <code>solace</code> , <code>tervela</code> , <code>rabbitmq</code> , or <code>kafka</code> transports instead of the default <code>native</code> transport, then use the required client configuration files specified in the description of the C++ <a href="#">C_dfESPpubsubSetPubsubLib()</a> API call.
<code>loglevel=trace   debug   info   warn   error   fatal   off</code>	Sets the logging level for the adapter. This is the same range of logging levels that you can set in the <code>C_dfESPpubsubInit()</code> <code>publish/subscribe</code> API call and in the engine <code>initialize()</code> call. The default level is <code>warn</code> .
<code>logconfigfile=file</code>	Specifies the log configuration file.
<code>tokenlocation=location</code>	Specifies the location of the file in the local file system that contains the OAuth token that is required for authentication by the <code>publish/subscribe</code> server.

Key-Value Pair	Description
<code>configfilesection=[section]</code>	Specifies the name of the section in <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.config</code> (Windows) to parse for configuration parameters. Specify the value as <code>[configfilesection]</code> .
<code>mqttpasswordencrypted=true   false</code>	When <code>true</code> , <code>mqttpassword</code> is encrypted
<code>restartonerror=true   false</code>	When <code>true</code> , specifies to restart the adapter if a fatal error is reported.
<code>transportconfigfile=file</code>	Specifies the publish/subscribe transport configuration file. For <code>solace</code> , the default is <code>./solace.cfg</code> . For <code>tervela</code> , the default is <code>./client.config</code> . For <code>rabbitmq</code> , the default is <code>./rabbitmq.cfg</code> . For <code>kafka</code> , the default is <code>./kafka.cfg</code> . <b>Note:</b> No transport configuration file is required for native transport.
<code>mqttretainmsg=true   false</code>	When <code>true</code> , make the published message retained in the MQTT Server. The default is “ <code>false</code> ”.
<code>mqttmsgmaxdeliveryattempts=number</code>	Specifies the number of times to try to resend the message in case of failure. The default is 20.
<code>mqttmsgdelaydeliveryattempts=milliseconds</code>	Specifies the delay in milliseconds between delivery attempts specified in <code>mqttmsgmaxdeliveryattempts</code> . The default is 500.
<code>mqttmsgwaitbeforere retry=seconds</code>	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.
<code>mqttmaxinflightmsg=number</code>	Specifies the number of QoS 1 and 2 messages that can be simultaneously in flight. The default is 20.
<code>csvincludeschema=never   once   pereventblock</code>	When <code>mqttmsgtype=CSV</code> , specifies when to pass window schema to CSV subscriber callback. The default value is “ <code>never</code> ”.
<code>csvmsgperevent=true   false</code>	When <code>true</code> , send one message per event. By default, one message is sent per transactional event.
<code>csvmsgpereventblock=true   false</code>	When <code>true</code> , send one message per event block. By default, one message is sent per transactional event block.
<code>doubleprecision=number</code>	Specifies the number of fractional digits in the ASCII representation of a double. The default value is 6.

## Publisher Usage

Table 85 Required Keys

Key-Value Pair	Description
<code>type=pub</code>	Specifies a publisher adapter.
<code>url=pubsubURL</code>	Specifies the standard URL in the form <code>dfESP://host:port/project/continuousquery/window</code> .
<code>mqtthost= hostname</code>	Specifies the MQTT server host name.
<code>mqtttopic=topic</code>	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).
<code>mqttclientid=string</code>	Specifies the string to use as the MQTT Client ID. If NULL, a random client ID is generated. If NULL, <code>mqttdontcleansession</code> must be <code>false</code> . Must be unique among all clients connected to the MQTT server.
<code>mqttqos=0   1   2</code>	Specifies the requested Quality of Service.
<code>mqttmsgtype=binary   csv   json   protobuf   opaquestringfield</code>	Specifies the message format. When the Source window schema has three fields, the value of <code>opaquestringfield</code> is assumed to be <code>"index:int64,topic:string,message:string"</code> . When the Source window schema has two fields, the value of <code>opaquestringfield</code> is assumed to be <code>"index:int64,message:string"</code> .

Table 86 Optional Keys

Key-Value Pair	Description
<code>mqttuserid=username</code>	Specifies the user name required to authenticate the session with the MQTT server.
<code>mqttpassword=password</code>	Specifies the password associated with <code>mqttuserid</code> .
<code>mqttport=port</code>	Specifies the MQTT server port. The default is 1883.
<code>mqttdontcleansession=true   false</code>	When <code>true</code> , instructs the MQTT Server to keep all messages and subscriptions on disconnect (instead of cleaning them). The default is <code>"false"</code> .
<code>mqttkeepaliveinterval=seconds</code>	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
<code>mqttssl=true   false</code>	When <code>true</code> , uses TLS to connect to the MQTT broker. The default is “false”. In order to use TLS, the SAS Event Stream Processing encryption overlay must be installed.
<code>mqttsslcafile=path</code>	When <code>mqttssl=true</code> , specifies the path to a file containing the PEM encoded trusted CA certificate files. Either <code>mqttsslcafile</code> or <code>mqttsslcapath</code> must be specified.
<code>mqttsslcapath=path</code>	When <code>mqttssl=true</code> , specifies the path to a file containing the PEM encoded trusted CA certificate files. Either <code>mqttsslcafile</code> or <code>mqttsslcapath</code> must be specified.  See <code>mosquitto.conf</code> for more information about configuring this directory.
<code>mqttsslcertfile=path</code>	When <code>mqttssl=true</code> , specifies the path to a file containing the encoded certificate file for this client. Both <code>mqttsslcertfile</code> and <code>mqttsslkeyfile</code> must be provided if one of them is.
<code>mqttsslkeyfile=path</code>	When <code>mqttssl=true</code> , specifies the path to a file containing the encoded private key for this client. Both <code>mqttsslcertfile</code> and <code>mqttsslkeyfile</code> must be provided if one of them is.
<code>mqttsslpassword=password</code>	When <code>mqttssl=true</code> and <code>mqttsslkeyfile</code> is encrypted, specifies the password for decryption.
<code>dateformat=format</code>	Specifies the format of <code>DATE</code> and <code>STAMP</code> fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds ( <code>DATE</code> ) or microseconds ( <code>STAMP</code> ) since epoch. The <code>dateformat</code> parameter accepts any time format that is supported by the UNIX <code>strftime</code> function.
<code>protofile=file</code>	Specifies the <code>.proto</code> file to be used for Google protocol buffer support.  This key is ignored when <code>mqttmsgtype</code> is not set to <code>protobuf</code> .
<code>protomsg=message</code>	Specifies the message itself in the <code>.proto</code> file that is specified by the <code>protofile</code> parameter.  This key is ignored when <code>mqttmsgtype</code> is not set to <code>protobuf</code> .
<code>gdconfig=file</code>	Specifies the guaranteed delivery configuration file.
<code>transport=native   solace   tervela   rabbitmq   kafka</code>	Specifies the transport type. If you specify <code>solace</code> , <code>tervela</code> , <code>rabbitmq</code> , or <code>kafka</code> transports instead of the default <code>native</code> transport, then use the required client configuration files specified in the description of the C++ <code>C_dfESPpubsubSetPubsubLib()</code> API call.
<code>loglevel=trace   debug   info   warn   error   fatal   off</code>	Sets the logging level for the adapter. This is the same range of logging levels that you can set in the <code>C_dfESPpubsubInit()</code> <code>publish/subscribe</code> API call and in the engine <code>initialize()</code> call. The default level is <code>warn</code> .
<code>logconfigfile=file</code>	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 <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.config</code> (Windows) to parse for configuration parameters. Specify the value as <code>[configfilesection]</code> .
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 <code>./solace.cfg</code> . For tervela, the default is <code>./client.config</code> . For rabbitmq, the default is <code>./rabbitmq.cfg</code> . For kafka, the default is <code>./kafka.cfg</code> . <b>Note:</b> 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=delimiter	Specifies the character delimiter for field data in input CSV events. The default delimiter is the <code>,</code> 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 <code>publishwithupsert</code> 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.

Key-Value Pair	Description
<code>quiesceproject=true   false</code>	When <code>true</code> , quiesces the project after all events are injected into the Source window.
<code>stripnullsfromcsv=true   false</code>	When <code>true</code> , strip all nulls from input CSV events.

## Publisher Failover

For information about implementing hot failover for publisher adapters, see [“Publisher Adapter Failover with Kafka”](#) in *SAS Event Stream Processing: Implementing Failover*.

## Using the Nurego Connector

Nurego is a cloud-based analytics and automation solution for subscription businesses. The Nurego connector is subscriber-only, and intended for use with a [Metering window](#). When subscribed to the Metering window, the connector builds a REST request for each subscribed event. It forwards the request to a Nurego server through the HTTPS protocol.

The primary use case for the Nurego connector is to periodically provide usage information (number of events processed) to the Nurego server. Because the REST request format is Nurego specific, this connector is not general purpose.

Specify the URL of the Nurego service that fields the REST request with the mandatory configuration parameter `serviceurl`. The `serviceurl` must be set to the Nurego REST API endpoint that accepts single-usage reporting: `https://api.nurego.com/v1/subscriptions/subscription_id/usage`

The format and contents of the JSON payload in the REST request is as follows:

```
{
  "provider": "cloud-foundry",
  "feature_id": "num_events",
  "amount": "<number of events>",
  "usage_date": "<usage_date>"
}
```

The connector builds and sends this JSON request for every event that is generated by the Metering window. (By default the Metering window generates an event every five seconds). If the metering interval has been reconfigured to a value greater than 30 seconds, then the Nurego connector logs an error and halts at start-up.

**Table 87** Required Parameters of the Nurego Connector

Parameter	Description
<code>type</code>	Specifies to subscribe. Must be “sub”.
<code>snapshot</code>	Specifies whether to send snapshot data. When <code>true</code> , 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.
<code>serviceurl</code>	Specifies the target Nurego REST service URL.

<code>certificate</code>	Specifies the full path and filename of the client certificate that is used to establish the HTTPS connection to the Nurego REST service.
<code>username</code>	Specifies the user name to use in requests to Nurego for a new token.
<code>password</code>	Specifies the password to use in requests to Nurego for a new token.
<code>instanceid</code>	Specifies the instance ID to use in requests to Nurego for a new token

**Table 88** *Optional Parameters of the Nurego Connector*

Parameter	Description
<code>certpassword</code>	Specifies the password associated with the client certificate that is configured in certificate.
<code>collapse</code>	Enables conversion of UPDATE_BLOCK events to make subscriber output publishable.
<code>rmretdel</code>	Specifies to remove all delete events from event blocks received by a subscriber that were introduced by a window retention policy
<code>configfilesection</code>	Specifies the name of the section in <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.config</code> (Windows) to parse for configuration parameters. Specify the value as <code>[configfilesection]</code> .

## Using the OPC-UA Connector and Adapter

### 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 89** *Supported Mappings for a Publisher:*

ESP 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 90** *Supported Mappings for a Subscriber:*

ESP 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 91** 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 <code>opc.tcp://</code> ).
snapshot	Specifies whether to send snapshot data.  When <code>true</code> , 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 92** Optional Parameters for Publisher OPC-UA Connectors

Parameter	Description
opcuanamespaceuri	Specifies the OPC-UA server namespace URI. The default is the namespace at <code>index=0</code> .
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 ESP window schema fields, in the form <code>&lt;identifier type&gt;_&lt;identifier&gt;</code> . 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".

Parameter	Description
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 <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.config</code> (Windows) to parse for configuration parameters. Specify the value as <code>[configfilesection]</code> .
publishwithupsert	Builds events with <code>opcode=Upsert</code> instead of <code>Insert</code> .
maxevents	Specifies the maximum number of events to publish.
opcuanodeiddelimiter	Specifies the character used to delimit fields in the <code>opcuanodeids</code> parameter. The default value is <code>'</code> .

**Table 93** *Optional Parameters for Subscriber OPC-UA Connectors*

Parameter	Description
opcuanamespaceuri	Specifies the OPC-UA server namespace URI. The default is the namespace at <code>index=0</code> .
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 ESP window schema fields, in the form <code>&lt;identifier type&gt;_&lt;identifier&gt;</code> . 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 <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.config</code> (Windows) to parse for configuration parameters. Specify the value as <code>[configfilesection]</code> .
opcuanodeiddelimiter	Specifies the character used to delimit fields in the <code>opcuanodeids</code> parameter. The default value is <code>'</code> .

## 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 94** Required Keys

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

**Table 95** Optional Keys

Key-Value Pair	Description
<code>opcuanamespaceuri=uri</code>	Specifies the OPC-UA server namespace URI. The default is the namespace at <code>index=0</code> .
<code>opcuausername=username</code>	Specifies OPC-UA user name. By default, there is none.
<code>opcuapassword=password</code>	Specifies the OPC-UA password. By default, there is none.
<code>opcuanodeids=list</code>	Specifies a comma-separated list of Node IDs to map to ESP window schema fields, in the form <code>&lt;identifier type&gt;_&lt;identifier&gt;</code> . The default assumes that window field names are in Node ID form.
<code>gdconfig=list</code>	Specifies the guaranteed delivery configuration file.
<code>transport=native   solace   tervela   rabbitmq   kafka</code>	Specifies the transport type. If you specify <code>solace</code> , <code>tervela</code> , <code>rabbitmq</code> , or <code>kafka</code> transports instead of the default <code>native</code> transport, then use the required client configuration files specified in the description of the C++ <code>C_dfESPpubsubSetPubsubLib()</code> 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 <code>C_dfESPPubsubInit()</code> publish/subscribe API call and in the engine <code>initialize()</code> call. The default level is <code>warn</code> .
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 <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.config</code> (Windows) to parse for configuration parameters. Specify the value as <code>[configfilesection]</code> .
restartonerror=true   false	When <code>true</code> , specifies to restart the adapter if a fatal error is reported.
transportconfigfile=file	Specifies the publish/subscribe transport configuration file. For <code>solace</code> , the default is <code>./solace.cfg</code> . For <code>tervela</code> , the default is <code>./client.config</code> . For <code>rabbitmq</code> , the default is <code>./rabbitmq.cfg</code> . For <code>kafka</code> , the default is <code>./kafka.cfg</code> . <b>Note:</b> No transport configuration file is required for native transport.
opcuanodeiddelimiter=character	Specifies the character used to delimit fields in the <code>opcuanodeids</code> parameter. The default value is <code>'\'</code> .

## Publisher Usage

Table 96 Required Keys

Key-Value Pair	Description
type=pub	Specifies a publisher adapter
url=pubsubURL	Specifies the standard URL in the form <code>dfESP://host:port/project/continuousquery/window</code> .
opcuaendpoint=endpoint	Specifies the OPC-UA server endpoint, without the <code>opc.tcp://</code> prefix.

Table 97 Optional Keys

Key-Value Pair	Description
<code>opcuanamespaceuri=uri</code>	Specifies the OPC-UA server namespace URI. The default is the namespace at <code>index=0</code> .
<code>opcuausername=username</code>	Specifies OPC-UA user name. By default, there is none.
<code>opcuapassword=password</code>	Specifies the OPC-UA password. By default, there is none.
<code>opcuanodeids=list</code>	Specifies a comma-separated list of Node IDs to map to ESP window schema fields, in the form <code>&lt;identifier type&gt;_&lt;identifier&gt;</code> . The default assumes that window field names are in Node ID form.
<code>gdconfig=file</code>	Specifies the guaranteed delivery configuration file.
<code>transport=native   solace   tervela   rabbitmq   kafka</code>	Specifies the transport type. If you specify <code>solace</code> , <code>tervela</code> , <code>rabbitmq</code> , or <code>kafka</code> transports instead of the default <code>native</code> transport, then use the required client configuration files specified in the description of the C++ <code>C_dfESPpubsubSetPubsubLib()</code> API call.
<code>loglevel=trace   debug   info   warn   error   fatal   off</code>	Sets the logging level for the adapter. This is the same range of logging levels that you can set in the <code>C_dfESPpubsubInit()</code> <code>publish/subscribe</code> API call and in the engine <code>initialize()</code> call. The default level is <code>warn</code> .
<code>logconfigfile=file</code>	Specifies the log configuration file.
<code>tokenlocation=location</code>	Specifies the location of the file in the local file system that contains the OAuth token that is required for authentication by the <code>publish/subscribe</code> server.
<code>configfilesection=[section]</code>	Specifies the name of the section in <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.config</code> (Windows) to parse for configuration parameters. Specify the value as <code>[configfilesection]</code> .
<code>restartonerror=true   false</code>	When <code>true</code> , specifies to restart the adapter if a fatal error is reported.
<code>transportconfigfile=file</code>	Specifies the <code>publish/subscribe</code> transport configuration file. For <code>solace</code> , the default is <code>./solace.cfg</code> . For <code>tervela</code> , the default is <code>./client.config</code> . For <code>rabbitmq</code> , the default is <code>./rabbitmq.cfg</code> . For <code>kafka</code> , the default is <code>./kafka.cfg</code> . <b>Note:</b> No transport configuration file is required for native transport.
<code>opcuanodeiddelimiter=character</code>	Specifies the character used to delimit fields in the <code>opcuanodeids</code> parameter. The default value is <code>'</code> .

Key-Value Pair	Description
<code>blocksize=size</code>	Sets the block size. The default value is 1.
<code>transactional=true   false</code>	When <code>true</code> , events are transactional.
<code>publishwithupsert=true   false</code>	When <code>true</code> , build events with opcode = Upsert instead of Insert.
<code>publishinterval=seconds</code>	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.
<code>maxevents=number</code>	Specifies the maximum number of events to publish.
<code>quiesceproject=true   false</code>	When <code>true</code> , 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” in SAS Event Stream Processing: Implementing Failover](#).

---

## Using the PI Connector and Adapter

### 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
<i>elementindex</i>	Value of the connector <i>afelement</i> configuration parameter. Specify either an AF element name or an AF element template name.
<i>element</i>	Name of the AF element associated with the <i>value</i> .
<i>attribute</i>	Name of the AF attribute owned by the AF <i>element</i> .
<i>value</i>	Value of the AF <i>attribute</i> .
<i>timestamp</i>	Timestamp associated with the <i>value</i> .
<i>status</i>	Status associated with the <i>value</i> .

**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:

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
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 event stream processing data types to PI point mappings are as follows:

Event Stream Processor Data Type	PI Point
ESP_INT32	PIPointType::Int16 PIPointType::Int32

Event Stream Processor Data Type	PI Point
ESP_DOUBLE	PIPointType::Float16 PIPointType::Float32
ESP_TIMESTAMP	PIPointType::Timestamp
ESP_UTF8STR	PIPointType::String

Use the following parameters with PI connectors:

**Table 98** 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 99** 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 100** 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.

Parameter	Description
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 <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.config</code> (Windows) to parse for configuration parameters. Specify the value as <code>[configfilesection]</code> .

**Table 101** 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 <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.config</code> (Windows) to parse for configuration parameters. Specify the value as <code>[configfilesection]</code> .
publishwithupsert	Builds events with <code>opcode=Upsert</code> instead of <code>Insert</code> .
maxevents	Specifies the maximum number of events to publish.

Parameter	Description
<code>allvaluestostrings</code>	Specifies to convert all received values to a string. Requires that the value field in the Source window have <code>type = ESP_UTF8STR</code> .

## 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 102** Required Keys

Key-Value Pair	Description
<code>type=sub</code>	Specifies a subscriber adapter.
<code>url=pubsubURL</code>	Specifies the standard URL in the form <b>dfESP://host:port/project/continuousquery/window</b> . Append <code>?snapshot=true   false</code> for subscribers. Append <code>?collapse=true   false</code> or <code>?rmretdel=true   false</code> or both for subscribers if needed.
<code>afelement=element</code>	Specifies the AF element or element template name. Wildcards are supported.

**Table 103** Optional Keys

Key-Value Pair	Description
<code>iselementtemplate=true   false</code>	When <code>true</code> , specifies that the value of <code>afelement</code> is the name of an element template.
<code>pisystem=name</code>	Specifies the name of the PI system.
<code>afdatabase=name</code>	Specifies the name of the Asset Framework database.
<code>afrootelement=element</code>	Specifies a root element in the Asset Framework hierarchy from which to search for <code>afelement</code> .

Key-Value Pair	Description
<code>afattribute=attribute</code>	Specifies an attribute in <code>afelement</code> and ignores other attributes there.
<code>gdconfig=file</code>	Specifies the guaranteed delivery configuration file.
<code>transport=native   solace   tervela   rabbitmq   kafka</code>	Specifies the transport type. If you specify <code>solace</code> , <code>tervela</code> , <code>rabbitmq</code> , or <code>kafka</code> transports instead of the default <code>native</code> transport, then use the required client configuration files specified in the description of the C++ <code>C_dfESPPubsubSetPubsubLib()</code> API call.
<code>loglevel=trace   debug   info   warn   error   fatal   off</code>	Sets the logging level for the adapter. This is the same range of logging levels that you can set in the <code>C_dfESPPubsubInit()</code> <code>publish/subscribe</code> API call and in the engine <code>initialize()</code> call. The default level is <code>warn</code> .
<code>loglevel=trace   debug   info   warn   error   fatal   off</code>	Sets the logging level for the adapter. This is the same range of logging levels that you can set in the <code>C_dfESPPubsubInit()</code> <code>publish/subscribe</code> API call and in the engine <code>initialize()</code> call. The default level is <code>warn</code> .
<code>tokenlocation=location</code>	Specifies the location of the file in the local file system that contains the OAuth token that is required for authentication by the <code>publish/subscribe</code> server.
<code>configfilesection=[section]</code>	Specifies the name of the section in <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.config</code> (Windows) to parse for configuration parameters. Specify the value as <code>[configfilesection]</code> .
<code>restartonerror=true   false</code>	When <code>true</code> , specifies to restart the adapter if a fatal error is reported.
<code>transportconfigfile=file</code>	Specifies the <code>publish/subscribe</code> transport configuration file. For <code>solace</code> , the default is <code>./solace.cfg</code> . For <code>tervela</code> , the default is <code>./client.config</code> . For <code>rabbitmq</code> , the default is <code>./rabbitmq.cfg</code> . For <code>kafka</code> , the default is <code>./kafka.cfg</code> . <b>Note:</b> No transport configuration file is required for native transport.

## Publisher Usage

**Table 104** Required Keys

Key-Value Pair	Description
<code>type=pub</code>	Specifies a publisher adapter.

Key-Value Pair	Description
<code>url=pubsubURL</code>	Specifies the standard URL in the form <code>dfESP://host:port/project/continuousquery/window</code> .
<code>afelement=element</code>	Specifies the AF element or element template name. Wildcards are supported.

**Table 105** Optional Keys

Key-Value Pair	Description
<code>iselementtemplate=true   false</code>	When <code>true</code> , specifies that the value of <code>afelement</code> is the name of an element template.
<code>pisystem=name</code>	Specifies the name of the PI system.
<code>afdatabase=name</code>	Specifies the name of the Asset Framework database.
<code>afrootelement=element</code>	Specifies a root element in the Asset Framework hierarchy from which to search for <code>afelement</code> .
<code>afattribute=attribute</code>	Specifies an attribute in <code>afelement</code> and ignores other attributes there.
<code>gdconfig=file</code>	Specifies the guaranteed delivery configuration file.
<code>transport=native   solace   tervela   rabbitmq   kafka</code>	Specifies the transport type. If you specify <code>solace</code> , <code>tervela</code> , <code>rabbitmq</code> , or <code>kafka</code> transports instead of the default <code>native</code> transport, then use the required client configuration files specified in the description of the C++ <code>C_dfESPpubsubSetPubsubLib()</code> API call.
<code>loglevel=trace   debug   info   warn   error   fatal   off</code>	Sets the logging level for the adapter. This is the same range of logging levels that you can set in the <code>C_dfESPpubsubInit()</code> <code>publish/subscribe</code> API call and in the engine <code>initialize()</code> call. The default level is <code>warn</code> .
<code>logconfigfile=file</code>	Specifies the log configuration file.
<code>tokenlocation=location</code>	Specifies the location of the file in the local file system that contains the OAuth token that is required for authentication by the <code>publish/subscribe</code> server.
<code>configfilesection=[section]</code>	Specifies the name of the section in <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.config</code> (Windows) to parse for configuration parameters. Specify the value as <code>[configfilesection]</code> .
<code>restartonerror=true   false</code>	When <code>true</code> , 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 <code>./solace.cfg</code> . For tervela, the default is <code>./client.config</code> . For rabbitmq, the default is <code>./rabbitmq.cfg</code> . For kafka, the default is <code>./kafka.cfg</code> . <b>Note:</b> No transport configuration file is required for native transport.
archivestamp=timestamp	Specifies that when connecting to the AF server, retrieve all archived values from the specified timestamp onwards. Timestamp is in the form "%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 <code>value</code> field in the Source window have <code>type = ESP_UTF8STR</code> .

## Publisher Failover

For information about implementing hot failover for publisher adapters, see [“Publisher Adapter Failover with Kafka” in SAS Event Stream Processing: Implementing Failover](#).

## 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 event stream processing 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 106** Required Parameters for the Project Publish Connector

Parameter	Description
type	Specifies to publish. Must be “pub”.
srcproject	Specifies the name of the source project.

Parameter	Description
<code>srccontinuousquery</code>	Specifies the name of the source continuous query.
<code>srcwindow</code>	Specifies the name of the Source window.

**Table 107** *Optional Parameters for the Project Publish Connector*

Parameter	Description
<code>maxevents</code>	Specifies the maximum number of events to publish.
<code>configfilesection</code>	Specifies the name of the section in <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.config</code> (Windows) to parse for configuration parameters. Specify the value as <code>[configfilesection]</code> .

## Using the Pylon Publisher Connector and Adapter

### 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 108** Required Parameters for Pylon Connectors

Parameter	Description
<code>type</code>	Specifies to publish. Must be <code>pub</code> .

**Table 109** Optional Parameters for Pylon Connectors

Parameter	Description
<code>cameraipaddress</code>	Specifies the camera IP address. The default value is the address of the first camera found on the local subnet.
<code>maxnumframes</code>	Specifies the maximum number of frames to publish. The default value is no maximum.
<code>maxframerate</code>	Specifies the maximum number of frames per second to publish. The default value is the rate at which frames are received from the camera.
<code>camerafeaturesfile</code>	Specifies a Pylon Features Stream (*.pfs) configuration file to load. The default is to use the current camera configuration unmodified.
<code>camerawidth</code>	Specifies the Area-Of-Interest width. The default value is the value in the current camera configuration.
<code>cameraheight</code>	Specifies the Area-Of-Interest height. The default value is the value in the current camera configuration.
<code>camerapixelformat</code>	Specifies the image pixel format. The default value is the format in the current camera configuration.
<code>camerapacketsize</code>	Specifies the Ethernet packet size. The default value is the value in the current camera configuration.
<code>transactional</code>	Sets the event block type to transactional. The default event block type is normal.
<code>configfilesection</code>	Specifies the name of the section in <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.config</code> (Windows) to parse for configuration parameters. Specify the value as <code>[configfilesection]</code> .

Parameter	Description
<code>publishwithupsert</code>	Builds events with <code>opcode=Upsert</code> instead of <code>opcode=Insert</code> .
<code>cameraxoffset</code>	Specifies the Area-Of-Interest horizontal offset. The default value is the value in the current camera configuration.
<code>camerayoffset</code>	Specifies the Area-Of-Interest vertical offset. The default value is the value in the current camera configuration.
<code>maxevents</code>	Specifies the maximum number of events to publish.

## 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 110 Required Keys*

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

*Table 111 Optional Keys*

Key-Value Pair	Description
<code>gdconfig=file</code>	Specifies the guaranteed delivery configuration file.
<code>transport=native   solace   tervela   rabbitmq   kafka</code>	Specifies the transport type. If you specify <code>solace</code> , <code>tervela</code> , <code>rabbitmq</code> , or <code>kafka</code> transports instead of the default <code>native</code> transport, then use the required client configuration files specified in the description of the C++ <code>C_dfESPpubsubSetPubsubLib()</code> API call.
<code>loglevel=trace   debug   info   warn   error   fatal   off</code>	Sets the logging level for the adapter. This is the same range of logging levels that you can set in the <code>C_dfESPpubsubInit()</code> <code>publish/subscribe</code> API call and in the engine <code>initialize()</code> call. The default level is <code>warn</code> .
<code>logconfigfile=file</code>	Specifies the log configuration file.
<code>tokenlocation=location</code>	Specifies the location of the file in the local file system that contains the OAuth token that is required for authentication by the <code>publish/subscribe</code> server.

Key-Value Pair	Description
<code>configfilesection=[section]</code>	Specifies the name of the section in <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.config</code> (Windows) to parse for configuration parameters. Specify the value as <code>[configfilesection]</code> .
<code>transactional=true   false</code>	When <code>true</code> , events are transactional.
<code>publishwithupsert=true   false</code>	When <code>true</code> , build events with opcode = Upsert instead of Insert.
<code>transportconfigfile=file</code>	Specifies the publish/subscribe transport configuration file. For solace, the default is <code>./solace.cfg</code> . For tervela, the default is <code>./client.config</code> . For rabbitmq, the default is <code>./rabbitmq.cfg</code> . For kafka, the default is <code>./kafka.cfg</code> . <b>Note:</b> No transport configuration file is required for native transport.
<code>cameraipaddress=ipaddress</code>	Specifies the camera IP address. The default value is the address of the first camera found on the local subnet.
<code>maxnumframes=number</code>	Specifies the maximum number of frames to publish. The default value is no maximum.
<code>maxframerate=number</code>	Specifies the maximum number of frames per second to publish. The default value is the rate at which frames are received from the camera.
<code>camerafeaturesfile=configfile</code>	Specifies a Pylon Features Stream (*.pfs) configuration file to load. The default is to use the current camera configuration unmodified.
<code>camerawidth=width</code>	Specifies the Area-Of-Interest width. The default value is the value in the current camera configuration.
<code>cameraheight=height</code>	Specifies the Area-Of-Interest height. The default value is the value in the current camera configuration.
<code>camerapixelformat=format</code>	Specifies the image pixel format. The default value is the format in the current camera configuration.
<code>camerapacketsize=size</code>	Specifies the Ethernet packet size. The default value is the value in the current camera configuration.
<code>cameraxoffset=horizontaloffset</code>	Specifies the Area-Of-Interest horizontal offset. The default value is the value in the current camera configuration.
<code>camerayoffset=verticaloffset</code>	Specifies the Area-Of-Interest vertical offset. The default value is the value in the current camera configuration.
<code>maxevents=number</code>	Specifies the maximum number of events to publish.

Key-Value Pair	Description
<code>quiesceproject=true</code>   <code>false</code>	When <code>true</code> , quiesces the project after all events are injected into the Source window.
<code>restartonerror=true</code>   <code>false</code>	When <code>true</code> , specifies to restart the adapter when a fatal error is reported.

## Using the Rabbit MQ Connector and Adapter

### 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 event 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 event stream processing publish/subscribe client plug-in library is available. This library enables a standard event stream processing publish/subscribe client application to exchange event blocks with an event 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 ESP 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 <code>buspersistence</code> parameter	Effect
<code>true</code>	<code>auto-delete = false</code> <code>durable = true</code>
<code>false</code>	<code>auto-delete = true</code> <code>durable = false</code>

The following holds when consuming from those queues:

Setting of the <code>buspersistence</code> parameter	Effect
<code>true</code>	<code>exclusive = true</code> <code>noack = false</code>
<code>false</code>	<code>exclusive = false</code> <code>noack = true</code>

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 event 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 SAS Event Stream Processing 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 the `rmqpasswordencrypted` parameter.

**Table 112** Required Parameters for Subscriber Rabbit MQ Connectors

Parameter	Description
<code>type</code>	Specifies to subscribe. Must be "sub".
<code>rmquserid</code>	Specifies the user name required to authenticate the connector's session with the Rabbit MQ server.
<code>rmqpassword</code>	Specifies the password associated with <code>rmquserid</code> .
<code>rmqhost</code>	Specifies the Rabbit MQ server host name.
<code>rmqport</code>	Specifies the Rabbit MQ server port.
<code>rmqexchange</code>	Specifies the Rabbit MQ exchange created by the connector, if nonexistent.
<code>rmqtopic</code>	Specifies the Rabbit MQ routing key to which messages are published.
<code>rmqtype</code>	Specifies <code>binary</code> , <code>csv</code> , <code>json</code> , <code>protobuf</code> , or the name of a string field in the subscribed window schema.
<code>urlhostport</code>	Specifies the <code>host:port</code> 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. 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	<p>Specifies whether to send snapshot data.</p> <p>When <code>true</code>, 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.</p> <p>This parameter is invalid if <code>buspersistence</code> or <code>hotfailover</code> is enabled.</p>

**Table 113** 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 <code>rmquserid</code> .
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 <code>binary</code> , <code>csv</code> , <code>json</code> , <code>protobuf</code> , or <code>opaquestring</code> . For <code>opaquestring</code> , the Source window schema is assumed to be <code>index:int64,message:string</code> .
urlhostport	Specifies the <code>host:port</code> field in the metadata topic subscribed to on start-up to field metadata requests.

**Table 114** 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.

Parameter	Description
<code>dateformat</code>	Specifies the format of <code>DATE</code> and <code>STAMP</code> fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds ( <code>DATE</code> ) or microseconds ( <code>STAMP</code> ) since epoch. The <code>dateformat</code> parameter accepts any time format that is supported by the UNIX <code>strftime</code> function.
<code>buspersistence</code>	Specify to send messages using persistent delivery mode.
<code>protofile</code>	Specifies the <code>.proto</code> file that contains the Google Protocol Buffers message definition used to convert event blocks to <code>protobuf</code> messages. When you specify this parameter, you must also specify the <code>protomsg</code> parameter.  This parameter is ignored when <code>rmqtype</code> is not set to <code>protobuf</code> .
<code>protomsg</code>	Specifies the name of a Google Protocol Buffers message in the <code>.proto</code> file that you specified with the <code>protofile</code> parameter. Event blocks are converted into this message.  This parameter is ignored when <code>rmqtype</code> is not set to <code>protobuf</code> .
<code>csvincludeschema</code>	When <code>rmqtype=CSV</code> , specifies when to prepend output CSV data with the window's serialized schema. Valid values are <code>never</code> , <code>once</code> , and <code>pereventblock</code> . The default value is "never".
<code>useclientmsgid</code>	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.
<code>configfilesection</code>	Specifies the name of the section in <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.config</code> (Windows) to parse for configuration parameters. Specify the value as <code>[configfilesection]</code> .
<code>rmqpasswordencrypted</code>	Specifies that <code>rmqpassword</code> is encrypted.
<code>rmqvhost</code>	Specifies the Rabbit MQ <code>vhost</code> . The default is <code>/</code> .
<code>csvmsgperevent</code>	For CSV, specifies to send one message per event. The default is one message per transactional event block or else one message per event.
<code>csvmsgpereventblock</code>	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
<code>rmqcontenttype</code>	<p>Specifies the value of the <code>content_type</code> parameter in messages sent to RabbitMQ. The default value depends on the message format as follows:</p> <ul style="list-style-type: none"> <li>■ <code>binary</code> <code>application/octet-stream</code></li> <li>■ <code>csv</code> <code>text/csv; charset=utf-8</code></li> <li>■ <code>json</code> <code>application/json</code></li> <li>■ <code>protobufs</code> <code>application/x-protobuf</code></li> <li>■ <code>opaque</code> <code>text/csv; charset=utf-8</code></li> </ul>
<code>rmqheaders</code>	A comma-separated list of key value optional headers in messages sent to RabbitMQ. The default value is no headers.
<code>rmqssl</code>	Enables TLS encryption on the connection to the Rabbit MQ server.
<code>rmqsslcacert</code>	<p>When <code>rmqssl</code> is enabled, specifies the full path of the TLS CA certificate <code>.pem</code> file.</p> <p><b>Important:</b> You cannot use escaped characters in the <i>path</i>.</p>
<code>rmqsslkey</code>	<p>When <code>rmqssl</code> is enabled, specifies the full path of the TLS key <code>.pem</code> file.</p> <p><b>Important:</b> You cannot use escaped characters in the <i>path</i>.</p>
<code>rmqsslcert</code>	<p>When <code>rmqssl</code> is enabled, specifies the full path of the TLS certificate <code>.pem</code> file.</p> <p><b>Important:</b> You cannot use escaped characters in the <i>path</i>.</p>
<code>doubleprecision</code>	Specifies the number of fractional digits in the ASCII representation of a double. The default value is 6.
<code>rmqheartbeat</code>	Specifies the RabbitMQ heartbeat interval in seconds. The default value is 0, meaning that heartbeats are disabled.
<code>usedeliverytagmsgid</code>	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 115** Optional Parameters for Publisher Rabbit MQ Connectors

Parameter	Description
<code>transactional</code>	When <code>rmqtype=CSV</code> , sets the event block type to <code>transactional</code> . The default event block type is <code>normal</code> .
<code>blocksize</code>	When <code>rmqtype=CSV</code> , specifies the number of events to include in a published event block. The default value is 1.

Parameter	Description
<code>dateformat</code>	Specifies the format of <code>DATE</code> and <code>STAMP</code> fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds ( <code>DATE</code> ) or microseconds ( <code>STAMP</code> ) since epoch. The <code>dateformat</code> parameter accepts any time format that is supported by the UNIX <code>strftime</code> function.
<code>buspersistence</code>	Controls both <code>auto-delete</code> and <code>durable</code> .
<code>buspersistencequeue</code>	Specifies the queue name used by a persistent publisher.
<code>ignorecsvparseerrors</code>	Specifies that when a field in an input CSV event cannot be parsed, the event is dropped, an error is logged, and publishing continues.
<code>protofile</code>	Specifies the <code>.proto</code> file that contains the Google Protocol Buffers message definition used to convert event blocks to <code>protobuf</code> messages. When you specify this parameter, you must also specify the <code>protomsg</code> parameter.  This parameter is ignored when <code>rmqtype</code> is not set to <code>protobuf</code> .
<code>protomsg</code>	Specifies the name of a Google Protocol Buffers message in the <code>.proto</code> file that you specified with the <code>protofile</code> parameter. Event blocks are converted into this message.  This parameter is ignored when <code>rmqtype</code> is not set to <code>protobuf</code> .
<code>configfilesection</code>	Specifies the name of the section in <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.config</code> (Windows) to parse for configuration parameters. Specify the value as <code>[configfilesection]</code> .
<code>csvfielddelimiter</code>	Specifies the character delimiter for field data in input CSV events. The default delimiter is the <code>,</code> character.
<code>noautogenfield</code>	Specifies that input events are missing the key field that is autogenerated by the source window.
<code>ackwindow</code>	Specifies the time period (in seconds) to leave messages that are received from Rabbit MQ unacknowledged. Applies only when <code>buspersistence = true</code> , when, by default, messages are never acknowledged. When configured, messages are acknowledged this number of seconds after having been received. Must be configured if <code>acktimer</code> is configured.
<code>acktimer</code>	Specifies the time interval (in seconds) for how often to check whether to send acknowledgments that are triggered by the <code>ackwindow</code> parameter. Must be configured if <code>ackwindow</code> is configured.
<code>publishwithupsert</code>	Builds events with <code>opcode=Upsert</code> instead of <code>Insert</code> .
<code>rmqpasswordencrypted</code>	Specifies that <code>rmqpassword</code> is encrypted.
<code>addcsvopcode</code>	Prepends an opcode and comma to input CSV events. The opcode is <code>Insert</code> unless <code>publishwithupsert</code> is enabled.
<code>addcsvflags</code>	Specifies the event type to insert into input CSV events (with a comma). Valid values are <code>normal</code> and <code>partialupdate</code> .

Parameter	Description
<code>rmqvhost</code>	Specifies the Rabbit MQ vhost. The default is <code>/</code> .
<code>useclientmsgid</code>	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.
<code>rmqssl</code>	Enables TLS encryption on the connection to the Rabbit MQ server.
<code>rmqsslcacert</code>	When <code>rmqssl</code> is enabled, specifies the full path of the TLS CA certificate <code>.pem</code> file.
<code>rmqsslkey</code>	When <code>rmqssl</code> is enabled, specifies the full path of the TLS key <code>.pem</code> file.
<code>rmqsslcert</code>	When <code>rmqssl</code> is enabled, specifies the full path of the TLS certificate <code>.pem</code> file.
<code>maxevents</code>	Specifies the maximum number of events to publish.
<code>stripnullsfromcsv</code>	Strip all nulls from input CSV events.
<code>rmqheartbeat</code>	Specifies the RabbitMQ heartbeat interval in seconds. The default value is 0, meaning that heartbeats are disabled.
<code>usedeliverytagmsgid</code>	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.

## 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 116 Required Keys*

Key-Value Pair	Description
<code>type=sub</code>	Specifies a subscriber adapter.

Key-Value Pair	Description
url=pubsubURL	Specifies the standard URL in the form <b>dfESP://host:port/project/continuousquery/window</b> . Append <code>?snapshot=true</code>   <code>false</code> for subscribers. Append <code>?collapse=true</code>   <code>false</code> or <code>?rmretDel=true</code>   <code>false</code> 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.
rmqtopic=key	Specifies the Rabbit MQ routing key.
urlhostport=field	Specifies the <i>host:port</i> 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 117 Optional Keys

Key-Value Pairs	Description
rmqssl=true   false	When <code>true</code> , enables TLS encryption on the connection to the Rabbit MQ server.
rmqsslcacert=path	When <code>rmqssl</code> is <code>true</code> , specifies the full path of the TLS CA certificate <code>.pem</code> file. <b>Important:</b> You cannot use escaped characters in <i>path</i> .
rmqsslkey=path	When <code>rmqssl</code> is <code>true</code> , specifies the full path of the TLS key <code>.pem</code> file. <b>Important:</b> You cannot use escaped characters in <i>path</i> .
rmqsslcert=path	When <code>rmqssl</code> is <code>true</code> , specifies the full path of the TLS certificate <code>.pem</code> file. <b>Important:</b> You cannot use escaped characters in <i>path</i> .
buspersistence=true   false	When <code>true</code> , use durable queues and persistent messages.

Key-Value Pairs	Description
<code>dateformat=format</code>	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 <code>dateformat</code> parameter accepts any time format that is supported by the UNIX <code>strftime</code> function.
<code>protofile=file</code>	Specifies the <code>.proto</code> file to be used for Google protocol buffer support. This key is ignored when <code>rmqtype</code> is not set to <code>protobuf</code> .
<code>protomsg=message</code>	Specifies the message itself in the <code>.proto</code> file that is specified by the <code>protofile</code> parameter. This key is ignored when <code>rmqtype</code> is not set to <code>protobuf</code> .
<code>gdconfig=file</code>	Specifies the guaranteed delivery configuration file.
<code>transport=native   solace   tervela   rabbitmq   kafka</code>	Specifies the transport type. If you specify <code>solace</code> , <code>tervela</code> , <code>rabbitmq</code> , or <code>kafka</code> transports instead of the default <code>native</code> transport, then use the required client configuration files specified in the description of the C++ <code>C_dfESPPubsubSetPubsubLib()</code> API call.
<code>loglevel=trace   debug   info   warn   error   fatal   off</code>	Sets the logging level for the adapter. This is the same range of logging levels that you can set in the <code>C_dfESPPubsubInit()</code> <code>publish/subscribe</code> API call and in the engine <code>initialize()</code> call. The default level is <code>warn</code> .
<code>logconfigfile=file</code>	Specifies the log configuration file.
<code>tokenlocation=location</code>	Specifies the location of the file in the local file system that contains the OAuth token that is required for authentication by the <code>publish/subscribe</code> server.
<code>configfilesection=[section]</code>	Specifies the name of the section in <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.config</code> (Windows) to parse for configuration parameters. Specify the value as <code>[configfilesection]</code> .
<code>rmqpasswordencrypted=true   false</code>	When <code>true</code> , <code>rmqpassword</code> is encrypted.
<code>rmqvhost=vhost</code>	Specifies the Rabbit MQ virtual host. The default is <code>/</code> .
<code>restartonerror=true   false</code>	When <code>true</code> , specifies to restart the adapter if a fatal error is reported.

Key-Value Pairs	Description
<code>transportconfigfile=file</code>	Specifies the publish/subscribe transport configuration file. For <code>solace</code> , the default is <code>./solace.cfg</code> . For <code>tervela</code> , the default is <code>./client.config</code> . For <code>rabbitmq</code> , the default is <code>./rabbitmq.cfg</code> . For <code>kafka</code> , the default is <code>./kafka.cfg</code> . <b>Note:</b> No transport configuration file is required for native transport.
<code>csvincludeschema=never   once   pereventblock</code>	When <code>rmqtype=CSV</code> , specifies when to pass the window's serialized schema to subscriber callback. The default value is <code>never</code> .
<code>csvmsgperevent=true   false</code>	When <code>true</code> , for CSV, specifies to send one message per event. The default is one message per transactional event block or else one message per event.
<code>csvmsgpereventblock=true   false</code>	When <code>true</code> , 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.
<code>rmqcontenttype=value</code>	Specifies the value of the <code>content_type</code> parameter in messages sent to RabbitMQ. The default value depends on the message format, as follows: <ul style="list-style-type: none"> <li>■ <code>binary</code> "application/octet-stream"</li> <li>■ <code>csv</code> "text/csv; charset=utf-8"</li> <li>■ <code>json</code> "application/json"</li> <li>■ <code>protobufs</code> "application/x-protobuf"</li> <li>■ <code>opaque</code> "text/csv; charset=utf-8"</li> </ul>
<code>rmqheaders=list</code>	Specifies a comma-separated list of <code>key:value</code> optional headers in messages sent to RabbitMQ. The default value is no headers.
<code>rmqheartbeat=value</code>	Specifies the RabbitMQ heartbeat interval in seconds. The default value is 0, which means that heartbeats are disabled.

## Publisher Usage

Table 118 Required Keys

Key-Value Pair	Description
<code>type=pub</code>	Specifies a publisher adapter.

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

**Table 119** Optional Keys

Key-Value Pair	Description
<code>rmqssl=true   false</code>	When <code>true</code> , enables TLS encryption on the connection to the Rabbit MQ server.
<code>rmqsslcacert=path</code>	When <code>rmqssl</code> is <code>true</code> , specifies the full path of the TLS CA certificate <code>.pem</code> file.
<code>buspersistence=true   false</code>	When <code>true</code> , use durable queues and persistent messages.
<code>dateformat=format</code>	Specifies the format of <code>DATE</code> and <code>STAMP</code> fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds ( <code>DATE</code> ) or microseconds ( <code>STAMP</code> ) since epoch. The <code>dateformat</code> parameter accepts any time format that is supported by the UNIX <code>strftime</code> function.
<code>protofile=file</code>	Specifies the <code>.proto</code> file to be used for Google protocol buffer support. This key is ignored when <code>rmqtype</code> is not set to <code>protobuf</code> .
<code>protomsg=message</code>	Specifies the message itself in the <code>.proto</code> file that is specified by the <code>protofile</code> parameter. This key is ignored when <code>rmqtype</code> is not set to <code>protobuf</code> .
<code>gdconfig=file</code>	Specifies the guaranteed delivery configuration file.

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, then use the required client configuration files specified in the description of the C++ <code>C_dfESPPubsubSetPubsubLib()</code> 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 <code>C_dfESPPubsubInit()</code> publish/subscribe API call and in the engine <code>initialize()</code> 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 <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.config</code> (Windows) to parse for configuration parameters. Specify the value as <code>[configfilesection]</code> .
rmqpasswordencrypted=true   false	When true, <code>rmqpassword</code> is encrypted.
rmqyhost=vhost	Specifies the Rabbit MQ virtual host. The default is <code>/</code> .
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 <code>./solace.cfg</code> . For tervela, the default is <code>./client.config</code> . For rabbitmq, the default is <code>./rabbitmq.cfg</code> . For kafka, the default is <code>./kafka.cfg</code> . <b>Note:</b> 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 <code>buspersistence</code> is true.

Key-Value Pair	Description
<code>acktimer=seconds</code>	When <code>buspersistence</code> is <code>true</code> , specifies the time interval for how often to check whether to send acknowledgments that are triggered by the <code>ackwindow</code> parameter.
<code>csvfielddelimiter=delimiter</code>	Specifies the character delimiter for field data in input CSV events. The default delimiter is the <code>,</code> character.
<code>noautogenfield=true   false</code>	When <code>true</code> , specifies that input events are missing the key field that is autogenerated by the Source window.
<code>publishwithupsert=true   false</code>	When <code>true</code> , build events with <code>opcode = Upsert</code> instead of <code>Insert</code> .
<code>addcsvopcode= true   false</code>	When <code>true</code> , prepends an <code>opcode</code> and comma to write CSV events. The <code>opcode</code> is <code>Insert</code> unless <code>publishwithupsert</code> is <code>true</code> .
<code>addcsvflags=none   normal   partialupdate</code>	Specifies the event type to Insert into input CSV events (with comma).
<code>maxevents=number</code>	Specifies the maximum number of events to publish.
<code>quiesceproject=true   false</code>	When <code>true</code> , quiesces the project after all events are injected into the Source window.
<code>stripnullsfromcsv=true   false</code>	When <code>true</code> , strip all nulls from input CSV events.
<code>rmqheartbeat=value</code>	Specifies the RabbitMQ heartbeat interval in seconds. The default value is <code>0</code> , which means that heartbeats are disabled.

## Publisher Failover

For information about implementing hot failover for publisher adapters, see [“Publisher Adapter Failover with Kafka”](#) in *SAS Event Stream Processing: Implementing Failover*.

## 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 ESP URL that is configured in the `esprespurl` 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 120 Required Keys*

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

*Table 121 Optional Keys*

Key-Value Pair	Description
<code>configfilesection=name</code>	Specifies the name of the section in <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/javaadapters.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\javaadapters.config</code> (Windows) to parse for configuration parameters.
<code>dateformat=format</code>	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 <code>dateformat</code> parameter accepts any time format that is supported by the UNIX <code>strftime</code> function.
<code>espresprurl=url</code>	Specifies the publish/subscribe standard URL to which responses should be published.
<code>gdconfigfile=filename</code>	Specifies the guaranteed delivery configuration file for the client.

Key-Value Pair	Description
<code>ignorefailedhttpconnects=true   false</code>	When true, drop the subscribed event and continue when an HTTP connect fails. The default is true.
<code>opaquejson=field</code>	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 <code>ESP_UTF8STR</code> .
<code>pubsublib= native   solace   tervela   rabbitmq   kafka</code>	Specifies the transport type. When you specify the <code>solace</code> , <code>tervela</code> , <code>rabbitmq</code> , or <code>kafka</code> transports instead of the default <code>native</code> transport, use the required client configuration files specified in the description of the C++ <code>C_dfESPpubsubSetPubsubLib()</code> API call.
<code>maxnumthreads=num</code>	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.
<code>loglevel=severe   warning   info</code>	Specifies the application logging level.
<code>tokenlocation=location</code>	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.
<code>httpretries=number</code>	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.
<code>httprequestproperties=list</code>	Specifies any number of additional headers to include in the HTTP POST request to the REST service. Specify as semicolon– separated <code>&lt;key&gt;:&lt;value&gt;</code> pairs. For keys that support multiple values, separate those values with a comma.
<code>httpstatuscodes=list</code>	Specifies a comma-separated list of acceptable status codes in response to the HTTP post. The default required response is 201.
<code>httpretryinterval=seconds</code>	Specifies the number of seconds between HTTP post retries. The default is 0.
<code>restartonerror=true   false</code>	When true, restarts the adapter when a fatal error is reported.
<code>transportconfigfile=file</code>	Specifies the full path to the publish/subscribe transport configuration file. The default file depends on the transport type specified with through the <code>pubsublib</code> parameter: For <code>solace</code> , the default file is <code>solace.cfg</code> . For <code>tervela</code> , the default file is <code>client.config</code> . For <code>rabbitmq</code> , the default file is <code>rabbitmq.cfg</code> . For <code>kafka</code> , the default file is <code>kafka.cfg</code> . <b>Note:</b> No transport configuration file is required for native transport.

## Using the SAS Data Set Adapter

The SAS data set adapter resides in `dfx-esp-dataset-adapter.jar`, which bundles the Java publisher and subscriber SAS Event Stream Processing clients.

The adapter uses SAS Java Database Connectivity (JDBC) to connect to a SAS Workspace Server. This SAS Workspace Server manages reading and writing of the data set. The SAS data set name and SAS Workspace Server user credentials are passed as required parameters to the adapter.

The SAS JDBC requires `log4j-version.jar` for logging. Configure the value of the `DFESP_LOG4J_JAR` environment variable to be the path to `log4j-version.jar`.

**Note:** JAR files are located in `$DFESP_HOME/lib`.

Configure the value of the `DFESP_LOG4J_XML` environment variable to the location of the `log4j.xml` file.

The user password must be passed in encrypted form unless `-w true` is configured. The encrypted version of the password can be generated by using OpenSSL, which must be installed on your system. When you install the SAS Event Stream Processing 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 "password" | openssl enc -e -aes-256-cbc -a -salt -pass pass:"espDSadapterUsedByUser=userid"
```

The subscriber client receives event blocks and appends corresponding rows to the SAS data set it creates. It also supports Update and Delete operations on the data set.

**Note:** Invoking the SAS data set adapter on an existing data set overwrites the data set.

The Workspace Server creates the data set on its local file system, so the data set name must comply with these SAS naming rules:

- The length of the names can be up to 32 characters.
- Names must begin with a letter of the Latin alphabet (A–Z, a–z) or the underscore. Subsequent characters can be letters of the Latin alphabet, numerals, or underscores.
- Names cannot contain blanks or special characters except for the underscore.
- Names can contain mixed-case letters. SAS internally converts the member name to uppercase.

You can explicitly specify the data set schema using the `-s` option. You can use the `-a` switch to do the following:

- specify a number of received events to analyze
- extract an appropriate row size for the data set before creating data set rows

The `-s` or `-a` switch must be present.

You can also configure the subscriber client to periodically write a SAS data set using the optional `periodicity` or `maxnumrows` parameters. If so configured, a timestamp is appended to the filename of each written file. Be aware that Update and Delete operations are not supported if `periodicity` or `maxnumrows` is configured, because the referenced row might not be present in the currently open data set.

If you have configured a subscriber client with multiple ESP URLs, events from multiple windows in multiple models are aggregated into a single output data set. In this case, each subscribed window must have the same schema. Also, the order in which rows are appended to the data set is not guaranteed.

The publisher client does the following:

- reads rows from the SAS data set

- converts them into events with `opcode = insert` (unless you specify `-r`)
- injects event blocks into a Source window of an engine

If you configure a publisher client with multiple URLs, each generated event block is injected into multiple Source windows. Each Source window must have the same schema.

Subscriber usage:

**dfesp\_dataset\_adapter** -C *key1=val1,key2=val2,key3=val3,...*

**Table 122** Required Keys

Key-Value Pair	Description
<code>wssurl=url</code>	Specifies the SAS Workspace Server connection URL in the form "host:port".
<code>dsname=name</code>	Specifies the subscriber output file or publisher input file. The full path, including the extension, must be specified for <code>dsname</code> . For example, "D:/sas/SASBI/Lev1/AppData/filename.sas7bdat"
<code>url=url</code>	Specifies the event stream processing standard URL in the following format: "dfESP://host:port/project/contquery/window?snapshot=true   false?collapse=true   false?rmretel=true   false". <b>Note:</b> Multiple URLs, separated by commas, are permitted.
<code>type=sub</code>	Specifies a subscriber adapter.
<code>username=name</code>	Specifies the SAS Workspace Server user name.
<code>password=password</code>	Specifies the user password to access the SAS Workspace Server.

**Table 123** Optional Keys

Key-Value Pair	Description
<code>obstoanalyze=number</code>	Specifies the number of observations to analyze to define the output SAS data set structure. Default value is 4096.
<code>configfilesection=section</code>	Specifies the name of the section in <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/javaadapters.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\javaadapters.config</code> (Windows) to parse for configuration parameters.
<code>gdconfigfile=file</code>	Specifies the guaranteed delivery configuration file.
<code>pubsublib= native   solace   tervela   rabbitmq   kafka</code>	Specifies the transport type. The default value is <code>native</code> .
<code>loglevel=severe   warning   info</code>	Specifies the application logging level. The default value is <code>warning</code> .

Key-Value Pair	Description
maxnumrows=number	Specifies the output file periodicity in number of rows. Invalid if data is not insert-only.
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.
dsschema=schema	<p>Specifies the output data set schema in the following form:</p> <pre>"rowname1:sastype1:sasformat1:label1,...,rownameN:sastypeN:sasformatN:labelN"</pre> <p>If you omit <i>sasformatX</i>, then no SAS format is applied to the row. If you omit <i>labelX</i>, then no label is applied to the row.</p> <p>You must specify the type size for type 'char'. The size for type 'num' has a default size of 8 and does not need to be specified.</p> <p>Labels do not need to be wrapped in quotation marks.</p> <p>See the example here:</p> <pre>-s "ID:num,OUTPUT_DTTM:num:DATETIME20.:Output Datetime,VIN:char(17)"</pre> <p><b>Note:</b> If you had previously specified the -a option, this option is ignored.</p>
restartonerror=true   false	When true, restarts the adapter if a fatal error is reported.
unencryptedpassword=true   false	When true, the configured password is not encrypted. By default, it is.
bufsize=size	Specifies the socket buffer size. By default, it is 16MB.
transportconfigfile=file	<p>Specifies the full path to the publish/subscribe transport configuration file. The default file depends on the transport type specified with through the <code>pubsublib</code> parameter:</p> <p>For <code>solace</code>, the default file is <code>solace.cfg</code>.</p> <p>For <code>tervela</code>, the default file is <code>client.config</code>.</p> <p>For <code>rabbitmq</code>, the default file is <code>rabbitmq.cfg</code>.</p> <p>For <code>kafka</code>, the default file is <code>kafka.cfg</code>.</p> <p><b>Note:</b> No transport configuration file is required for native transport.</p>

Publisher usage:

**dfesp\_dataset\_adapter** -C *key1=val1, key2=val2, key3=val3, ...*

**Table 124** Required Keys

Key-Value Pair	Description
wssurl=url	Specifies the SAS Workspace Server connection URL in the form "host:port".

Key-Value Pair	Description
dsname=name	Specifies the subscriber output file or publisher input file. The full path, including the extension, must be specified for dsname. For example, "D:/sas/SASBI/Lev1/AppData/filename.sas7bdat"
url=url	Specifies the event stream processing standard URL in the following format: "dfESP://host:port/project/contquery/window". <b>Note:</b> Multiple URLs, separated by commas, are permitted.
type=pub	Specifies a publisher adapter.
username=name	Specifies the SAS Workspace Server user name.
password=password	Specifies the user password to access the SAS Workspace Server.

Table 125 Optional Keys

Key-Value Pair	Description
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/sas/viya/config/etc/SASEventStreamProcessingEngine/default/javaadapters.config (Linux) or %ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\javaadapters.config (Windows) to parse for configuration parameters.
transactional=true   false	When true, event blocks are transactional. By default, they are normal.
gdconfigfile=file	Specifies the guaranteed delivery configuration file.
pubsublib= native   solace   tervera   rabbitmq   kafka	Specifies the transport type. The default 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.
sqlquery=query	Specifies an SQL query to get SAS data set content. For example: "select * from dataset". <b>Note:</b> For <i>dataset</i> , specify the <i>name</i> paired with the dsname key, but leave off the extension.
quiesceproject=true   false	When true, quiesce the project after all events are injected into the Source window.

Key-Value Pair	Description
<code>publishwithupsert=true   false</code>	When true, build events with <code>opcode=Upsert</code> . By default, events are built with <code>opcode=Insert</code> . <b>javaadapters.config</b> parameter name: <code>publishwithupsert</code>
<code>restartonerror=true   false</code>	When true, restarts the adapter when a fatal error is reported. <b>javaadapters.config</b> parameter name: <code>restartonerror</code>
<code>unencryptedpassword=true   false</code>	When true, the configured password is not encrypted. By default, it is.
<code>maxevents=number</code>	Specifies the maximum number of events to publish.
<code>bufsize=size</code>	Specifies the socket buffer size. By default, it is 16MB.
<code>transportconfigfile=file</code>	Specifies the full path to the publish/subscribe transport configuration file. The default file depends on the transport type specified with through the <code>pubsublib</code> parameter: For <code>solace</code> , the default file is <code>solace.cfg</code> . For <code>tervela</code> , the default file is <code>client.config</code> . For <code>rabbitmq</code> , the default file is <code>rabbitmq.cfg</code> . For <code>kafka</code> , the default file is <code>kafka.cfg</code> . <b>Note:</b> No transport configuration file is required for native transport.

For information about implementing hot failover for publisher adapters, see [“Publisher Adapter Failover with Kafka” in SAS Event Stream Processing: Implementing Failover](#).

## Using the SMTP Connector and Adapter

### 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 126** Required Parameters for the SMTP Connector

Parameter	Description
<code>type</code>	Specifies to subscribe. Must be "sub".
<code>smtpserver</code>	Specifies the SMTP server host name or IP address.
<code>sourceaddress</code>	Specifies the email address to be used in the "from" field of the email.
<code>destaddress</code>	Specifies the email address to which to send the email message.
<code>snapshot</code>	Specifies whether to send snapshot data.  When <code>true</code> , 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 127** Optional Parameters for SMTP Connectors

Parameter	Description
<code>collapse</code>	Enables conversion of UPDATE_BLOCK events to make subscriber output publishable.
<code>emailperevent</code>	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.
<code>rmretdel</code>	Specifies to remove all delete events from event blocks received by a subscriber that were introduced by a window retention policy.
<code>configfilesection</code>	Specifies the name of the section in <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.config</code> (Windows) to parse for configuration parameters. Specify the value as <code>[configfilesection]</code> .
<code>dateformat</code>	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 <code>dateformat</code> parameter accepts any time format that is supported by the UNIX <code>strftime</code> function.
<code>doubleprecision</code>	Specifies the number of fractional digits in the ASCII representation of a double. The default value is 6.
<code>connectionpereventblock</code>	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.

## 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 128 Required Keys*

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

*Table 129 Optional Keys*

Key-Value Pair	Description
<code>gdconfig=file</code>	Specifies the guaranteed delivery configuration file.
<code>transport=native   solace   tervela   rabbitmq   kafka</code>	Specifies the transport type. If you specify <code>solace</code> , <code>tervela</code> , <code>rabbitmq</code> , or <code>kafka</code> transports instead of the default <code>native</code> transport, then use the required client configuration files specified in the description of the C++ <code>C_dfESPpubsubSetPubsubLib()</code> API call.
<code>loglevel=trace   debug   info   warn   error   fatal   off</code>	Sets the logging level for the adapter. This is the same range of logging levels that you can set in the <code>C_dfESPpubsubInit()</code> <code>publish/subscribe</code> API call and in the engine <code>initialize()</code> call. The default level is <code>warn</code> .
<code>logconfigfile=file</code>	Specifies the log configuration file.
<code>tokenlocation=location</code>	Specifies the location of the file in the local file system that contains the OAuth token that is required for authentication by the <code>publish/subscribe</code> server.

Key-Value Pair	Description
<code>configfilesection=[section]</code>	Specifies the name of the section in <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.config</code> (Windows) to parse for configuration parameters. Specify the value as <code>[configfilesection]</code> .
<code>emailperevent=true   false</code>	When <code>true</code> , send an email message for every event instead of one per event block.
<code>restartonerror=true   false</code>	When <code>true</code> , specifies to restart the adapter if a fatal error is reported.
<code>transportconfigfile=file</code>	Specifies the publish/subscribe transport configuration file. For <code>solace</code> , the default is <code>./solace.cfg</code> . For <code>tervela</code> , the default is <code>./client.config</code> . For <code>rabbitmq</code> , the default is <code>./rabbitmq.cfg</code> . For <code>kafka</code> , the default is <code>./kafka.cfg</code> . <b>Note:</b> No transport configuration file is required for native transport.
<code>dateformat=format</code>	Specifies the format of <code>DATE</code> and <code>STAMP</code> fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds ( <code>DATE</code> ) or microseconds ( <code>STAMP</code> ) since epoch. The <code>dateformat</code> parameter accepts any time format that is supported by the UNIX <code>strftime</code> function.
<code>doubleprecision=number</code>	Specifies the number of fractional digits in the ASCII representation of a double. The default value is 6.
<code>connectionpereventblock=true   false</code>	When <code>true</code> , reconnect and disconnect to and from the SMTP server for every event block. When <code>false</code> , maintain a persistent connection to the SMTP server. The default value is <code>false</code> .

## Using the Sniffer Connector and Adapter

### Overview

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 instantiated 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 the `httpports` 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 SAS Event Stream Processing 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 SAS Event Stream Processing server.

The `packetfields` configuration parameter uses SAS Event Stream Processing 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 130** Required Parameters for the Sniffer Connector

Parameter	Description
<code>type</code>	Specifies to publish. Must be "pub".
<code>interface</code>	Specifies the name of the network interface on the local machine from which to capture packets.
<code>protocol</code>	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.
<code>packetfields</code>	<p>Specifies the packet fields to be extracted from a captured packet and included in the published event. Use SAS Event Stream Processing schema syntax. This value does not include the <code>index:int64</code> <code>frame_time:stamp</code>, or <code>ts:stamp</code> fields.</p> <p>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.</p> <p>An example for Radius is as follows:  <code>radius_AcctStatusType:int32,radius_EventTimestamp:stamp,radius_FramedIPAddress:string,radius_UserName:string</code>.</p> <p>An example for HTTP is as follows:  <code>ip_Source:string,ip_Destination:string,http_Host:string,http_Referer:string,http_UserAgent:string,http_GET_RequestURI:string</code>.</p> <p>An example for TLS is as follows:  <code>sslts_Handshake_ClientHello_GMTUnixTime:date,sslts_Handshake_ClientHello_SessionID:string,sslts_Handshake_ClientHello_CypherSuites:string,sslts_Handshake_ClientHello_CompressionMethods:string,sslts_Handshake_ClientHello_Extensions_ServerName:string</code>.</p> <p>If <code>protocol</code> is not set to 80, 1813, or 443 and <code>httpports</code> is not configured, the only supported values are as follows:</p> <ul style="list-style-type: none"> <li>■ the IP source and destination address</li> <li>■ the TCP or UDP source and destination ports</li> <li>■ <code>payload:string</code></li> </ul>

Table 131 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.
addtimestamp	Specifies to append an <code>ESP_TIMESTAMP</code> 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 <code>packetfields</code> parameter.
configfilesection	Specifies the name of the section in <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.config</code> (Windows) to parse for configuration parameters. Specify the value as <code>[configfilesection]</code> .
vendorid	Specifies the <code>vendor-Id</code> field to match when capturing the <code>Attribute-Specific</code> field in a <code>Vendor-Specific</code> attribute in a <code>Radius Accounting-Request</code> packet.
vendortype	Specifies the <code>vendor-Type</code> field to match when capturing the <code>Attribute-Specific</code> field in a <code>Vendor-Specific</code> attribute in a <code>Radius Accounting-Request</code> packet.
indexfieldname	Specifies the name to use instead of <code>index</code> for the <code>index:int64</code> field in the Source window schema.
publishwithupsert	Specifies to build events with <code>opcode=Upsert</code> instead of <code>opcode=Insert</code> .
pcapfilter	Specifies a filter expression as defined in the <code>pcap</code> documentation. Passed to the <code>pcap</code> 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.

## 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` run-time 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 132** Required Keys

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

**Table 133** Optional Keys

Key-Value Pair	Description
<code>gdconfig=file</code>	Specifies the guaranteed delivery configuration file.
<code>transport=native   solace   tervela   rabbitmq   kafka</code>	Specifies the transport type. If you specify <code>solace</code> , <code>tervela</code> , <code>rabbitmq</code> , or <code>kafka</code> transports instead of the default <code>native</code> transport, then use the required client configuration files specified in the description of the C++ <code>C_dfESPpubsubSetPubsubLib()</code> API call.
<code>loglevel=trace   debug   info   warn   error   fatal   off</code>	Sets the logging level for the adapter. This is the same range of logging levels that you can set in the <code>C_dfESPpubsubInit()</code> <code>publish/subscribe</code> API call and in the engine <code>initialize()</code> call. The default level is <code>warn</code> .
<code>logconfigfile=file</code>	Specifies the log configuration file.
<code>tokenlocation=location</code>	Specifies the location of the file in the local file system that contains the OAuth token that is required for authentication by the <code>publish/subscribe</code> server.
<code>configfilesection=[section]</code>	Specifies the name of the section in <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.config</code> (Windows) to parse for configuration parameters. Specify the value as <code>[configfilesection]</code> .
<code>blocksize=size</code>	Sets the block size. The default value is 1.
<code>transactional=true   false</code>	When <code>true</code> , events are transactional.

Key-Value Pair	Description
<code>addtimestamp= true   false</code>	When <code>true</code> , appends an <code>ESP_TIMESTAMP</code> field to each published event.
<code>vendorid=field</code>	Specifies the vendor ID field to match when capturing the Attribute-Specific field in a Vendor-Specific attribute in a Radius Accounting-Request packet.
<code>vendortype=field</code>	Specifies the vendor type field to match when capturing the Attribute-Specific field in a Vendor-Specific attribute in a Radius Accounting-Request packet.
<code>indexfieldname=name</code>	Specifies the name to use instead of <code>index</code> for the <code>index:int65</code> field in the Source window schema.
<code>publishwithupsert=true   false</code>	When <code>true</code> , build events with <code>opcode = Upsert</code> instead of <code>Insert</code> .
<code>pcapfilter=expression</code>	Specifies a filter expression as defined in the <code>pcap</code> documentation. The value is passed to the <code>pcap</code> driver in order to filter packets received by the adapter.
<code>httpports=list</code>	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 <code>80</code> .
<code>ignorenonpayloadpackets=true   false</code>	When <code>true</code> , specifies to ignore packets that have no payload, as calculated by subtracting the TCP or UDP header size from the packet size.
<code>restartonerror=true   false</code>	When <code>true</code> , specifies to restart the adapter if a fatal error is reported.
<code>transportconfigfile=file</code>	Specifies the publish/subscribe transport configuration file. For <code>solace</code> , the default is <code>./solace.cfg</code> . For <code>tervela</code> , the default is <code>./client.config</code> . For <code>rabbitmq</code> , the default is <code>./rabbitmq.cfg</code> . For <code>kafka</code> , the default is <code>./kafka.cfg</code> . <b>Note:</b> No transport configuration file is required for native transport.
<code>maxevents=number</code>	Specifies the maximum number of events to publish.
<code>quiesceproject=true   false</code>	When <code>true</code> , 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” in SAS Event Stream Processing: Implementing Failover](#).

## Using the Solace Connector and Adapter

### 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 event 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 `buspersistencequeue` 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 ESP 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 SAS Event Stream Processing 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 134** 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.
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	<p>Specifies whether to send snapshot data.</p> <p>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.</p>

**Table 135** 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 136** *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.
configfilesection	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].
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 137** *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.

Parameter	Description
<code>protomsg</code>	Specifies the name of a Google Protocol Buffers message in the <code>.proto</code> file that you specified with the <code>protofile</code> parameter. Event blocks are converted into this message.
<code>configfilesection</code>	Specifies the name of the section in <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.config</code> (Windows) to parse for configuration parameters. Specify the value as <code>[configfilesection]</code> .
<code>json</code>	Enables transport of event blocks encoded as JSON messages.
<code>publishwithupsert</code>	Specifies to build with <code>opcode = Upsert</code> instead of <code>opcode = Insert</code> .
<code>dateformat</code>	Specifies the format of <code>DATE</code> and <code>STAMP</code> fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds ( <code>DATE</code> ) or microseconds ( <code>STAMP</code> ) since epoch. The <code>dateformat</code> parameter accepts any time format that is supported by the UNIX <code>strftime</code> function.
<code>solpasswordencrypted</code>	Specifies that <code>solpassword</code> is encrypted.
<code>getmsgfromdestattr</code>	Extracts the payload from the destination attribute instead of the message body.
<code>transactional</code>	When <code>getmsgfromdestattr</code> is enabled, sets the event block type to <code>transactional</code> . The default event block type is <code>normal</code> .
<code>blocksize</code>	When <code>getmsgfromdestattr</code> is enabled, specifies the number of events to include in a published event block. The default value is 1.
<code>maxevents</code>	Specifies the maximum number of events to publish.

## 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 138 Required Keys

Key-Value Pair	Description
<code>type=sub</code>	Specifies a subscriber adapter.
<code>url=pubsubURL</code>	Specifies the standard URL in the form <code>dfESP://host:port/project/continuousquery/window</code> . Append <code>?snapshot=true</code>   <code>false</code> for subscribers. Append <code>?collapse=true</code>   <code>false</code> or <code>?rmretdel=true</code>   <code>false</code> or both for subscribers if needed.
<code>soluserid=username</code>	Specifies the Solace user name.
<code>solpassword=password</code>	Specifies the Solace password.
<code>solhostport=hostport</code>	Specifies the Solace <i>host:port</i> .
<code>solvpn=vpn</code>	Specifies the Solace VPN name.
<code>soltopic=topic</code>	Specifies the Solace topic.
<code>urlhostport=hostport</code>	Specifies the <i>host:port</i> field in the metadata topic subscribed to by the connector.
<code>numbufferedmsgs=number</code>	Specifies the maximum number of messages buffered by a standby subscriber connector.

Table 139 Optional Keys

Key-Value Pair	Description
<code>buspersistence=true</code>   <code>false</code>	When <code>true</code> , use durable queues and persistent messages.
<code>protofile=file</code>	Specifies the <code>.proto</code> file to be used for Google protocol buffer support.
<code>protomsg=message</code>	Specifies the message itself in the <code>.proto</code> file that is specified by the <code>protofile</code> parameter.
<code>json=true</code>   <code>false</code>	When <code>true</code> , transport JSON messages instead of event blocks.
<code>gdconfig=file</code>	Specifies the guaranteed delivery configuration file.
<code>transport=native</code>   <code>solace</code>   <code>tervela</code>   <code>rabbitmq</code>   <code>kafka</code>	Specifies the transport type. If you specify <code>solace</code> , <code>tervela</code> , <code>rabbitmq</code> , or <code>kafka</code> transports instead of the default <code>native</code> transport, then use the required client configuration files specified in the description of the C++ <code>C_dfESPpubsubSetPubsubLib()</code> 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 <code>C_dfESPpubsubInit()</code> publish/subscribe API call and in the engine <code>initialize()</code> call. The default level is <code>warn</code> .
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 <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.config</code> (Windows) to parse for configuration parameters. Specify the value as <code>[configfilesection]</code> .
dateformat=format	Specifies the format of <code>DATE</code> and <code>STAMP</code> fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds ( <code>DATE</code> ) or microseconds ( <code>STAMP</code> ) since epoch. The <code>dateformat</code> parameter accepts any time format that is supported by the UNIX <code>strftime</code> 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 <code>./solace.cfg</code> . For tervela, the default is <code>./client.config</code> . For rabbitmq, the default is <code>./rabbitmq.cfg</code> . For kafka, the default is <code>./kafka.cfg</code> . <b>Note:</b> No transport configuration file is required for native transport.

## Publisher Usage

Table 140 Required Keys

Key-Value Pair	Description
type=pub	Specifies a publisher adapter.
url=pubsubURL	Specifies the standard URL in the form <code>dfESP://host:port/project/continuousquery/window</code> .
soluserid=username	Specifies the Solace user name.

Key-Value Pair	Description
<code>solpassword=password</code>	Specifies the Solace password.
<code>solhostport=hostport</code>	Specifies the Solace <i>host:port</i> .
<code>solvpn=vpn</code>	Specifies the Solace VPN name.
<code>soltopic=topic</code>	Specifies the Solace topic.
<code>urlhostport=hostport</code>	Specifies the <i>host:port</i> field in the metadata topic subscribed to by the connector.

**Table 141** *Optional Keys*

Key-Value Pair	Description
<code>buspersistence=true   false</code>	When <code>true</code> , use durable queues and persistent messages.
<code>protofile=file</code>	Specifies the <code>.proto</code> file to be used for Google protocol buffer support.
<code>protomsg=message</code>	Specifies the message itself in the <code>.proto</code> file that is specified by the <code>protofile</code> parameter.
<code>json=true   false</code>	When <code>true</code> , transport JSON messages instead of event blocks.
<code>gdconfig=file</code>	Specifies the guaranteed delivery configuration file.
<code>transport=native   solace   tervela   rabbitmq   kafka</code>	Specifies the transport type. If you specify <code>solace</code> , <code>tervela</code> , <code>rabbitmq</code> , or <code>kafka</code> transports instead of the default <code>native</code> transport, then use the required client configuration files specified in the description of the C++ <code>C_dfESPpubsubSetPubsubLib()</code> API call.
<code>loglevel=trace   debug   info   warn   error   fatal   off</code>	Sets the logging level for the adapter. This is the same range of logging levels that you can set in the <code>C_dfESPpubsubInit()</code> <code>publish/subscribe</code> API call and in the engine <code>initialize()</code> call. The default level is <code>warn</code> .
<code>logconfigfile=file</code>	Specifies the log configuration file.
<code>tokenlocation=location</code>	Specifies the location of the file in the local file system that contains the OAuth token that is required for authentication by the <code>publish/subscribe</code> server.
<code>configfilesection=[section]</code>	Specifies the name of the section in <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.config</code> (Windows) to parse for configuration parameters. Specify the value as <code>[configfilesection]</code> .

Key-Value Pair	Description
<code>dateformat=format</code>	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 <code>dateformat</code> parameter accepts any time format that is supported by the UNIX <code>strftime</code> function.
<code>solpasswordencrypted=true   false</code>	When true, the Solace password is encrypted.
<code>restartonerror=true   false</code>	When true, specifies to restart the adapter if a fatal error is reported.
<code>transportconfigfile=file</code>	Specifies the publish/subscribe transport configuration file. For <code>solace</code> , the default is <code>./solace.cfg</code> . For <code>tervela</code> , the default is <code>./client.config</code> . For <code>rabbitmq</code> , the default is <code>./rabbitmq.cfg</code> . For <code>kafka</code> , the default is <code>./kafka.cfg</code> . <b>Note:</b> No transport configuration file is required for native transport.
<code>buspersistencequeue=queue</code>	Specifies the queue name used by Solace Guaranteed Messaging publisher.
<code>publishwithupsert=true   false</code>	When true, build events with opcode = Upsert instead of Insert.
<code>blocksize=size</code>	Sets the block size. The default value is 1.
<code>transactional=true   false</code>	When true, events are transactional.
<code>getmsgfromdestattr=true   false</code>	When true, extracts the payload from the destination attribute instead of the message body.
<code>maxevents=number</code>	Specifies the maximum number of events to publish.
<code>quiesceproject=true   false</code>	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” in SAS Event Stream Processing: Implementing Failover](#).

## Using the Teradata Connector and Adapter

### 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 `NLS_PATH` environment variable appropriately. For example, with a TTU installation in `/opt/teradata`, define `NLS_PATH` 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
<code>stream</code>	Works like a standard ESP 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 <code>tdatainsertonly</code> configuration parameter to <code>false</code> , serialization is automatically enabled in the TPT to maintain correct ordering of row data over multiple sessions.
<code>update</code>	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).
<code>load</code>	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 <code>odbc.ini</code> 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 SAS Event Stream Processing 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 142** Required Parameters for Teradata Connectors

Parameter	Description
<code>type</code>	Specifies to subscribe. Must be "sub".
<code>desttablename</code>	Specifies the target table name.
<code>tdatausername</code>	Specifies the user name for the user account on the target Teradata server.
<code>tdatauserpwd</code>	Specifies the user password for the user account on the target Teradata server.
<code>tdataatdpid</code>	Specifies the target Teradata server name
<code>tdatamaxsessions</code>	Specifies the maximum number of sessions created by the TPT to the Teradata server.
<code>tdataminsessions</code>	Specifies the minimum number of sessions created by the TPT to the Teradata server.
<code>tdatadriver</code>	Specifies the operator: <code>stream</code> , <code>update</code> , or <code>load</code> .
<code>tdatainsertonly</code>	Specifies whether events in the subscriber event stream processing window are insert only. Must be <code>true</code> when using the load operator.
<code>snapshot</code>	Specifies whether to send snapshot data.  When <code>true</code> , 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 143** Optional Parameters for Teradata Connectors

Parameter	Description
<code>rmretdel</code>	Removes all delete events from event blocks received by the subscriber that were introduced by a window retention policy.
<code>tdatabatchperiod</code>	Specifies the batch period in seconds. Required when using the <code>update</code> operator, otherwise ignored.
<code>stage1tablename</code>	Specifies the first staging table. Required when using the <code>load</code> operator, otherwise ignored.
<code>stage2tablename</code>	Specifies the second staging table. Required when using the <code>load</code> operator, otherwise ignored.
<code>connectstring</code>	Specifies the connect string used to access the target and staging tables. See <a href="#">"Using the Database Connector"</a> for Teradata <code>connectstring</code> requirements.

Parameter	Description
<code>tdatatracelevel</code>	Specifies the trace level for Teradata messages written to the trace file in the current working directory. The trace file is named <i>operator1.txt</i> . 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).
<code>configfilesection</code>	Specifies the name of the section in <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.config</code> (Windows) to parse for configuration parameters. Specify the value as <code>[configfilesection]</code> .
<code>tdatauserpwdencrypted</code>	Specifies that <code>tdatauserpwd</code> is encrypted.

## 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. For more information, see [“Using the Database Adapter”](#).

**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 144** Required Keys

Key-Value Pair	Description
<code>url=pubsubURL</code>	Specifies the standard URL in the form <code>dfESP://host:port/project/continuousquery/window</code> . Append <code>?snapshot=true   false</code> . Append <code>?collapse=true   false</code> or <code>?rmretdel=true   false</code> or both if needed.
<code>tdatadriver=stream   update   load</code>	Specifies the Teradata operator. For more information about these operators, see the documentation provided with the Teradata Tools and Utilities.
<code>tdatausername=username</code>	Specifies the Teradata user name.
<code>tdatauserpwd=password</code>	Specifies the Teradata user password.
<code>desttablename=name</code>	Specifies the name of the target table on the Teradata server.
<code>tdataatdpid=name</code>	Specifies the name of the target Teradata server.

Key-Value Pair	Description
<code>tdatamaxsessions=number</code>	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.
<code>tdataminsessions=number</code>	Specifies the minimum number of sessions on the Teradata server.
<code>tdatainsertonly=true   false</code>	When <code>true</code> , specifies that the subscribed window contains insert-only data.

**Table 145** Optional Keys

Key-Value Pair	Description
<code>gdconfig=file</code>	Specifies the guaranteed delivery configuration file.
<code>transport=native   solace   tervela   rabbitmq   kafka</code>	Specifies the transport type. If you specify <code>solace</code> , <code>tervela</code> , <code>rabbitmq</code> , or <code>kafka</code> transports instead of the default <code>native</code> transport, then use the required client configuration files specified in the description of the C++ <code>C_dfESPpubsubSetPubsubLib()</code> API call.
<code>loglevel=trace   debug   info   warn   error   fatal   off</code>	Sets the logging level for the adapter. This is the same range of logging levels that you can set in the <code>C_dfESPpubsubInit()</code> <code>publish/subscribe</code> API call and in the engine <code>initialize()</code> call. The default level is <code>warn</code> .
<code>logconfigfile=file</code>	Specifies the log configuration file.
<code>tokenlocation=location</code>	Specifies the location of the file in the local file system that contains the OAuth token that is required for authentication by the <code>publish/subscribe</code> server.
<code>configfilesection=[section]</code>	Specifies the name of the section in <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.config</code> (Windows) to parse for configuration parameters. Specify the value as <code>[configfilesection]</code> .
<code>tdatabatchperiod=seconds</code>	Specifies the batch period in seconds. This parameter is required when <code>tdatadrivers=update</code> .
<code>stage1tablename=name</code>	Specifies the name of the first staging table on the Teradata server. This parameter is required when <code>tdatadrivers=load</code> .
<code>stage2tablename=name</code>	Specifies the name of the second staging table on the Teradata server. This parameter is required when <code>tdatadrivers=load</code> .

Key-Value Pair	Description
<code>connectstring=string</code>	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 <code>tdataadapter=load</code> .
<code>tdatatracelevel=level</code>	Specifies the trace level for Teradata messages written to the trace file in the current working directory.  The trace file is named <i>operator1.txt</i> . 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).
<code>tdatauserpwdencrypted=true   false</code>	When <code>true</code> , <code>tdatauserpwd</code> is encrypted.
<code>restartonerror=true   false</code>	When <code>true</code> , specifies to restart the adapter if a fatal error is reported.
<code>transportconfigfile=file</code>	Specifies the publish/subscribe transport configuration file. For <code>solace</code> , the default is <code>./solace.cfg</code> . For <code>tervela</code> , the default is <code>./client.config</code> . For <code>rabbitmq</code> , the default is <code>./rabbitmq.cfg</code> . For <code>kafka</code> , the default is <code>./kafka.cfg</code> .  <b>Note:</b> No transport configuration file is required for native transport.

## Using the Teradata Listener Connector and Adapter

### 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](#).

**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 SAS Event Stream Processing 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). For more information about TLS libraries that are used in SAS Event Stream Processing, see [SAS Event Stream Processing: Security](#).

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 SAS Event Stream Processing 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 SAS Event Stream Processing, use the [WebSocket connector](#) with `contentType=json`.

**Table 146** Required Parameters for Teradata Listener Connectors

Parameter	Description
<code>type</code>	Specifies to subscribe. Must be "sub".
<code>ingestUrl</code>	Specifies the URL for the Listener Ingest REST API (version 1). This URL must end with <b>messages</b> .
<code>SSLCert</code>	Specifies the path to a file that contains TLS certificates securely connect to the Listener Ingest service. Listener uses TLS 1.2.
<code>sourceKey</code>	Specifies the Listener source secret key that identifies the Listener source feed to which SAS Event Stream Processing sends data. Contact your Teradata Listener administrator to obtain the source secret key.
<code>snapshot</code>	Specifies whether to send snapshot data. When <code>true</code> , 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.
<code>tdlistenerversion</code>	Specifies the Teradata Listener version (for example, 2.02).

**Table 147** Optional Parameters for Teradata Listener Connectors

Parameter	Description
<code>ingestBlocksize</code>	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.
<code>contentType</code>	Specifies the format of the data sent from SAS Event Stream Processing to Listener, either <code>json</code> or <code>plaintext</code> (comma-delimited). The default is "json".
<code>ingestDelim</code>	Specifies the character that delimits data rows in a multi-row message from SAS Event Stream Processing to the Listener Ingest REST API. The delimiter must not be a JSON punctuation character. The default is a tilde (~).

Parameter	Description
doubleprecision	Specifies the number of fractional digits in the ASCII representation of a double. The default value is 6.

## Using the Teradata Listener Adapter

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

**dfesp\_tdlister\_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 148** Required Keys

Key-Value Pair	Description
url=pubsubURL	Specifies the standard URL in the form <b>dfESP://host:port/project/continuousquery/window</b> . Append <code>?snapshot=true</code>   <code>false</code> for subscribers. Append <code>?collapse=true</code>   <code>false</code> or <code>?rmretdel=true</code>   <code>false</code> or both for subscribers if needed.
ingestUrl=url	Specifies the URL for the Listener Ingest REST API (version 1). This URL must end with <b>messages</b> .
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 SAS Event Stream Processing sends data. Contact your Teradata Listener administrator to obtain the source key.

**Table 149** 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 SAS Event Stream Processing 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 SAS Event Stream Processing to the Listener. The default is "json".
debug=true   false	Enables debugging. The default is <code>false</code> .
gdconfig=file	Specifies the guaranteed delivery configuration file.

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, then use the required client configuration files specified in the description of the C++ <code>C_dfESPPubsubSetPubsubLib()</code> 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 <code>C_dfESPPubsubInit()</code> publish/subscribe API call and in the engine <code>initialize()</code> 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 <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.config</code> (Windows) to parse for configuration parameters. Specify the value as <code>[configfilesection]</code> .
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 <code>./solace.cfg</code> . For tervela, the default is <code>./client.config</code> . For rabbitmq, the default is <code>./rabbitmq.cfg</code> . For kafka, the default is <code>./kafka.cfg</code> . <b>Note:</b> No transport configuration file is required for native transport.
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).

## Using the Tervela Connector and Adapter

### 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 ESP 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 `tvuserid` and `tvpassword` 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.*tvclientname* (where *tvclientname* is a connector configuration parameter). This message contains the following information:

- The mapping of the ESP engine name to a `host:port` field potentially used by an ESP 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 ESP engines running the involved connectors must all be identical. This set of ESP 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 SAS Event Stream Processing 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 150** Required Parameters for Subscriber Tervela Connectors

Parameter	Description
<code>type</code>	Specifies to subscribe. Must be "sub".
<code>tvauserid</code>	Specifies a user name defined in the Tervela TPM. Publish-topic entitlement rights must be associated with this user name.
<code>tvapassword</code>	Specifies the password associated with <code>tvauserid</code> .
<code>tvaprimarystmx</code>	Specifies the host name or IP address of the primary TMX.
<code>tvatopic</code>	Specifies the topic name for the topic to which to subscribe. This topic must be configured on the TPM for the GD service and <code>tvauserid</code> must be assigned the Guaranteed Delivery subscribe rights for this Topic in the TPM.
<code>tvaclientname</code>	Specifies the client name associated with the Tervela Guaranteed Delivery context. If hot failover is enabled, this name must match the <code>tvaclientname</code> of other subscriber connectors in the failover group. Otherwise, the name must be unique among all instances of Tervela connectors.
<code>tvamaxoutstand</code>	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 <code>host/port</code> string sent in the metadata message published by the connector on topic <code>SAS.META.tvaclientname</code> when it starts.
snapshot	Specifies whether to send snapshot data.  When <code>true</code> , 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 151** Required Parameters for Publisher Tervela Connectors

Parameter	Description
type	Specifies to publish. Must be "pub".
tvuserid	Specifies a user name defined in the Tervela TPM. Subscribe-topic entitlement rights must be associated with this user name.
tvpassword	Specifies the password associated with <code>tvuserid</code> .
tvprimarytmx	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.
tvsubname	Specifies the name assigned to the Guaranteed Delivery subscription being created. The combination of this name and <code>tvaclientname</code> 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 <code>host:port</code> string sent in the metadata message published by the connector on topic <code>SAS.META.tvaclientname</code> when it starts.

**Table 152** 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.

Parameter	Description
tvasecondarytmx	Specifies the host name or IP address of the secondary TMX. Required if 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).
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 <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.config</code> (Windows) to parse for configuration parameters. Specify the value as <code>[configfilesection]</code> .
protofile	Specifies the <code>.proto</code> 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 <code>protomsg</code> parameter.
protomsg	Specifies the name of a Google Protocol Buffers message in the <code>.proto</code> file that you specified with the <code>protofile</code> 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 <code>dateformat</code> parameter accepts any time format that is supported by the UNIX <code>strftime</code> function.
tvapasswordencrypted	Specifies that <code>tvapassword</code> is encrypted.

**Table 153** 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.

Parameter	Description
<code>tvalogfile</code>	Causes the connector to log to the specified file instead of to syslog (on Linux or Solaris) or Tervela.log (on Windows)
<code>configfilesection</code>	Specifies the name of the section in <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.config</code> (Windows) to parse for configuration parameters. Specify the value as <code>[configfilesection]</code> .
<code>protofile</code>	Specifies the <code>.proto</code> 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 <code>protomsg</code> parameter.
<code>protomsg</code>	Specifies the name of a Google Protocol Buffers message in the <code>.proto</code> file that you specified with the <code>protofile</code> parameter. Event blocks are converted into this message.
<code>json</code>	Enables transport of event blocks encoded as JSON messages.
<code>publishwithupsert</code>	Specifies to build events with <code>opcode = Upsert</code> instead of <code>opcode = Insert</code> .
<code>dateformat</code>	Specifies the format of <code>DATE</code> and <code>STAMP</code> fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds ( <code>DATE</code> ) or microseconds ( <code>STAMP</code> ) since epoch. The <code>dateformat</code> parameter accepts any time format that is supported by the UNIX <code>strftime</code> function.
<code>tvpasswordencrypted</code>	Specifies that <code>tvpassword</code> is encrypted.
<code>maxevents</code>	Specifies the maximum number of events to publish.

## 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 154 Required Keys

Key-Value Pair	Description
<code>type=sub</code>	Specifies a subscriber adapter.
<code>url=pubsubURL</code>	Specifies the standard URL in the form <code>dfESP://host:port/project/continuousquery/window</code> . Append <code>?snapshot=true   false</code> . Append <code>?collapse=true   false</code> or <code>?rmretdel=true   false</code> or both if needed.
<code>tvuserid=username</code>	Specifies the Tervela user name.
<code>tvapassword=password</code>	Specifies the Tervela password.
<code>tvaprimarystmx=tmx</code>	Specifies the Tervela primary TMX. The TMX-500 Message Switch is the primary appliance used to communicate across the Tervela data fabric.
<code>tvatopic=topic</code>	Specifies the Tervela topic.
<code>tvaclientname=name</code>	Specifies the Tervela client name.
<code>urlhostport=string</code>	Specifies the <code>host:port</code> string sent in connector metadata message
<code>tvamaxoutstand=number</code>	Specifies the Tervela maximum number of unacknowledged messages.
<code>numbufferedmsgs=number</code>	Specifies the maximum number of messages buffered by a standby subscriber connector.

Table 155 Optional Keys

Key-Value Pair	Description
<code>tvasecondarystmx=tmx</code>	Specifies the Tervela secondary TMX.
<code>tvalogfile=file</code>	Specifies the Tervela log file. The default is "syslog".
<code>protofile=file</code>	Specifies the <code>.proto</code> file to be used for Google protocol buffer support.
<code>protomsg=message</code>	Specifies the message itself in the <code>.proto</code> file that is specified by the <code>protofile</code> parameter.
<code>json=true   false</code>	When <code>true</code> , transport JSON messages instead of event blocks.
<code>gdconfig=file</code>	Specifies the guaranteed delivery configuration file.

Key-Value Pair	Description
transport=native   solace   tervela   rabbitmq   kafka	Specifies the transport type. If you specify <code>solace</code> , <code>tervela</code> , <code>rabbitmq</code> , or <code>kafka</code> transports instead of the default <code>native</code> transport, then use the required client configuration files specified in the description of the C++ <code>C_dfESPpubsubSetPubsubLib()</code> 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 <code>C_dfESPpubsubInit()</code> <code>publish/subscribe</code> API call and in the engine <code>initialize()</code> call. The default level is <code>warn</code> .
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 <code>publish/subscribe</code> server.
configfilesection=[section]	Specifies the name of the section in <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.config</code> (Windows) to parse for configuration parameters. Specify the value as <code>[configfilesection]</code> .
dateformat=format	Specifies the format of <code>DATE</code> and <code>STAMP</code> fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds ( <code>DATE</code> ) or microseconds ( <code>STAMP</code> ) since epoch. The <code>dateformat</code> parameter accepts any time format that is supported by the UNIX <code>strftime</code> function.
tvapasswordencrypted=true   false	When <code>true</code> , <code>tvapassword</code> is encrypted
restartonerror=true   false	When <code>true</code> , specifies to restart the adapter if a fatal error is reported.
transportconfigfile=file	Specifies the <code>publish/subscribe</code> transport configuration file. For <code>solace</code> , the default is <code>./solace.cfg</code> . For <code>tervela</code> , the default is <code>./client.config</code> . For <code>rabbitmq</code> , the default is <code>./rabbitmq.cfg</code> . For <code>kafka</code> , the default is <code>./kafka.cfg</code> . <b>Note:</b> 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 156 Required Keys

Key-Value Pair	Description
<code>type=pub</code>	Specifies a publisher adapter.
<code>url=pubsubURL</code>	Specifies the standard URL in the form <code>dfESP://host:port/project/continuousquery/window</code> .
<code>tvuserid=username</code>	Specifies the Tervela user name.
<code>tvapassword=password</code>	Specifies the Tervela password.
<code>tvaprimarystmx=tmx</code>	Specifies the Tervela primary TMX. The TMX-500 Message Switch is the primary appliance used to communicate across the Tervela data fabric.
<code>tvatopic=topic</code>	Specifies the Tervela topic.
<code>tvaclientname=name</code>	Specifies the Tervela client name.
<code>urlhostport=string</code>	Specifies the <code>host:port</code> string sent in connector metadata message
<code>tvasubname=name</code>	Specifies the Tervela name of the Guaranteed Delivery subscription

Table 157 Optional Keys

Key-Value Pair	Description
<code>tvasecondarystmx=tmx</code>	Specifies the Tervela secondary TMX.
<code>tvalogfile=file</code>	Specifies the Tervela log file. The default is "syslog".
<code>protofile=file</code>	Specifies the <code>.proto</code> file to be used for Google protocol buffer support.
<code>protomsg=message</code>	Specifies the message itself in the <code>.proto</code> file that is specified by the <code>protofile</code> parameter.
<code>json=true   false</code>	When <code>true</code> , transport JSON messages instead of event blocks.
<code>gdconfig=file</code>	Specifies the guaranteed delivery configuration file.
<code>transport=native   solace   tervela   rabbitmq   kafka</code>	Specifies the transport type. If you specify <code>solace</code> , <code>tervela</code> , <code>rabbitmq</code> , or <code>kafka</code> transports instead of the default <code>native</code> transport, then use the required client configuration files specified in the description of the C++ <code>C_dfESPpubsubSetPubsubLib()</code> 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 <code>C_dfESppubsubInit()</code> publish/subscribe API call and in the engine <code>initialize()</code> call. The default level is <code>warn</code> .
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 <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.config</code> (Windows) to parse for configuration parameters. Specify the value as <code>[configfilesection]</code> .
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 <code>dateformat</code> parameter accepts any time format that is supported by the UNIX <code>strftime</code> function.
tvapasswordencrypted=true   false	When <code>true</code> , <code>tvapassword</code> is encrypted
restartonerror=true   false	When <code>true</code> , specifies to restart the adapter if a fatal error is reported.
transportconfigfile=file	Specifies the publish/subscribe transport configuration file. For <code>solace</code> , the default is <code>./solace.cfg</code> . For <code>tervela</code> , the default is <code>./client.config</code> . For <code>rabbitmq</code> , the default is <code>./rabbitmq.cfg</code> . For <code>kafka</code> , the default is <code>./kafka.cfg</code> . <b>Note:</b> No transport configuration file is required for native transport.
publishwithupsert=true   false	When <code>true</code> , build events with <code>opcode = Upsert</code> instead of <code>Insert</code> .
maxevents=number	Specifies the maximum number of events to publish.
quiesceproject=true   false	When <code>true</code> , 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” in SAS Event Stream Processing: Implementing Failover](#).

## Using the Tibco Rendezvous Connector and Adapter

### 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 158** Required Parameters for Subscriber Tibco RV Connectors

Parameter	Description
<code>type</code>	Specifies to subscribe. Must be "sub".
<code>tibrvsubject</code>	Specifies the Tibco RV subject name.
<code>tibrvtype</code>	Specifies <code>binary</code> , <code>csv</code> , <code>json</code> , <code>protobuf</code> , or the name of a string field in the subscribed window schema.

Parameter	Description
snapshot	Specifies whether to send snapshot data. When <code>true</code> , 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 159** 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 <code>binary</code> , <code>csv</code> , <code>json</code> , <code>protobuf</code> , or <code>opaquestring</code> . For <code>opaquestring</code> , the Source window schema is assumed to be <code>index:int64,message:string</code> .

**Table 160** 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 <code>dateformat</code> parameter accepts any time format that is supported by the UNIX <code>strftime</code> function.
configfilesection	Specifies the name of the section in <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.config</code> (Windows) to parse for configuration parameters. Specify the value as <code>[configfilesection]</code> .

Parameter	Description
protofile	Specifies the <code>.proto</code> 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 <code>protomsg</code> parameter.  This parameter is ignored when <code>tibrvtype</code> is not set to <code>protobuf</code> .
protomsg	Specifies the name of a Google Protocol Buffers message in the <code>.proto</code> file that you specified with the <code>protofile</code> parameter. Event blocks are converted into this message.  This parameter is ignored when <code>tibrvtype</code> is not set to <code>protobuf</code> .
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 161** 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 <code>dateformat</code> parameter accepts any time format that is supported by the UNIX <code>strftime</code> function.
tibrvservice	Specifies the Rendezvous service used by the Tibco RV transport created by the connector. The default service name is <code>rendezvous</code> .
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.
configfilesection	Specifies the name of the section in <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.config</code> (Windows) to parse for configuration parameters. Specify the value as <code>[configfilesection]</code> .

Parameter	Description
<code>ignorecsvparseerrors</code>	Specifies that when a field in an input CSV event cannot be parsed, the event is dropped, an error is logged, and publishing continues.
<code>protofile</code>	Specifies the <code>.proto</code> 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 <code>protomsg</code> parameter.  This parameter is ignored when <code>tibrvtype</code> is not set to <code>protobuf</code> .
<code>protomsg</code>	Specifies the name of a Google Protocol Buffers message in the <code>.proto</code> file that you specified with the <code>protofile</code> parameter. Event blocks are converted into this message.  This parameter is ignored when <code>tibrvtype</code> is not set to <code>protobuf</code> .
<code>csvfielddelimiter</code>	Specifies the character delimiter for field data in input CSV events. The default delimiter is the <code>,</code> character.
<code>noautogenfield</code>	Specifies that input events are missing the key field that is autogenerated by the source window.
<code>publishwithupsert</code>	Specifies to build events with <code>opcode=Upsert</code> instead of <code>opcode=Insert</code> .
<code>addcsvopcode</code>	Prepends an opcode and comma to input CSV events. The opcode is <code>Insert</code> unless <code>publishwithupsert</code> is enabled.
<code>addcsvflags</code>	Specifies the event type to insert into input CSV events (with a comma). Valid values are <code>normal</code> and <code>partialupdate</code> .
<code>maxevents</code>	Specifies the maximum number of events to publish.
<code>stripnullsfromcsv</code>	Strip all nulls from input CSV events.

## 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="string"`).

## Subscriber Usage

Table 162 Required Keys

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

Table 163 Optional Keys

Key-Value Pair	Description
<code>tibrvservice=service</code>	Specifies the Tibco RV service. Rendezvous daemon processes communicate using UDP or PGM services.
<code>tibrvnetwork=network</code>	Specifies the Tibco RV network.
<code>tibrvdaemon=daemon</code>	Specifies the Tibco RV daemon.
<code>dateformat=format</code>	Specifies the format of <code>DATE</code> and <code>STAMP</code> fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds ( <code>DATE</code> ) or microseconds ( <code>STAMP</code> ) since epoch. The <code>dateformat</code> parameter accepts any time format that is supported by the UNIX <code>strftime</code> function.
<code>protofile=file</code>	Specifies the <code>.proto</code> file to be used for Google protocol buffer support. This key is ignored when <code>tibrvtype</code> is not set to <code>protobuf</code> .
<code>protomsg=message</code>	Specifies the message itself in the <code>.proto</code> file that is specified by the <code>protofile</code> parameter. This key is ignored when <code>tibrvtype</code> is not set to <code>protobuf</code> .
<code>gdconfig=file</code>	Specifies the guaranteed delivery configuration file.
<code>transport=native</code>   <code>solace</code>   <code>tervela</code>   <code>rabbitmq</code>   <code>kafka</code>	Specifies the transport type. If you specify <code>solace</code> , <code>tervela</code> , <code>rabbitmq</code> , or <code>kafka</code> transports instead of the default <code>native</code> transport, then use the required client configuration files specified in the description of the C++ <code>C_dfESPpubsubSetPubsubLib()</code> 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 <code>C_dfESppubsubInit()</code> publish/subscribe API call and in the engine <code>initialize()</code> call. The default level is <code>warn</code> .
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 <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.config</code> (Windows) to parse for configuration parameters. Specify the value as <code>[configfilesection]</code> .
restartonerror=true   false	When <code>true</code> , specifies to restart the adapter if a fatal error is reported.
transportconfigfile=file	Specifies the publish/subscribe transport configuration file. For <code>solace</code> , the default is <code>./solace.cfg</code> . For <code>tervela</code> , the default is <code>./client.config</code> . For <code>rabbitmq</code> , the default is <code>./rabbitmq.cfg</code> . For <code>kafka</code> , the default is <code>./kafka.cfg</code> . <b>Note:</b> No transport configuration file is required for native transport.
csvmsgperevent=true   false	When <code>true</code> , send one message per event. By default, one message is sent per transactional event block.
csvmsgpereventblock=true   false	When <code>true</code> , 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 164** Required Keys

Key-Value Pair	Description
type=pub	Specifies a publisher adapter.
url=pubsubURL	Specifies the standard URL in the form <code>dfESP://host:port/project/continuousquery/window</code> .
tibrvsubject=subject	Specifies the Tibco RV subject.

Key-Value Pair	Description
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 165** 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.
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 <b>protobuf</b> .
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 <b>protobuf</b> .
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++ <a href="#">C_dfESPpubsubSetPubsubLib()</a> 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.

Key-Value Pair	Description
configfilesection=[section]	Specifies the name of the section in <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.config</code> (Windows) to parse for configuration parameters. Specify the value as <code>[configfilesection]</code> .
restartonerror=true   false	When <code>true</code> , 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 <code>./solace.cfg</code> . For tervela, the default is <code>./client.config</code> . For rabbitmq, the default is <code>./rabbitmq.cfg</code> . For kafka, the default is <code>./kafka.cfg</code> . <b>Note:</b> No transport configuration file is required for native transport.
blocksize=size	Sets the block size. The default value is 1.
transactional=true   false	When <code>true</code> , 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 <code>,</code> character.
noautogenfield=true   false	When <code>true</code> , specifies that input events are missing the key field that is autogenerated by the Source window.
publishwithupsert=true   false	When <code>true</code> , build events with opcode = Upsert instead of Insert.
addcsvopcode= true   false	When <code>true</code> , prepends an opcode and comma to write CSV events. The opcode is Insert unless <code>publishwithupsert</code> is <code>true</code> .
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 <code>true</code> , quiesces the project after all events are injected into the Source window.
stripnullsfromcsv=true   false	When <code>true</code> , strip all nulls from input CSV events.

## Publisher Failover

For information about implementing hot failover for publisher adapters, see [“Publisher Adapter Failover with Kafka”](#) in *SAS Event Stream Processing: Implementing Failover*.

## Using the Timer Connector and Adapter

### 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 166** Required Parameters for Timer Connectors

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

**Table 167** Optional Parameters for Timer Connectors

Parameter	Description
<code>label</code>	Specifies the string to be written to the source window "label" field. The default value is the connector name.
<code>timeformat</code>	Specifies the format of the <code>basetime</code> parameter. The default value is <code>"%Y-%m-%d %H:%M:%S"</code> .
<code>transactional</code>	Sets the event block type to <code>transactional</code> . The default value is <code>normal</code> .

<code>configfilesection</code>	Specifies the name of the section in <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.config</code> (Windows) to parse for configuration parameters. Specify the value as <code>[configfilesection]</code> .
<code>publishwithupsert</code>	Builds events with <code>opcode = Upsert</code> instead of <code>opcode = Insert</code> .
<code>maxevents</code>	Specifies the maximum number of events to publish.

## 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 168** Required Keys

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

**Table 169** Optional Keys

Key-Value Pair	Description
<code>gdconfig=file</code>	Specifies the guaranteed delivery configuration file.
<code>transport=native   solace   tervela   rabbitmq   kafka</code>	Specifies the transport type. If you specify <code>solace</code> , <code>tervela</code> , <code>rabbitmq</code> , or <code>kafka</code> transports instead of the default <code>native</code> transport, then use the required client configuration files specified in the description of the C++ <code>C_dfESPpubsubSetPubsubLib()</code> API call.
<code>loglevel=trace   debug   info   warn   error   fatal   off</code>	Sets the logging level for the adapter. This is the same range of logging levels that you can set in the <code>C_dfESPpubsubInit()</code> <code>publish/subscribe</code> API call and in the engine <code>initialize()</code> call. The default level is <code>warn</code> .
<code>logconfigfile=file</code>	Specifies the log configuration file.

Key-Value Pair	Description
<code>tokenlocation=location</code>	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.
<code>configfilesection=[section]</code>	Specifies the name of the section in <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.config</code> (Windows) to parse for configuration parameters. Specify the value as <code>[configfilesection]</code> .
<code>transactional=true   false</code>	When <code>true</code> , events are transactional.
<code>publishwithupsert=true   false</code>	When <code>true</code> , build events with opcode = Upsert instead of Insert.
<code>restartonerror=true   false</code>	When <code>true</code> , specifies to restart the adapter if a fatal error is reported.
<code>transportconfigfile=file</code>	Specifies the publish/subscribe transport configuration file. For solace, the default is <code>./solace.cfg</code> . For tervela, the default is <code>./client.config</code> . For rabbitmq, the default is <code>./rabbitmq.cfg</code> . For kafka, the default is <code>./kafka.cfg</code> . <b>Note:</b> No transport configuration file is required for native transport.
<code>label=string</code>	Specifies the string to be written to the Source window <code>label</code> field. The default value is the connector name.
<code>timeformat=format</code>	Specifies the format of the <code>basetime</code> parameter. The default value is <code>%Y-%m-%d %H:%M:%S</code> .
<code>maxevents=number</code>	Specifies the maximum number of events to publish.
<code>quiesceproject=true   false</code>	When <code>true</code> , quiesces the project after all events are injected into the Source window.

## Using the URL Connector

The URL connector enables you to bring data into SAS Event Stream Processing over a URL-based connection, transform the data, and publish events into an event stream processing model. Some examples of data that you can publish into SAS Event Stream Processing 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 ESP server as events.

The XML defining the event stream processing 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 ESP server.

For sample configuration files, see the URL connector examples located in the examples folder of your SAS Event Stream Processing 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 event 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 ESP server using a single publisher and several request definitions.

Use the URL connector only to publish events.

**Table 170** Required Parameters for Publisher URL Connectors

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

**Table 171** 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.

## Using the UVC Connector and Adapter

### Using the UVC Connector

The UVC connector enables you to publish photos taken by a V4L2-compatible camera to SAS Event Stream Processing. 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. SAS Event Stream Processing 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 172** Required Parameters for the UVC Publisher Connector

Parameter	Description
<code>type</code>	Specifies to publish. Must be "pub".

**Table 173** Optional Parameters for the UVC Publisher Connector

Parameter	Description
<code>frame_rate</code>	Specifies the frames per second that the camera streams. Must be a double. The default value is 15.
<code>format_in</code>	Specifies the image format of captured photos. The default is <code>mjpg</code> . <code>yuyv</code> , an uncompressed image format, is also supported.
<code>format_out</code>	Specifies the image format that the connector publishes. The default is <code>mjpg</code> . <code>yuyv</code> , an uncompressed image format, is supported only when <code>format_in</code> is <code>yuyv</code> .

Parameter	Description
width	Specifies the width of the photo. Not all resolutions are supported. Consult the camera menu or use the <code>v4l2-ctl</code> 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 <code>v4l2-ctl</code> command-line utility for a list of supported resolutions.
brightness	Specifies the brightness of the photo. Consult the camera menu or use the <code>v4l2-ctl</code> command-line utility for a list of supported brightness values.
gain	Specifies the gain of the photo. Consult the camera menu or use the <code>v4l2-ctl</code> command-line utility for a list of supported gain values.
saturation	Specifies the saturation of the photo. Consult the camera menu or use the <code>v4l2-ctl</code> command-line utility for a list of supported saturation values.
contrast	Specifies the contrast of the photo. Consult the camera menu or use the <code>v4l2-ctl</code> 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 <code>/dev/videoX</code> .
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 <code>predelay</code> is <i>n</i> , the connector waits <i>n</i> seconds to start. If the connector fails to detect the camera, the ESP server exits with a fatal error.
maxevents	Specifies the maximum number of events to publish.

## Using the UVC Publisher Adapter

The UVC adapter enables you to publish photos taken by a V4L2-compatible camera to SAS Event Stream Processing.

**dfesp\_ufv\_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 174 Required Keys

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

Table 175 Optional Keys

Key-Value Pair	Description
transport=native   solace   tervela   rabbitmq   kafka	Specifies the transport type. If you specify <code>solace</code> , <code>tervela</code> , <code>rabbitmq</code> , or <code>kafka</code> transports instead of the default <code>native</code> transport, use the required client configuration files specified in the description of the C++ <code>C_dfESPpubsubSetPubsubLib()</code> 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 <code>C_dfESPpubsubInit()</code> <code>publish/subscribe</code> API call and in the engine <code>initialize()</code> call. The default level is <code>warn</code> .
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 <code>publish/subscribe</code> server.
configfilesection=[section]	Specifies the name of the section in <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.config</code> (Windows) to parse for configuration parameters. Specify the value as <code>[configfilesection]</code> .
blocksize=size	Sets the block size. The default value is 1.
transactional=true   false	When <code>true</code> , events are transactional.
publishwithupsert=true   false	When <code>true</code> , build events with <code>opcode = Upsert</code> instead of <code>Insert</code> .
restartonerror=true   false	When <code>true</code> , 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 <code>"/dev/video0"</code> .
format_in=format	Specifies the image format of captured photos. The default is <code>mjpg</code> . <code>yuyv</code> , an uncompressed image format, is also supported.

Key-Value Pair	Description
<code>format_out=format</code>	Specifies the photo format output from the adapter. The default is <code>mjpg</code> . <code>yuyv</code> is an uncompressed image format. It is supported only when <code>format_in</code> is <code>yuyv</code> .
<code>width=width</code>	Specifies the width for the frame resolution. The default is 640.
<code>height=height</code>	Specifies the height for the frame resolution. The default is 480.
<code>brightness=level</code>	Specifies the brightness level. The default value is 0. Consult the camera menu or use the <code>v4l2-ctl</code> command-line utility for a list of supported brightness values.
<code>gain=gain</code>	Specifies the gain. The default value is 0. Consult the camera menu or use the <code>v4l2-ctl</code> command-line utility for a list of supported gain values.
<code>saturation=saturation</code>	Specifies the saturation. The default value is 0. Consult the camera menu or use the <code>v4l2-ctl</code> command-line utility for a list of supported saturation values.
<code>contrast=contrast</code>	Specifies the contrast. The default value is 0. Consult the camera menu or use the <code>v4l2-ctl</code> command-line utility for a list of supported contrast values.
<code>predelay=seconds</code>	Specifies the number of seconds to delay before a retry if unable to start camera. The default is 0.
<code>transportconfigfile=file</code>	Specifies the publish/subscribe transport configuration file. For <code>solace</code> , the default is <code>./solace.cfg</code> . For <code>tervela</code> , the default is <code>./client.config</code> . For <code>rabbitmq</code> , the default is <code>./rabbitmq.cfg</code> . For <code>kafka</code> , the default is <code>./kafka.cfg</code> . <b>Note:</b> No transport configuration file is required for native transport.
<code>maxevents=number</code>	Specifies the maximum number of events to publish.
<code>quiesceproject=true   false</code>	When <code>true</code> , 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”](#) in *SAS Event Stream Processing: Implementing Failover*.

## 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](#).

The WebSocket connector enables you to read data over a WebSocket-based connection and publish the data into SAS Event Stream Processing. 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 SAS Event Stream Processing.

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 176** Required Parameters for WebSocket Connectors

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

**Table 177** Optional Parameters for WebSocket Connectors

Parameter	Description
sslCertificate	Specifies the location of the TLS certificate to use when connecting to a secure server.

Parameter	Description
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.

---

## 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 a SAS Event Stream Processing 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 a SAS Event Stream Processing event is always `base64` encoded.

Use the following parameters for MQ connectors.

**Table 178** Required Parameters for Subscriber MQ Connectors

Parameter	Description
<code>type</code>	Specifies to subscribe. Must be "sub".
<code>mqtype</code>	Specifies <code>binary</code> , <code>csv</code> , <code>json</code> , <code>xml</code> , <code>protobuf</code> , or the name of a string field in the subscribed window schema.

Parameter	Description
snapshot	<p>Specifies whether to send snapshot data.</p> <p>When <code>true</code>, 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.</p>

**Table 179** Required Parameters for Publisher MQ Connectors

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

**Table 180** Optional Parameters for Subscriber MQ Connectors

Parameter	Description
mqtopic	Specifies the MQ topic name. Required if <code>mqqueue</code> is not configured.
mqqueue	Specifies the MQ queue name. Required if <code>mqtopic</code> is not configured.
collapse	Enables conversion of <code>UPDATE_BLOCK</code> events to make subscriber output publishable.
queuemanager	Specifies the MQ queue manager.
dateformat	Specifies the format of <code>DATE</code> and <code>STAMP</code> fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds ( <code>DATE</code> ) or microseconds ( <code>STAMP</code> ) since epoch. The <code>dateformat</code> parameter accepts any time format that is supported by the UNIX <code>strftime</code> 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 <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.config</code> (Windows) to parse for configuration parameters. Specify the value as <code>[configfilesection]</code> .
protofile	<p>Specifies the <code>.proto</code> 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 <code>protomsg</code> parameter.</p> <p>This parameter is ignored when <code>mqtype</code> is not set to <code>protobuf</code>.</p>

Parameter	Description
protomsg	Specifies the name of a Google Protocol Buffers message in the <code>.proto</code> file that you specified with the <code>protofile</code> parameter. Event blocks are converted into this message.  This parameter is ignored when <code>mqtype</code> is not set to <code>protobuf</code> .
usecorrelid	Copies the value of the <code>correlid</code> 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 181** Optional Parameters for Publisher MQ Connectors

Parameter	Description
mqtopic	Specifies the MQ topic name. Required if <code>mqqueue</code> is not configured.
mqqueue	Specifies the MQ queue name. Required if <code>mqtopic</code> is not configured.
mqsubname	Specifies the MQ subscription name. Required if <code>mqqueue</code> 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.
dateformat	Specifies the format of <code>DATE</code> and <code>STAMP</code> fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds ( <code>DATE</code> ) or microseconds ( <code>STAMP</code> ) since epoch. The <code>dateformat</code> parameter accepts any time format that is supported by the UNIX <code>strftime</code> function.
queuemanager	Specifies the MQ queue manager.
configfilesection	Specifies the name of the section in <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.config</code> (Windows) to parse for configuration parameters. Specify the value as <code>[configfilesection]</code> .
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.

Parameter	Description
<code>protofile</code>	Specifies the <code>.proto</code> 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 <code>protomsg</code> parameter.  This parameter is ignored when <code>mqtype</code> is not set to <code>protobuf</code> .
<code>protomsg</code>	Specifies the name of a Google Protocol Buffers message in the <code>.proto</code> file that you specified with the <code>protofile</code> parameter. Event blocks are converted into this message.  This parameter is ignored when <code>mqtype</code> is not set to <code>protobuf</code> .
<code>csvfielddelimiter</code>	Specifies the character delimiter for field data in input CSV events. The default delimiter is the <code>,</code> character.
<code>noautogenfield</code>	Specifies that input events are missing the key field that is autogenerated by the source window.
<code>publishwithupsert</code>	Builds events with <code>opcode=Upsert</code> instead of <code>Insert</code> .
<code>addcsvopcode</code>	Prepends an opcode and comma to input CSV events. The opcode is <code>Insert</code> unless <code>publishwithupsert</code> is enabled.
<code>addcsvflags</code>	Specifies the event type to insert into input CSV events (with a comma). Valid values are <code>normal</code> and <code>partialupdate</code> .
<code>usecorrelid</code>	Copies the value of the MQ message correlation ID into the <code>correlid</code> field in every SAS Event Stream Processing event.
<code>ignoremqmdformat</code>	Specifies to ignore the value of the Message Descriptor Format parameter, and assume the message format is compatible with the <code>mqtype</code> parameter setting.
<code>maxevents</code>	Specifies the maximum number of events to publish.
<code>stripnullsfromcsv</code>	Strip all nulls from input CSV events.

## 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 182 Required Keys

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

Table 183 Optional Keys

Key-Value Pair	Description
<code>mqtopic=name</code>	Specifies the MQ topic name.
<code>queuemanager=name</code>	Specifies the MQ queue manager name.
<code>dateformat=format</code>	Specifies the format of <code>DATE</code> and <code>STAMP</code> fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds ( <code>DATE</code> ) or microseconds ( <code>STAMP</code> ) since epoch. The <code>dateformat</code> parameter accepts any time format that is supported by the UNIX <code>strftime</code> function.
<code>protofile=file</code>	Specifies the <code>.proto</code> file to be used for Google protocol buffer support. This key is ignored when <code>mqtype</code> is not set to <code>protobuf</code> .
<code>protomsg=message</code>	Specifies the message itself in the <code>.proto</code> file that is specified by the <code>protofile</code> parameter. This key is ignored when <code>mqtype</code> is not set to <code>protobuf</code> .
<code>gdconfig=file</code>	Specifies the guaranteed delivery configuration file.
<code>transport=native</code>   <code>solace</code>   <code>tervela</code>   <code>rabbitmq</code>   <code>kafka</code>	Specifies the transport type. If you specify <code>solace</code> , <code>tervela</code> , <code>orrabbitmq</code> or <code>kafka</code> transports instead of the default <code>native</code> transport, use the required client configuration files specified in the description of the C++ <code>C_dfESPpubsubSetPubsubLib()</code> API call.
<code>loglevel=trace</code>   <code>debug</code>   <code>info</code>   <code>warn</code>   <code>error</code>   <code>fatal</code>   <code>off</code>	Sets the logging level for the adapter. This is the same range of logging levels that you can set in the <code>C_dfESPpubsubInit()</code> <code>publish/subscribe</code> API call and in the engine <code>initialize()</code> call. The default level is <code>warn</code> .

Key-Value Pair	Description
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 <code>/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config</code> (Linux) or <code>%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.config</code> (Windows) to parse for configuration parameters. Specify the value as <code>[configfilesection]</code> .
usecorrelid=true   false	When true, copies the MQ correlation ID to or from the <code>correlid</code> 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.
mqueue=name	Specifies the MQ queue name.
transportconfigfile=file	Specifies the publish/subscribe transport configuration file. For solace, the default is <code>./solace.cfg</code> . For tervela, the default is <code>./client.config</code> . For rabbitmq, the default is <code>./rabbitmq.cfg</code> . For kafka, the default is <code>./kafka.cfg</code> . <b>Note:</b> 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 representation of a double. The default value is 6.

## Publisher Usage

Table 184 Required Keys

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

Key-Value Pair	Description
<code>mqtype=binary   csv   json   xml   protobuf   opaquestring</code>	Specifies a message form of binary, csv, json, xml, or protobuf. For opaquestring, the Source window schema is assumed to be <code>index:int64,message:string</code> .

**Table 185** *Optional Keys*

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

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

## Publisher Failover

For information about implementing hot failover for publisher adapters, see [“Publisher Adapter Failover with Kafka” in SAS Event Stream Processing: Implementing Failover](#).

# 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](#). 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.

---

## 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
<code>dfESPconnectorInfo</code>	Specifies the connector name, publish/subscribe type, initialization function pointer, and configuration data pointers.
<code>subRequiredConfig</code>	Specifies an array of <code>dfESPconnectorParmInfo_t</code> entries listing required configuration parameters for a subscriber.
<code>sizeofSubRequiredConfig</code>	Specifies the number of entries in <code>subRequiredConfig</code> .
<code>pubRequiredConfig</code>	Specifies an array of <code>dfESPconnectorParmInfo_t</code> entries listing required configuration parameters for a publisher.
<code>sizeofPubRequiredConfig</code>	Specifies the number of entries in <code>pubRequiredConfig</code> .
<code>subOptionalConfig</code>	Specifies an array of <code>dfESPconnectorParmInfo_t</code> entries listing optional configuration parameters for a subscriber.
<code>sizeofSubOptionalConfig</code>	Specifies the number of entries in <code>subOptionalConfig</code> .
<code>pubOptionalConfig</code>	Specifies an array of <code>dfESPconnectorParmInfo_t</code> entries listing optional configuration parameters for a publisher.
<code>sizeofPubOptionalConfig</code>	Specifies the number of entries in <code>pubOptionalConfig</code> .

A connector must define these static methods:

Static Method	Description
<code>dfESPconnector *initialize(dfESPengine *engine, dfESPpsLib_t psLib)</code>	Returns an instance of the connector.
<code>dfESPconnectorInfo *getConnectorInfo()</code>	Returns the <code>dfESPconnectorInfo</code> 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
<code>start()</code>	Starts the connector. Must call base class method <code>checkConfig()</code> to validate connector configuration before starting. Must also call base class method <code>start()</code> . Must set variable <code>_started = true</code> upon success.
<code>stop()</code>	Stops the connector. Must call base class method <code>stop()</code> . Must leave the connector in a state whereby <code>start()</code> can be subsequently called to restart the connector.
<code>callbackFunction()</code>	Specifies the method invoked by the engine to pass event blocks generated by the window to which it is connected.
<code>errorCallbackFunction()</code>	Specifies the method invoked by the engine to report errors detected by the engine. Must call user callback function <code>_errorCallback</code> , if nonzero.
<code>setupCallbackFunction()</code>	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 SAS Event Stream Processing 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.