SAS® Event Stream Processing 6.2: Connectors and Adapters

What Are Connectors and Adapters?

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.

IMPORTANT The following connectors and adapters are no longer supported or included in SAS Viya 3.5 revision 24w44. No updates to this functionality are available for earlier releases.

- HDFS (Hadoop Distributed File System) adapter
- Nurego connector
- Pylon Publisher connector and adapter
- SMTP connector and adapter
- Sniffer connector and adapter
- Tibco Rendezvous connector and adapter
- Tervela connector and adapter
- Teradata connector and adapter
- HDAT reader adapter
- BoardReader publisher adapters
- SAS LASR Analytic Server adapter

Tervela transport is not an available for any remaining connector or adapter.

Table 1 *Connectors and Adapters*

Name	Description	Connector Available?	Adapter Available ?
Adapter	Functions as a wrapper around an adapter, enabling the ESP server to process the adapter as a connector.	Yes	No
Audio	Enables you to grab digitized audio from a locally installed sound card.	Yes	Yes
BACNet	Enables you to stream data over a communications protocol for Building Automation and Control (BAC) networks. The protocol is based on the ISO 16484-5 standard protocol.	Yes	Yes
BoardReader	Enables you to stream data from a feed aggregator. The site boardreader.com enables you to search message boards, websites, blogs, and other social media.	No	Yes

Name	Description	Connector Available?	Adapter Available ?
SAS Cloud Analytic Services	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
Cassandra	Enables you to stream data to and from Apache Cassandra databases.	No	Yes
Database	Supports publish and subscribe operations to a large number of databases.	Yes	Yes
Event Stream Processor	Enables you to subscribe to a window and publish what it passes into another Source window.	No	Yes
File and Socket	Supports 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	Enables 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	Enables 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

Name	Description	Connector Available?	Adapter Available ?
Modbus	Enables you to stream data to and from Modbus, a serial communications protocol commonly used to connect industrial electronic devices.	Yes	Yes
MQTT	Enables 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-DA	Enables you to publish through OPC Data Access (OPC-DA). OPC-DA is machine-to-machine protocol that is a precursor to OPC Unified Architecture (OPC-UA).	No	Yes
OPC-UA	Enables 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	Enables 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
PI Web	Enables you to use the PI Web API to retrieve and manipulate time series, asset, and event frame data.	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
Pylon	Enables you to communicate with a Basler GigE or USB camera to continuously publish captured frames into a Source window. For GigE cameras, there must be a known, fixed IP address, and the attached Ethernet network cable must provide power using Power-over-Ethernet.	Yes	Yes

Name	Description	Connector Available?	Adapter Available ?
QuasarDB	Enables you to stream data to and from QuasarDB, a distributed database for time series data that combines in-memory capabilities with long-term storage.	Yes	Yes
Rabbit MQ	Enables you to stream data to and from RabbitMQ, a lightweight open-source message broker.	Yes	Yes
REST	Supports subscribe operations to generate HTTP POST requests to a configured REST service.	No	Yes
SAS Data Set	Connects to a SAS Workspace Server in order to stream data to and from a SAS data set.	No	Yes
SMTP	Enables you to stream data through the Simple Mail Transfer Protocol, an internet standard for electronic mail (email) transmission.	Yes	Yes
Sniffer	Enables 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
Solace	Enables you to stream data to and from Solace, a message broker to establish event-driven interactions between applications and microservices.	Yes	Yes
Teradata	Enables you to stream data to and from a Teradata server.	Yes	Yes
Tervela	Enables 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
Tibco Rendezvous	Enables you to stream data to and from Tibco RV, a software product that provides a message bus for enterprise application integration (EAI).	Yes	Yes
Timer	Generates and publishes trigger events at regular intervals.	Yes	Yes
URL	Consumes data into a project through a URL-based connection.	Yes	No

Name	Description	Connector Available?	Adapter Available ?
UVC	Enables you to publish photos taken by a V4L2–compatible camera to an event stream processing model.	Yes	Yes
WebSocket	Enables you to read data over a WebSocket-based connection and publish the data into a project.	Yes	No
IBM WebSphere	Supports 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.

To include a connector when you do not have permissions to edit **esp-properties.yml**, use the `ESPENV` environment variable. Set the value of `connectors.excluded.`*`connector`* to false. For example, to include the Pylon connector, run the following on the command line:

```
export ESPENV='connectors.excluded.pylon=false'
```

For more information about the `ESPENV` environment variable, see "Configuring the ESP Server" in *SAS Event Stream Processing: Using the ESP Server*.

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: running, stopped, or finished.

- running — this is the default state of a connector when a project starts
- stopped — a connector is not running because, as a result of connector orchestration, it is waiting for another connector to finish
- finished — a connector has published all of its input events

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 online. Download the ZIP file and inspect the following subdirectories of the **xml** directory:

- 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 online. Download the ZIP file and inspect the subdirectories of the **cxx** directory.

The sample_connector subdirectory 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
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 amqp-client-3.4.2.jar. You must obtain amqp-client-3.4.2.jar from the Rabbit MQ Java client libraries at http://www.rabbitmq.com/java-

client.html . Install amqp-client-3.4.2.jar in **$DFESP_HOME/lib** on your system and rename it rabbitmq-client.jar.

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 -C*key1*=*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 single quotation marks (for example, *key =\'str,ing \'*). If a value string includes a space, then you must enclose the entire string in double quotation marks (for example, *key = "str ing"*). If a value string includes both a comma and a space, then enclose the string in single quotation marks first (without the escape character), followed by double quotation marks (for example, *key = "'st,r 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 sets it to FINISHED when the adapter is not running. When the connector stops, the associated running adapter ends.

IMPORTANT When you run an adapter through the Adapter connector, the executable file for the adapter must reside in **$DFESP_HOME** or **$DFESP_HOME/bin**.

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
`command`	Specifies the command and key-value pairs used to run the adapter from the command line. You must specify all required adapter parameters within the `command` parameter.
`type`	Specifies the mode of the adapter. The mode can be **pub** or **sub**, and must match the mode of the running adapter.

Table 6 *Optional Parameters for the Adapter Connector*

Parameter	Description
`url`	Specifies the URL for the adapter connection. If this parameter is not included, a default URL is generated.

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 -C
             node=@CASSANDRA_HOST@,
             port=@CASSANDRA_PORT@,keyspace=Sesptest,table=pubSimple,
             selectstatement="select * from Sesptest.pubSimple"</property>
            <property name='type'>pub</property>
            <property name='url'>dfESP://host:port/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.

Note: When you run a project on Microsoft Windows, you must specify the command with both the name and the extension. For example:

```
<property name='command'>dfesp_rest_subscriber.bat ...</property>
```

Using the Audio Connector and Adapter

Overview

Digital audio is a stream of discrete samples at a regular time interval that represents the amplitude of the original analog waveform. The time interval and the number of bits that represent the numerical values of the amplitude varies with different digital encoding formats.

The supported formats on a specific computing platform depend on the following factors:

- the installed sound card
- the sound source (usually a connected microphone)
- the software drivers that encode the incoming analog signal

IMPORTANT The Audio connector and adapter is not supported on custom Linux systems developed with the Yocto Project.

Using the Audio Connector

The audio connector grabs digitized audio from a locally installed sound card for publish operations to a Source window. The Source window schema must consist of two or three int64 fields, depending on whether the sound source is mono or stereo. The first field serves as an incrementing index value to be used as the key field. The second and possibly third field contain the amplitude values representing the digitized audio. The input event block size is fixed at 1 to ensure minimal latency.

The connector supports the Signed 16-bit Little Endian format, with a default sample rate of 16kHz and a single channel. You can configure a 44.1kHz/2-channel mode instead, in which case the Source window schema requires three fields instead of two. These are common formats that are supported by most microphones and sound cards.

The audio connector also requires installed ALSA drivers on the platform running the connector. (See https://www.alsa-project.org.) These drivers are typically included in the base OS distribution on most Linux and Microsoft Windows systems. They can be easily installed if necessary. You can use the ALSA command line utilities to discover supported formats on your platform.

For example, the following command shows the supported formats for a sound card installed as device 0 on card 1:

```
% arecord -D hw:1,0 --dump-hw-params
```

The only required configuration parameter for the connector is a string of the form "`hw:`*`card`*`,`*`device`*". Two optional parameters are supported:

- use 44.1kHz/2-channel encoding
- specify a filename where the connector writes the digital stream in WAV format

Table 7 *Required Parameters for the Audio Connector*

Parameter	Description
`type`	Specifies to publish. Must be set to `pub`.
`devicename`	Specifies the sound card device. Format is `hw:`*`card`*`,`*`device`*.

Table 8 *Optional Parameters for the Audio Connector*

Parameter	Description
`44.1kHzstereo`	Sets the audio format to a sample rate of 44.1kHz and 2-channel encoding. The default value is a sample rate of 16kHz and 1-channel encoding.
`wavfilename`	Specifies the name of a WAV file to write to the current directory.
`transactional`	Sets the event block type to transactional. The default event block type is normal.

Parameter	Description
`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 ]`.
`publishwithupsert`	Specifies to build events with `opcode = Upsert` instead of `opcode = Insert`.
`maxevents`	Specifies the maximum number of events to publish. Specify an integer value.
`blocksize`	Specifies the number of events to include in a published event block. Specify an integer value. The default value is 1.

Using the Audio Adapter

The audio adapter grabs digitized audio from a local sound card for publish operations to an ESP source window.

`dfesp_audio_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. (Here is an example: `key=\"str,ing\"`.)

Table 9 *Required Keys*

Key-Value Pair	Description
`url=pubsubURL`	Specifies the standard URL in the form **`dfESP://host:port/project/continuousquery/window`**.
`devicename=name`	Specifies the sound card device, in the form "`hw:card,device`".

Table 10 *Optional Keys*

Key-Value Pair	Description
`gdconfig=file`	Specifies the guaranteed delivery configuration file.

Key-Value Pair	Description
`transport=native | solace | rabbitmq | kafka`	Specifies the transport type. If you specify Solace, RabbitMQ, or Kafka transports instead of the default native transport, then use the required client configuration files specified in the description of the C++ `C_dfESPpubsubSetPubsubLib()` API call.
`loglevel=trace | debug | info | warn | error | fatal | off`	Sets the logging level for the adapter. This is the same range of logging levels that you can set in the `C_dfESPpubsubInit()` publish/subscribe API call and in the engine `initialize()` call. The default level is **warn**.
`logconfigfile=`*file*	Specifies the log configuration file.
`tokenlocation=`*location*	Specifies the location of the file in the local file system that contains the OAuth token required for authentication by the publish/subscribe server.
`configfilesection=[`*section*`]`	Specifies the name of the section in **/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config** (Linux) or **%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.config** (Windows) to parse for configuration parameters. Specify the value as `[`*configfilesection*` ]`.
`restartonerror=true | false`	When **true**, specifies to restart the adapter if a fatal error is reported.
`transportconfigfile=`*file*	Specifies the publish/subscribe transport configuration file. For Solace, the default is **./solace.cfg**. For RabbitMQ, the default is **./rabbitmq.cfg**. For Kafka, the default is **./kafka.cfg**. **Note:** No transport configuration file is required for native transport.
`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.
`wavfilename=file`	Specifies the name of a WAV file to write to the current directory.

Key-Value Pair	Description
`44.1kHzstereo=true | false`	When **`true`**, sets the audio format to a sample rate of 44.1kHz and 2-channel encoding. The default value is a sample rate of 16kHz and 1-channel encoding.
`blocksize=`*`evntblcksize`*	Sets the event block size. Specify an integer value for *evntblcksize*. The default value is 1.

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.

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 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>
      },
      …
    ]
  },
  …
]
```

- Instead of `"PropertyIdentifier"`, you can configure `"PropertyIdentifierString"` in order to identify the property by name.

- You can configure `"ArrayIndex"` when the property is an array and you want only the value at the specified index.
- You can configure `"AnchorTime"` as an absolute timestamp using format `"%Y-%m-%d %H:%M:%S"`. It specifies the time when the initial read of the property occurs, and can be in the past or the present. Use it to stagger reads and thus reduce usage spikes on the BACnet network.

  If the value of `"AnchorTime"` is in the past, the first read of the property occurs at the next future timestamp that matches the `"AnchorTime"` incremented by an integer number of `"Period"` intervals. If the value is in the future, the first read of the property happens at the configured time. Further reads occur at intervals defined by the value of `"Period"`.

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.

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 11 *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 12 *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 13 *Optional Parameters for the Publisher BACnet Connector*

Parameter	Description
`bacnetipport`	Specifies the local port used by the connector. The default port number is `47808` .
`blocksize`	Specifies the number of events to include in a published event block. The default value is `1`.
`configfilesection`	Specifies the name of the section in **`/opt/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`*` ]`.
`ignorebacneterrors`	Specifies that when a read property request fails, log a warning and continue. The default action is to log an error and stop.
`publishwithupsert`	Builds events with `opcode=Upsert` instead of `Insert`.
`maxevents`	Specifies the maximum number of events to publish.
`transactional`	Sets the event block type to transactional. The default event block type is normal.

Table 14 *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

BACnet type	SAS Event Stream Processing data types
	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* ,...

Table 15 *Required Keys*

Key-Value Pair	Description
`url=` *pubsubURL*	Specifies the standard URL in the form **`dfESP://`** ***`host :port/ project/continuousquery /window`***.
`bacnetbbmdaddress=` *BBMDipaddr*	Specifies the IP address of the BBMD.
`bacnetbbmdport=` *BBMDport*	Specifies the local port used by the adapter. The default port number is `47808` .

Key-Value Pair	Description
`bacnetconfigfile= jsonfile`	Specifies the JSON configuration file containing BACnet device and object information.

Table 16 *Optional Keys*

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

Key-Value Pair	Description
`bacnetipport= localport`	Specifies the local port used by the adapter. The default port number is `47808` .
`transportconfigfile= file`	Specifies the transport configuration file. The default value depends on the transport type specified. For `solace` , the default is **`./solace.cfg`**. For `tervela` , the default is **`./client.config`**. For `rabbitmq` , the default is **`./rabbitmq.cfg`**. For `kafka` , the default is **`./kafka.cfg`**. **Note:** No transport configuration file is required for native transport.
`ignorebacneterrors=true | false`	Specifies that when a read property request fails, log a warning and continue. The default action is to log an error and stop.
`maxevents= maxevents`	Specifies the maximum number of events to publish.
`quiesceproject=true | false`	When *maxevents* is configured, quiesces the project after all events are injected into the Source window.

For information about implementing hot failover for publisher adapters, see "Publisher Adapter Failover with Kafka" in *SAS Event Stream Processing: Implementing Failover*.

Using the BoardReader Publisher Adapters

IMPORTANT This functionality is no longer supported or included in SAS Viya 3.5 revision 24w44. No updates to this functionality are available for earlier releases.

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

The BoardReader adapters provided by 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 Reference* . The corresponding Source window must contain fields that correspond to every received JSON key unless you specify the adapter parameter `esp-ignoremissingschemafields`. Events are built with the Insert opcode unless you configure the adapter to use Upsert instead. If needed, a JSON filtering function can be enabled using the `esp-matchsubstrings` parameter.

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

```
eventindex*:int64
adapterindex*:string
```

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

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

- `ProjectID:string`
- `Query:string`

The contents of these fields mirrors the values in the `customer_project_id` and `customer_query_id` configuration parameters, for correlation purposes.

Usage:

dfesp_*_boardreader -config-txt *configtxt* -esp-url *url* <-dev-mode-history-to-disk > <-dev-mode-resp-to-disk > <-email-from *emailfrom* > <-email-level *emaillevel* > <-email-smtphost *emailsmtphost* <-email-subject *emailsubject* >> <-email-to *emailto* > <-esp-dateformat *dateformat* > <-esp-logconfigfile *logconfigfile* > <-esp-loglevel *loglevel* > <-esp-ignoremissingschemafields > <-esp-matchsubstrings *substrings* > <-esp-publishwithupsert >

Table 17 *Parameters for the BoardReader Publisher Adapters*

Parameter	Description
`-config-txt` *`configtxt`*	Specifies the name of a text file in the configuration directory that contains BoardReader content source configuration parameters. See the sample files in your configuration directory .
`-dev-mode-history-to-disk`	Provides additional debugging: writes SQLite database formatted documents to disk in your configuration directory unless you have configured the `work_dir` parameter.
`-dev-mode-resp-to-disk`	Provides additional debugging: writes responses to disk in your configuration directory unless you have configured the `work_dir` parameter.
`-email-from` *`emailfrom`*	Specifies the source address for email logging messages.
`-email-level` *`emaillevel`*	Specifies the email logging level. Permitted values are DEBUG, INFO, WARNING, ERROR, and CRITICAL. The default value is CRITICAL.
`-email-smtphost` *`emailsmtphost`*	Specifies the SMTP host for email logging messages.
`-email-subject` *`emailsubject`*	Specifies the subject line for email logging messages.
`-email-to` *`emailto`*	Specifies the destination address for email logging messages.
`-esp-dateformat` *`dateformat`*	Specifies the format of `DATE` and `STAMP` fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (`DATE`) or microseconds (`STAMP`) since epoch. The `dateformat` parameter accepts any time format that is supported by the UNIX `strftime` function.
`-esp-ignoremissingschemafields`	Specifies to ignore and continue when the JSON key is missing in schema. By default, an error is

Parameter	Description
	reported and the process quits when the key is missing.
`-esp-logconfigfile` *logconfigfile*	Specifies the name of the log configuration file.
`-esp-loglevel` *loglevel*	Specifies the logging level. Permitted values are TRACE, DEBUG, INFO, WARN, ERROR, or FATAL. The default value is INFO.
`-esp-matchsubstrings` *substrings*	Specifies a list of comma separated `fieldname:value` pairs, where input events that do not contain a `value` substring in the `fieldname` string field are dropped. The comparison is case-insensitive.
`-esp-publishwithupsert`	Specifies to publish events with `opcode = Upsert`. By default, events are published with `opcode = Insert`.
`-esp-url` *url*	Specifies the publisher standard URL in the form "`dfESP://`*host* `:`*port*`/` *project*`/` *continuousquery* `/`*window*"

For information about implementing hot failover for publisher adapters, see "Publisher Adapter Failover with Kafka" 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. If a SASHDAT file of the same name as the configured table exists in the SAS Cloud Analytic Services server's file system, then the adapter loads it into the CAS table before appending events to it.

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*.

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 'password ' | openssl enc -e -aes-256-cbc -md md5\
  -a -salt -pass pass:'espCASadapterUsedByUser=username'
```

Specify the encrypted *password* suppplied 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*.

IMPORTANT The default values of Boolean parameters for this adapter are **false**. You cannot manually reset a Boolean parameter from **true** to **false** with `-C`. To reset the value, simply remove the parameter from the command line.

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 18 *Required Keys*

Key-Value Pairs	Description
`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><? rmretdel=true | false>". **Note:** Multiple URLs, separated by commas, are permitted.
`cashostport= url`	Specifies the CAS server host name and port in the form *"host:port "*
`type=sub`	Specifies a subscriber adapter.
`castable= table`	Specifies the full name of the CAS table.

Table 19 *Optional Keys*

Key-Value Pairs	Description
`commitblocks= number`	Specifies the number of event blocks to commit to the SAS Cloud Analytic Services table at one time. The default value is 8. This parameter trades throughput for latency. Increase the value to increase the number of event blocks appended to the SAS Cloud Analytic Services table before being committed.
`buffersize= size`	Specifies the buffer size. The default value is 512. This buffering parameter specifies the number of table rows that the adapter buffers before sending them to the SAS Cloud Analytic Services.
`configfilesection= section`	Specifies the name of the section in **`/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/javaadapters.config`** (Linux) or **`%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\javaadapters.config`** (Windows) to parse for configuration parameters.
`pwdencrypted= true | false`	When true, the CAS password is encrypted. The default value is false.
`gdconfigfile= file`	Specifies the guaranteed delivery configuration file.

Key-Value Pairs	Description
`cassessionid=` *id*	Specifies the UUID of the session to which to connect. If not specified, the SAS Cloud Analytic Services adapter creates a new SAS Cloud Analytic Services session.
`pubsublib= native | solace | tervela | rabbitmq | kafka`	Specifies the transport type. When you specify the `solace`, `tervela`, `rabbitmq`, or `kafka` transports instead of the default `native` transport, use the required client configuration files specified at "Using the Java Publish/Subscribe API" in *SAS Event Stream Processing: Publish/Subscribe API Reference* .
`loglevel=severe | warning | info`	Specifies the application logging level. The default level is `warning` .
`caslib=` *libname*	Specifies the SAS Cloud Analytic Services library containing the SAS Cloud Analytic Services table. The default is the currently active SAS Cloud Analytic Services library.
`casusername=` *username*	Specifies the user name used to authenticate the SAS Cloud Analytic Services session. Allowed only when *casoauthtoken* is not specified.
`tokenlocation=` *location*	Specifies the location of the file in the local file system that contains the OAuth token required for authentication by the publish/subscribe server.
`casoauthtoken=` *token*	Specifies the OAuth token that is used to authenticate the SAS Cloud Analytic Services session. Allowed only when *casusername* is not specified.
`caspassword=` *password*	Specifies the password used to authenticate the SAS Cloud Analytic Services session. When *caspassword* is not specified and *casusername* is specified, the password is extracted from that user's **`authoinfo/netrc`** file.
`columns=` *subset*	Specifies a subset of columns to read or write to and from the SAS Cloud Analytic Services table. Use the form *c1_name,...cn_name* . **Note:** When the column names in the SAS Cloud Analytic Services table match all the field names in the 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).

Key-Value Pairs	Description
`restartonerror= true | false`	When true, restarts the adapter when a fatal error is reported. The default value is false. **`javaadapters.config`** parameter name: `restartonerror`
`transportconfigfile= file`	Specifies the full path to the publish/subscribe transport configuration file. The default file depends on the transport type specified with through the `pubsublib` parameter: For `solace` , the default file is **`solace.cfg`**. For `tervela` , the default file is **`client.config`**. For `rabbitmq` , the default file is **`rabbitmq.cfg`**. For `kafka` , the default file is **`kafka.cfg`**. **Note:** No transport configuration file is required for native transport.
`maxnumrows= number`	Specifies the maximum number of table rows per addTable action. By default, the action is kept open indefinitely.
`periodicity= number`	Specifies the maximum number of seconds to keep the addTable action open. By default the action is kept open indefinitely.

Publisher Usage

dfesp_cas_adapter -C *key1 =val1*, *key2 =val2*, *key3 =val3* , ...

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

Table 20 *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* ".

Key-Value Pair	Description
`cashostport= url`	Specifies the CAS server host name and port in the form *"host:port "*
`type=pub`	Specifies a publisher adapter.
`castable= table`	Specifies the full name of the CAS table.

Table 21 *Optional Keys*

Key-Value Pair	Description
`blocksize= size`	Specifies the number of events to be flushed to the server. The default value is 1.
`configfilesection= section`	Specifies the name of the section in **`/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/javaadapters.config`** (Linux) or **`%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\javaadapters.config`** (Windows) to parse for configuration parameters.
`quiesceproject=true | false`	When true, specifies to quiesce the project after all events are injected into the Source window. **`javaadapters.config`** parameter name: `quiesceproject`
`pwdencrypted= true | false`	When true, the CAS password is encrypted. The default value is false..
`transactional= true | false`	When true, event blocks are transactional. By default, they are normal.
`gdconfigfile= file`	Specifies the guaranteed delivery configuration file.
`cassessionid= id`	Specifies the UUID of the session to which to connect. If not specified, the SAS Cloud Analytic Services adapter creates a new SAS Cloud Analytic Services session.
`pubsublib= native | solace | tervela | rabbitmq | kafka`	Specifies the transport type. When you specify the `solace`, `tervela`, `rabbitmq`, or `kafka` transports instead of the default `native` transport, use the required client configuration files specified at "Using the Java Publish/Subscribe API" in *SAS Event Stream Processing: Publish/Subscribe API Reference* .
`loglevel=severe | warning | info`	Specifies the application logging level. The default level is `warning` .

Key-Value Pair	Description
`caslib= libname`	Specifies the SAS Cloud Analytic Services library containing the SAS Cloud Analytic Services table. The default is the currently active SAS Cloud Analytic Services library.
`casusername= username`	Specifies the user name used to authenticate the SAS Cloud Analytic Services session. Allowed only when *casoauthtoken* is not specified.
`tokenlocation= location`	Specifies the location of the file in the local file system that contains the OAuth token required for authentication by the publish/subscribe server.
`casoauthtoken= token`	Specifies the OAuth token that is used to authenticate the SAS Cloud Analytic Services session. Allowed only when *casusername* is not specified.
`caspassword= password`	Specifies the password used to authenticate the SAS Cloud Analytic Services session. When *caspassword* is not specified and *casusername* is specified, the password is extracted from that user's **`authoinfo/netrc`** file.
`columns= subset`	Specifies a subset of columns to read or write to and from the SAS Cloud Analytic Services table. Use the form *c1_name,...cn_name* . **Note:** When the column names in the SAS Cloud Analytic Services table match all the field names in the 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).
`publishwithupsert=true | false`	When true, builds events with `opcode=Upsert`. The default value is false. By default, events are built with `opcode=Insert`. **`javaadapters.config`** parameter name: `publishwithupsert`
`restartonerror= true | false`	When true, restarts the adapter when a fatal error is reported. The default value is false. **`javaadapters.config`** parameter name: `restartonerror`
`maxevents= number`	Specifies the maximum number of events to publish.
`transportconfigfile= file`	Specifies the full path to the publish/subscribe transport configuration file. The default file depends on the

Key-Value Pair	Description
	transport type specified with through the `pubsublib` parameter: For `solace` , the default file is **`solace.cfg`**. For `tervela` , the default file is **`client.config`**. For `rabbitmq` , the default file is **`rabbitmq.cfg`**. For `kafka` , the default file is **`kafka.cfg`**. **Note:** No transport configuration file is required for native transport.

Note: When the SAS Cloud Analytic Services table contains fewer columns than the 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 Steam Processing data type mappings are as follows:

SAS Cloud Analytic Services type	SAS Event Stream Processing data type
INT32	INT32
INT64	INT64
DOUBLE	DOUBLE
DATE	DATETIME
CHAR	UTF8STR or RUTF8STR
VARCHAR	UTF8STR or RUTF8STR
DECSEXT	MONEY
TIME	INT64
DATETIME	TIMESTAMP

SAS Cloud Analytic Services type	SAS Event Stream Processing data type
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 drive, which supports Cassandra version 1.2 through 3.0 and later. It negotiates the version of native protocol to use during a connection. Other versions of the driver might work with the adapter, but they are not tested.

The client target platform must define the environment variable DFESP_DATASTAX_JARS to specify the location of the DataStax Java driver JAR files and other required JAR files. The additional JAR files that you need depend on the version of the Java driver that you download.

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

IMPORTANT The default values of Boolean parameters for this adapter are **`false`**. You cannot manually reset a Boolean parameter from **`true`** to **`false`** with `-C`. To reset the value, simply remove the parameter from the command line.

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 22 *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 23 *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"` .

Key-Value Pair	Description
`batchsize= number`	Specifies the total number of statements per Insert, Update, and Delete set of batches. The default is the event block size.
`configfilesection= section`	Specifies the name of the section in **`/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/javaadapters.config`** (Linux) or **`%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\javaadapters.config`** (Windows) to parse for configuration parameters.
`gdconfigfile= file`	Specifies the Guaranteed Delivery configuration file.
`pubsublib= native | solace | tervela | rabbitmq | kafka`	Specifies the transport type. When you specify the `solace`, `tervela`, `rabbitmq`, or `kafka` transports instead of the default `native` transport, use the required client configuration files specified at "Using the Java Publish/Subscribe API" in *SAS Event Stream Processing: Publish/Subscribe API Reference* .
`compression= type`	Specifies Cassandra transport compression, valid values are `"NONE"` , `"LZ4"`, `"SNAPPY"` , default is `"NONE"`.
`username= name`	Specifies the user name to log on to the Cassandra host.
`loglevel=severe | warning | info`	Specifies the application logging level. The default value is `warning` .
`tokenlocation= location`	Specifies the location of the file in the local file system that contains the OAuth token required for authentication by the publish/subscribe server.
`password= password`	Specifies the password to use with the specified user name.
`ssl=true | false`	When true, enables Transport Layer Security (TLS) on the Cassandra connection, using the default platform JSSE options. The default value is false.
`batchsecs= seconds`	Specifies the maximum number of seconds to hold an incomplete batch. The default is 0.
`restartonerror=true | false`	When true, restarts the adapter if a fatal error is reported. The default value is false.
`port= port`	Specifies the Cassandra cluster node port. The default value is 9042.

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

Publisher Usage

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

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

Table 24 *Required Keys*

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

Key-Value Pair	Description
	"dfESP:// *host*:*port* /*project*/ *contquery*/*window* ".

Table 25 *Optional Keys*

Key-Value Pair	Description
`loadbalancingpolicy=` *policy*	Specifies a Cassandra load balancing policy, with optional comma-separated wrapped policies. The default value is `"TokenAware,DCAwareRoundRobin"` .
`blocksize=` *number*	Specifies the number of events per event block. The default value is 1.
`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.
`gdconfigfile=` *file*	Specifies the guaranteed delivery configuration file.
`pubsublib= native | solace | tervela | rabbitmq | kafka`	Specifies the transport type. When you specify the `solace`, `tervela`, `rabbitmq`, or `kafka` transports instead of the default `native` transport, use the required client configuration files specified at "Using the Java Publish/Subscribe API" in *SAS Event Stream Processing: Publish/Subscribe API Reference* .
`compression=` *value*	Specifies Cassandra transport compression, valid values are `"NONE"` , `"LZ4"`, `"SNAPPY"` , default is `"NONE"`.
`username=` *name*	Specifies the user name to log on to the Cassandra host.
`loglevel=severe | warning | info`	Specifies the application logging level. The default value is `warning` .
`tokenlocation=` *location*	Specifies the location of the file in the local file system that contains the OAuth token that is required for authentication by the publish/subscribe server.
`password=` *password*	Specifies the password to use with the specified user name.
`quiesceproject=true|false`	When true, quiesces the project after all events are injected into the Source window. The default value is false.

Key-Value Pair	Description
`transactional=true|false`	When true, specifies that event blocks are transactional. The default value is false, and the event block type is normal.
`ssl=true|false`	When true, enables TLS on the Cassandra connection, using the default platform JSSE options. The default value is false.
`publishwithupsert=true|false`	When true, builds events with `opcode=Upsert` instead of `Insert` . The default value is false.
`restartonerror=true|false`	When true, restarts the adapter if a fatal error is reported. The default value is false.
`maxevents=` *number*	Specifies the maximum number of events to publish.
`port=` *number*	Specifies the Cassandra cluster node port. The default value is 9042.
`retrypolicy=` *policy*	Specifies a Cassandra retry policy, with optional comma-separated wrapped policies, default is "`Default`".
`transportconfigfile=` *file*	Specifies the full path to the publish/subscribe transport configuration file. The default file depends on the transport type specified with through the `pubsublib` parameter: For `solace` , the default file is **`solace.cfg`**. For `tervela` , the default file is **`client.config`**. For `rabbitmq` , the default file is **`rabbitmq.cfg`**. For `kafka` , the default file is **`kafka.cfg`**. **Note:** No transport configuration file is required for native transport.

Publisher Failover

For information about implementing hot failover for publisher adapters, see "Publisher Adapter Failover with Kafka" in *SAS Event Stream Processing: Implementing Failover*.

Using the Database Connector and Adapter

Using DataDirect ODBC Drivers

Progress DataDirect Connect Series for ODBC provides ODBC drivers for a number of databases. These drivers are built into SAS Event Stream Processing in order to facilitate connectivity through the Database connector and adapter.

When using an ODBC driver, you establish connectivity to a database through a system data source name (DSN). A DSN and the associated database user credentials are required configuration parameters. The general format of a DSN is as follows:

```
DSN=data_source[;attribute=value[;attribute=value]...]
```

You must specify a *data_source*, which names the desired wire protocol to use for database connectivity. For the data source that you specify, you enumerate one or more relevant attributes. Most wire protocols require that you specify a *userid* and *password* in order to access the database. Many wire protocols also require that you specify a *hostname* that identifies the location of the database.

Each attribute that you specify for the *data_source* overrides the value set in the odbc.ini file that is supplied with SAS Event Stream Processing.

For the Database connector and adapter, supply a DSN in the required `connectstring` parameter. The publisher obtains result sets from the database by using a single SQL statement that is configured by the user. Additional result sets can be obtained by stopping and restarting the connector. The subscriber writes window output events to the database table that is configured in the required `desttablename` parameter.

The following table lists the databases that are supported by the Database connector and adapter with their corresponding data sources. A link is provided to the complete list of attributes that you can configure for each data source.

Table 26 *Supported Databases*

Database	Data Source	Currently Supported Version
Amazon Redshift	Amazon Redshift Wire Protocol	08.00.2233
IBM Db2	Db2 Wire Protocol	08.02.0455
Greenplum	Greenplum Wire Protocol	07.16.1136

Database	Data Source	Currently Supported Version
Informix	Informix Wire Protocol	08.02.0124
MySQL	MySQL Wire Protocol	08.02.0466
Oracle	Oracle Wire Protocol	08.02.3072
PostgreSQL	PostgreSQL Wire Protocol	08.02.2397
Microsoft SQL Server	SQL Server Wire Protocol	08.02.1428
SAP Sybase ASE	Sybase Wire Protcol	07.16.0373
Sybase IQ	Sybase IQ Wire Protocol	08.02.0025
Teradata	Teradata	07.16.0165

IMPORTANT The following data sources are listed in the **odbc.ini** file that is supplied with the product. However, they are not supported. Use them at your own discretion.

- Apache Hive Wire Protocol
- Impala Wire Protocol
- OpenEdge Wire Protocol
- Salesforce

Table 27 *Minimal connectstring Value Required for Each Database*

Database	Minimal connectstring Value
Amazon Redshift	`DSN=Amazon Redshift Wire Protocol;Database=DB_name;HOST=Amazon_Redshift_cluster_IP_address;UID=username;PWD=password`
IBM Db2	`DSN=DB2 Wire Protocol;uid=username;pwd=password;ipAddress=DB_server_IP_address;database=DB_name;tcpPort=port`
Greenplum	`DSN=Greenplum Wire Protocol;uid=username;pwd=password;database=DB_name;hostName=DB_server_hostname`
Informix	`DSN=Informix Wire Protocol;uid=username;pwd=password;HostNam`

Database	Minimal connectstring Value
	e=*DB_server_hostname*;PortNumber=*port*;ServerName=*Informix_DB_server_name*;Database=*DB_name*
MySQL	DSN=MySQL Wire Protocol;uid=*username*;pwd=*password*;hostname=*DB_server_hostname*;database=*DB_name*
Oracle	DSN=Oracle Wire Protocol;uid=*username*;pwd=*password*;hostname=*DB_server_hostname*;servicename=*DB_server_process_name*
PostgreSQL	DSN=PostgreSQL Wire Protocol;uid=*username*;pwd=*password*;hostname=*DB_server_hostname*
Microsoft SQL Server	DSN=SQL Server Wire Protocol;uid=*username*;pwd=*password*;hostName=*DB_server_hostname*
SAP Sybase ASE	DSN=Sybase Wire Protocol;uid=*username*;pwd=*password*;database=*DB_name*;networkAddress=*DB_server_IP_address*,*port*
Sybase IQ	DSN=Sybase IQ Wire Protocol;uid=*username*;pwd=*password*;Database=*DB_name*;NetworkAddress=*DB_Server_IP_address*,*port*
Teradata	DSN=Teradata;uid=*username*;pwd=*password*;DBCName=*DB_Server_IP_address* | *DB_server_FQN*;database=*DB_name*

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 by using OpenSSL, which must be installed on your system. When you installed the SAS Event Stream Processing System Encryption and Authentication Overlay, you installed 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 -md md5 -a -salt\
 -pass pass:'espDBconnectorUsedByUser=username'
```

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

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

IMPORTANT When you migrate from any earlier release of SAS Event Stream Processing to release 6.2, do the following:

- Stop using a custom **odbc.ini** file. A working **odbc.ini** file is delivered with the product.
- Do not set the environment variable ODBCINI.
- Put all custom driver attributes in the DSN that you set through the connectstring.
- Keep in mind that DataDirect ODBC drivers have moved from **$DFESP_HOME/lib** to **$DFESP_HOME/odbc/lib**. This does not affect driver usage, as you now use the working **odbc.ini** file that is delivered with the product.

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.

Note: To use these connectors, you must set the SAS_LICENSE environment variable to **=${DFESP_HOME}/etc/license/license.txt**. The license file must contain SAS/ACCESS to ODBC.

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*.

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.

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

Table 28 *Required Parameters for Subscriber Database Connectors*

Parameter	Description
connectstring	When using DataDirect drivers, specifies the database DSN and user credentials. When using SAS/TKTS drivers, specifies the driver type and other driver specific parameters. **Note:** If you want to include uid and pwd parameters in the **odbc.ini** file rather than in the connectstring, then consider the parameters that you have available for the driver that you are using. For example, some drivers provide an option to use LogonID and Password in **odbc.ini** rather than uid and pwd in the connectstring.
desttablename	Specifies the target table name.
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.
type	Specifies to subscribe. Must be "sub".

Table 29 *Required Parameters for Publisher Database Connectors*

Parameter	Description
connectstring	When using DataDirect drivers, specifies the database DSN and user credentials. Use the following format: DSN=*dsn*[;uid=*userid*][;pwd=*password*]. When using SAS/TKTS drivers, specifies the driver type and other driver specific parameters. **Note:** If you want to include uid and pwd parameters in the **odbc.ini** file rather than in the connectstring, then consider the parameters that you have available for the driver that you are using. For example, some drivers use the parameters LogonID and Password in **odbc.ini** rather than uid and pwd in the connectstring.
type	Specifies to publish. Must be "pub".

Table 30 *Optional Parameters for Subscriber Database Connectors*

Parameter	Description
`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 ]`.
`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 `pwd` field in `connectstring` is encrypted.
`rmretdel`	Specifies to remove all delete events from event blocks received by a subscriber that were introduced by a window retention policy.
`noquotedcolumnnames`	Specifies to not surround column names in double quotes in SQL updates and deletes. The default value is **`false`**.
`sas`	When **`true`** , specifies to use the SAS threaded kernel drivers instead of the DataDirect drivers. The default value is **`false`**.

Table 31 *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 **`/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 ]`.

Parameter	Description
greenplumlogminer	Enables Greenplum log miner mode. "Using Log Miner Modes".
logminerdbname	Specifies the gpperfmon database that contains the queries_history table for Greenplum log miner mode. Use the following format: *dd-mmm-yyy hh:mm:ss*.
logminerschemaowner	Specifies the schema owner when using Oracle or Greenplum log miner mode. For more information, see "Using Log Miner Modes".
logminerstartdatetime	Specifies the start date time when using Oracle or Greenplum log miner mode. For more information, see "Using Log Miner Modes".
logminertablename	Specifies the table name when using Oracle or Greenplum log miner mode. For more information, see "Using Log Miner Modes".
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. "Using Log Miner Modes".
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 **true** specifies to the SAS threaded kernel drivers instead of the DataDirect drivers . The default value is **false**.
repeatcount	Specifies the number of times to repeat the publisher. The default value is 0 unless you have configured repeatinterval, in which case the publisher is repeated indefinitely.
repeatinterval	Specifies the number of seconds between publisher repetitions. The default value is 0.

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_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)
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
ESP_BINARY	SQL_BINARY, SQL_VARBINARY, SQL_LONGVARBINARY

Note: The DataDirect PostgreSQL driver converts the TIMESTAMPTZ data type to a string. To support this data type when you access a PostgreSQL table, map it to either the ESP_UTF8STR or ESP_RUTF8STR data type.

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

Publisher SQL Data Type	Event Stream Processor Data Types
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 <= 10), ESP_INT64 (only if scale = 0 and precision <= 16), ESP_DOUBLE (only if precision <= 16), ESP_MONEY
SQL_NUMERIC	ESP_INT32 (only if scale = 0 and precision <= 10), ESP_INT64 (only if scale = 0 and precision <= 16), ESP_DOUBLE (only if precision <= 16), ESP_MONEY
SQL_BINARY	ESP_BINARY
SQL_VARBINARY	ESP_BINARY
SQL_LONGVARBINARY	ESP_BINARY

Note: The DataDirect PostgreSQL driver converts the TIMESTAMPTZ data type to a string. To support this data type when you access a PostgreSQL table, map it to either the ESP_UTF8STR or ESP_RUTF8STR data type.

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 .

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, ROW_ID 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,sql_rowid:string,sche
ma:string,tablename: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. 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 ,...*

Subscriber Usage

Table 32 *Required Keys*

Key-Value Pairs	Description
`type=sub`	Specifies a subscriber adapter.
`url=` *pubsubURL*	Specifies the standard URL in the form **`dfESP://`** ***`host :port/ project/continuousquery /window`***. Append `?snapshot=true |false` for subscribers.When `?snapshot=true`, the subscriber receives a collection of Insert events that are contained in the window at that point in time. The subscriber then receives a stream of events produced from the time of the snapshot onward. Those subsequent events can be Inserts, Updates, or Deletes. Append `?rmretdel=true | false` for subscribers if needed.
`connectstring=` *string*	When using the DataDirect drivers, *string* specifies the database DSN and user credentials. Use the following format: `DSN=`*dsn*`[;uid=`*userid* `][;pwd=`*password* `]`. When using the SAS threaded kernel drivers, *string* specifies the driver type and other driver-specific parameters. **Note:** If you want to include `uid` and `pwd` parameters in the **`odbc.ini`** file rather than in the `connectstring`, then consider the parameters that you have available for the driver that you are using. For example, some drivers provide an option to use `LogonID` and `Password` in **`odbc.ini`** rather than `uid` and `pwd` in the `connectstring`.
`desttablename=` *table*	Specifies the subscriber target database table.

Table 33 *Optional Keys*

Key-Value Pair	Description
`gdconfig=` *file*	Specifies the guaranteed delivery configuration file.
`transport=native | solace | tervela | rabbitmq | kafka`	Specifies the transport type. If you specify `solace`, `tervela` , `rabbitmq`, or `kafka` transports instead of the default `native` transport, then use the required client configuration files specified in the description of the C++C_dfESPpubsubSetPubsubLib() API call.
`loglevel=trace | debug | info | warn | error | fatal | off`	Sets the logging level for the adapter. This is the same range of logging levels that you can set in the `C_dfESPpubsubInit()` publish/subscribe API call and in the engine `initialize()` call. The default level is `warn`.
`logconfigfile=` *logconfigfile*	Specifies the log configuration file.
`tokenlocation=` *location*	Specifies the location of the file in the local filesystem that contains the OAuth token required for authentication by the publish/subscribe server.
`maxcolbinding=` *maxwidth*	Specifies the maximum supported width of string columns. The default value is 4096.
`configfilesection=[` *section*`]`	Specifies the name of the section in **`/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config`** (Linux) or **`%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.config`** (Windows) to parse for configuration parameters. Specify the value as `[`*configfilesection*`]`.
`pwdencrypted=true | false`	When `true` , the password in `connectstring` is encrypted.
`restartonerror= true | false`	When `true` , restarts the adapter if a fatal error is reported.
`transportconfigfile=` *file*	Specifies the publish/subscribe transport configuration file. For `solace` , the default is **`./solace.cfg`**. For `tervela` , the default is **`./client.config`**. For `rabbitmq` , the default is **`./rabbitmq.cfg`**. For `kafka` , the default is **`./kafka.cfg`**. **Note:** No transport configuration file is required for native transport.

Key-Value Pair	Description
`commitrows= minrows`	Specifies the minimum number of output rows to buffer.
`commitsecs= maxsecs`	Specifies the maximum number of seconds to hold onto an incomplete commit buffer.
`ignoresqlerrors=true | false`	When **`true`** , ignores errors when executing SQL Inserts, Updates, or Deletes.
`SAS=true | false`	When **`true`** , specifies to use the SAS threaded kernel drivers instead of the DataDirect drivers. The default value is **`false`**.
`noquotedcolumnnames=true | false`	Specifies to not surround column names in double quotes in SQL updates and deletes. The default value is **`false`**.

Publisher Usage

Table 34 *Required Keys*

Key-Value Pairs	Description
`type=pub`	Specifies a publisher.
`url= pubsubURL`	Specifies the standard URL in the form **`dfESP://`** ***`host :port/ project/continuousquery /window`***.
`connectstring= string`	When using the DataDirect drivers, *string* specifies the database DSN and user credentials. Use the following format: `DSN=dsn[;uid=userid ][;pwd=password ].` When using the SAS threaded kernel drivers, *string* specifies the driver type and other driver-specific parameters. **Note:** If you want to include `uid` and `pwd` parameters in the **`odbc.ini`** file rather than in the `connectstring`, then consider the parameters that you have available for the driver that you are using. For example, some drivers provide an option to use `LogonID` and `Password` in **`odbc.ini`** rather than `uid` and `pwd` in the `connectstring`.

Table 35 *Optional Keys*

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

Key-Value Pairs	Description
`selectstatement= statement`	Specifies the publisher source database SQL statement.
`blocksize= size`	Specifies the event block size. The default value is 1.
`transactional=true | false`	When `true` , event blocks are transactional. By default, event blocks are normal.
`quiesceproject=true | false`	When `true` , quiesces the project after all events are injected into the Source window.
`logminerschemaowner= owner`	Specifies the schema owner for Oracle or Greenplum log miner mode.
`logminerdbname= database`	Specifies the `gpperfmon` database that contains the `queries_history` table.
`logminertablename= tablename`	Specifies the table name for Oracle or Greenplum log miner mode.
`logminerstartdatetime= datetime`	Specifies the start date time for Oracle or Greenplum log miner mode. Use the following format: `"dd-mmm-yyyy hh:mm:ss "`
`oraclelogminer=true | false`	When `true` , enables Oracle log miner mode.
`greenplumlogminer=true | false`	When `true` , enables Greenplum log miner mode.
`publishwithupsert=true | false`	When `true` , builds events with `opcode=Upsert` instead of `Insert`.
`maxevents= maxevents`	`Specifies the maximum number of events to publish.`
`SAS=true | false`	When **`true`** , specifies to use the SAS threaded kernel drivers instead of the the DataDirect drivers. The default value is **`false`** .
`repeatcount=value`	Specifies the number of times to repeat the publisher. The default value is 0 unless you have configured `repeatinterval`, in which case the publisher is repeated indefinitely.
`repeatinterval=value`	Specifies the number of seconds between publisher repetitions. The default value is 0.

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

Table 36 *Required Keys*

Key-Value Pair	Description
`publishurl=` *pubsubURL*	Specifies the standard URL in the form **`dfESP://`** ***`host :port/ project/continuousquery /window`***.
`subscribeurl=` *pubsubURL*	Specifies the standard URL in the form **`dfESP://`** ***`host :port/ project/continuousquery /window`***. Append `?rmretdel=true | false` if needed.

Table 37 *Optional Keys*

Key-Value Pair	Description
`loglevel=trace | debug | info | warn | error | fatal | off`	Sets the logging level for the adapter. This is the same range of logging levels that you can set in the `C_dfESPpubsubInit()` publish/subscribe API call and in the engine `initialize()` call. The default level is `warn`.
`logconfigfile=` *file*	Specifies the log configuration file.
`tokenlocation=` *location*	Specifies the location of the file in the local filesystem that contains the OAuth token required for authentication by the publish/subscribe server.

Key-Value Pair	Description
`restartonerror=true | false`	When `true`, specifies to restart the adapter if a fatal error is reported.

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 subscribe operations only)
- json
- syslog (supports only publish operations)
- xml
- Opaque string field (supports only subscribe operations, copies the contents of the specified *opaquestringfield* in the subscribed window to the file or socket)
- opaquebinary (file only, no socket support)
- CAT240 - EUROCONTROL Specification for Surveillance Data Exchange ASTERIX

IMPORTANT The HDAT data type is not supported in a Microsoft Windows environment

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

When the connector implements a socket connection, it might act as a client or server, regardless of whether it is a publisher or subscriber. When you specify both *host* and *port* in the **fsname**, the connector implements the client. The configured host and port specify the network peer implementing the server side of the connection. However, when *host* is blank (that is, when **fsname** has the form ": *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 38 *Required Parameters for File and Socket Connectors*

Parameter	Description
fsname	Specifies the input file for publishers, output file for subscribers, or socket connection with the form *host*: *port*. Leave *host* blank to implement a server instead of a client.
fstype	Specifies the type of file: binary | csv | xml | json | syslog | hdat | cef | asterixcat240 | *opaquestringfield* | qpaquebinary | *opaquestringfield* cef, syslog, and opaquebinary are supported only for publishers. hdat, *opaquestringfield*, and*opaquebinaryfield* are supported only for subscribers. For more information about the asterixcat240 type, see "ASTERIX Category 240 File and Socket Connector Notes".
type	Specifies whether to publish or subscribe.

Table 39 *Required Parameters for Subscriber File and Socket Connectors*

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

Table 40 *Optional Parameters for Subscriber File and Socket Connectors*

Parameter	Description
collapse	Converts UPDATE_BLOCK events to UPDATE events in order to make subscriber output publishable.
configfilesection	Specifies the name of the section in **/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/**

Parameter	Description
	connectors.config (Linux) or **%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.config** (Windows) to parse for configuration parameters. Specify the value as `[configfilesection ]`.
`dateformat`	Specifies the format of `DATE` and `STAMP` fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (`DATE`) or microseconds (`STAMP`) since epoch. The `dateformat` parameter accepts any time format that is supported by the UNIX `strftime` function.
`doubleprecision`	Specifies the number of fractional digits in the ASCII representation of a double. The default value is 6.
`hdatcashostport`	Specifies the CAS server host and port. Applies only to the **hdat** connector.
`hdatcaspassword`	Specifies the CAS server password. Applies only to the **hdat** connector.
`hdatcaspasswordencrypted`	Specifies that `hdatcaspassword` is encrypted
`hdatcasusername`	Specifies the CAS server user name. Applies only to the **hdat** connector.
`hdatfilename`	Specifies the name of the Objective Analysis Package Data (HDAT) file to be written to the Hadoop Distributed File System (HDFS). Include the full path, as shown in the HDFS browser when you browse the name node specified in the `fsname` parameter. Do not include the `.sashdat` extension. Applies only to and is required for the **hdat** connector.
`hdatlasrhostport`	Specifies the SAS LASR Analytic Server host and port. Applies only to the **hdat** connector.
`hdatlasrkey`	Specifies the path to **tklasrkey.sh**. By default, this file is located in **$DFESP_HOME/bin** . Applies only to the **hdat** connector.
`hdatmaxdatanodes`	Specifies the maximum number of data node connections. The default value is the total number of live data nodes known by the name node. This parameter is ignored when `hdatnumthreads` <=1. Applies only to the **hdat** connector.
`hdatmaxstringlength`	Specifies in bytes the fixed size of string fields in Objective Analysis Package Data (HDAT) files. You must specify a multiple of 8. Strings are padded with spaces. Applies only to and is required for the **hdat** connector.

Parameter	Description
	To specify unique string lengths per column, configure a comma-separated string of values, using no spaces. Every value must be a multiple of 8, and the number of values must be equal to the number of string fields in the subscribed window schema.
`hdatnumthreads`	Specifies the size of the thread pool used for multi-threaded writes to data node socket connections. A value of 0 or 1 indicates that writes are single-threaded and use only a name-node connection. Applies only to and is required for the **`hdat`** connector.
`hdfsblocksize`	Specifies in Mbytes the block size used to write an Objective Analysis Package Data (HDAT) file. Applies only to and is required for the **`hdat`** connector.
`hdfsnumreplicas`	Specifies the number of Hadoop Distributed File System (HDFS) replicas created with writing an Objective Analysis Package Data (HDAT) file. The default value is 1. Applies only to the **`hdat`** connector.
`header`	`false` | `true` | `full` For a CSV subscriber, specifies to write a header row that shows comma-separated fields. Set the value to `full` to include `opcode`, `flags`, in the header line.
`maxfilesize`	Specifies the maximum size in bytes of the subscriber output file. When reached, a new output file is opened. When configured, a timestamp is appended to all output filenames. This parameter does not apply to socket connectors with `fstype` other than **`hdat`**.
`periodicity`	Specifies the interval in seconds at which the subscriber output file is closed and a new output file opened. When configured, a timestamp is appended to all output filenames for when the file was opened. For socket connectors, use this parameter only when `fstype` is set to **`hdat`**.
`rate`	When latency mode is enabled, transmit at this specified rate to generated output files.
`rmretdel`	Specifies to remove all delete events from event blocks received by a subscriber that were introduced by a window retention policy.
`unbufferedoutputstreams`	Specifies to create an unbuffered stream when writing to a file or socket. By default, streams are buffered.

Parameter	Description
hdatsecure	Specifies to use SSH to connect to secure Hadoop plugins. The default value is **false**. Applies only to the **hdat** connector.
hdatsecuresshloginname	When hdatsecure=true , specifies the SSH log on name. Invalid whenhdatsecuresshidentityfile is not configured. The default value is the current user. Applies only to the **hdat** connector.
hdatsecuresshidentityfile	When hdatsecure=true , specifies the SSH identity file. Invalid when hdatsecuresshloginname is not configured. The default value is none. Applies only to the **hdat** connector.
hdatsecurehadoopcmdpath	When hdatsecure=true , specifies the full Hadoop command path on the remote name and data nodes. The default value is **"/opt/hadoop/bin/hadoop"** . Applies only to the **hdat** connector.

Table 41 *Optional Parameters for Publisher File and Socket Connectors*

Parameter	Description
addcsvflags	Specifies the event type to insert into input CSV events (with a comma). Valid values are normal and partialupdate .
addcsvopcode	Prepends an opcode and comma to incoming CSV events. The opcode is Insert unless publishwithupsert is enabled.
blocksize	Specifies the number of events to include in a published event block. The default value is 1.
cefsyslogprefix	When fstype=cef , specifies that CEF events contain the syslog prefix.
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*].
csvfielddelimiter	Specifies the character delimiter for field data in input CSV events. The default delimiter is the , character.
dateformat	Specifies the format of DATE and STAMP fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (DATE) or microseconds (STAMP) since epoch. The dateformat parameter accepts any time format that is supported by the UNIX strftime function.

Parameter	Description
growinginputfile	Enables reading from a growing input file by publishers. When enabled, the publisher reads indefinitely from the input file until the connector is stopped or the server drops the connection. This parameter does not apply to socket connectors.
header	Specifies the number of input lines to skip before starting publish operations. The default value is 0. This parameter applies only to csv and syslog publish connectors.
ignorecsvparseerrors	Specifies that when a field in an input CSV event cannot be parsed, the event is dropped, an error is logged, and publishing continues.
maxevents	Specifies the maximum number of events to publish.
noautogenfield	Specifies that input events are missing the key field that is autogenerated by the source window. **Note:** This parameter applies only when the format of input data is CSV.
prebuffer	Controls whether event blocks are buffered to an event block vector before doing any injects. The default value is "false". Not valid with growinginputfile or for a socket connector
publishwithupsert	Build events with opcode=Upsert instead of opcode=Insert.
rampupsecs	Specifies the number of seconds to linearly ramp up from 0 to the transmit rate that you request with the rate parameter. The default value is 0.
rate	Specifies the requested transmit rate in events per second.
repeatcount	Specifies the number of times to repeat the publish operation. Applies to publishers that publish a finite number of events. The default value is 0.
republishmodifiedfile	After the file is published, continue running and republish the file every time that it is modified. It is recommended to write the new version of the file to a different file name and then rename that file to the original file name after the update is complete. This prevents the connector from prematurely reading the file before it has been completely updated. This parameter is invalid for socket connectors, or when growinginputfile is true, or when repeatcount is configured.
transactional	Sets the event block type to transactional. The default event block type is normal.

Parameter	Description
`stripnullsfromcsv`	Strip all nulls from input CSV events.
`cat240numthreads`	Specifies the thread pool size used to parse Asterix CAT240 data blocks. Specify a positive integer value. The default value is 1. This parameter is ignored unless `fstype=asterixcat240`. For every CAT240 data block that is received by the publisher, a thread is allocated from the thread pool to parse the data block. When all threads are in use, the data block is queued and parsed after a thread becomes available.

ASTERIX Category 240 File and Socket Connector Notes

You can use the file and socket connector and adapter to stream radar video transmissions that comply with EUROCONTROL Specification for Surveillance Data Exchange ASTERIX Category 240. For the connector, specify a value of `asterixcat240` for the `fstype`. For the adapter, specify `fstype = asterixcat240`.

The publisher opens a client socket connection to the *`host:port`* that you configured for the `fsname` parameter. After the publisher connects, it continuously reads the socket for Asterix CAT240 messages, and builds a single event per each message received.

The Source window schema contains the following fields:

```
id*:int64,cat240datablock:blob,pointcloud:array(dbl)
```

Alternatively, the Source window schema can contain the following fields:

```
id*:int64,cat240datablock:blob
```

In that case, no point clouds are parsed.

Table 42 *Schema Fields*

Field	Description
`id`	An incrementing event index.
`cat240datablock`	A copy of the received message.

`pointcloud`	An array of doubles that contain repeated sequences of `x/y/z/amplitude` values, in the order received in the CAT240 message.
	The `z` coordinate is always zero. The `x` and `y` values are calculated for every nonzero amplitude cell value in the CAT240 data, based on that cell's range and azimuth.
	The x and y values are calculated as follows:
	`sigma = 2pi * azimuth`
	`x = range * sin(sigma)`

Field	Description
	`y = range * cos(sigma)` The `range` is measured in meters and is calculated according to the formula in the CAT240 specification. The `azimuth` is measured in degrees and is the average of the start and end azimuth values that are contained in the CAT240 message.

Enable debug level logging in order to see all the values that are received from the stream and all intermediate values calculated by the connector.

The subscriber runs a socket server on the port configured in the `fsname` parameter. The ESP server supports multiple concurrent client connections. The subscriber requires that the subscribed window schema contains a `cat240datablock:blob` field, which is expected to contain a copy of that field as written by the publisher. The ESP server writes the binary contents of that field to all connected clients for every event produced by the window. Every client receives the same data.

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 Reference* 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='udate' type='date'/>
      <field name='hostname' type='string'/>
     <field name='message' type='string'/>
  </fields>
</schema>
```

XML File and Socket Connector Data Format

The following XML elements are valid:

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

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

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

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

`opcode` **Tag Value**	**Opcode**
`i`	`Insert`
`u`	`Update`
`p`	`Upsert`
`d`	`Delete`
`s`	`Safe delete`

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

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

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

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

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

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)
- Opaque string field (subscriber only, copies the contents of the specified *opaquestringfield* in the subscribed window to the file or socket)
- opaquebinary
- CAT240 - EUROCONTROL Specification for Surveillance Data Exchange ASTERIX

IMPORTANT The HDAT data type is not supported in a Microsoft Windows environment

dfesp_fs_adapter -C *key1 =val1,key2 =val2,key3 =val3, ...*

Subscriber Usage

Table 43 *Required Keys*

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

Key-Value Pair	Description
`fstype=binary | csv | xml | json | hdat | asterixcat240 | opaquestringfield | opaquebinaryfield`	Specifies the file system type. For *opaquebinaryfield*, specifies the name of a blob field in the subscribed window schema. For more information about the `asterixcat240` type, see "ASTERIX Category 240 File and Socket Connector Notes".

Table 44 *Optional Keys*

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

Key-Value Pair	Description
	\connectors.config (Windows) to parse for configuration parameters. Specify the value as [*configfilesection*].
latencymode=true | false	When true , specifies latency mode.
restartonerror= true | false	When true , restarts the adapter if a fatal error is reported.
transportconfigfile= *file*	Specifies the publish/subscribe transport configuration file. For solace , the default is **./solace.cfg**. For tervela , the default is **./client.config**. For rabbitmq , the default is **./rabbitmq.cfg**. For kafka , the default is **./kafka.cfg**. **Note:** No transport configuration file is required for native transport.
periodicity= *seconds*	Specifies the output file time (in seconds). The active file is closed and a new active file opened with the timestamp appended to the filename
maxfilesize= *bytes*	Specifies the output file volume (in bytes). The active file is closed and a new active file opened with the timestamp appended to the filename.
aggrsize= *statsforblocks*	Specifies, in latency mode, statistics for aggregation block size. You can follow the value with a comma-separated value to specify the number of beginning and ending aggregation blocks to ignore in latency calculations.
hdatfilename= *filename*	Specifies the filename to write to Hadoop Distributed File System (HDFS).
hdatmaxdatanodes= *numnodes*	Specifies the maximum number of data nodes. The default value is the number of live data nodes.
hdfsblocksize= *mbytes*	Specifies the Hadoop Distributed File System (HDFS) block size in MB.
hdfsnumreplicas= *number*	Specifies the Hadoop Distributed File System (HDFS) number of replicas. The default value is 1.
hdatnumthreads= *numthreads*	Specifies the adapter thread pool size. A value <= 1 means that no data nodes are used.
hdatmaxstringlength= *size*	Specifies the fixed size to use for string fields in HDAT records. The value must be a multiple of 8.

Key-Value Pair	Description
`hdatlasrhostport= hostport`	Specifies the LASR server host and port.
`hdatlasrkey= path`	Specifies the path to tklasrkey.sh. By default, this file is located in **`$DFESP_HOME/bin`**
`hdatcashostport= hostport`	Specifies the CAS server host and port.
`hdatcasusername= username`	Specifies the CAS server user name.
`hdatcaspassword= password`	Specifies the CAS server password. **Note:** When required, you can configure this parameter with an encrypted password. The encrypted version of the password can be generated using OpenSSL, which must be installed on your system. When 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 'hdatcaspassword' | openssl enc -e -aes-256-cbc -md md5\ -a -salt -pass pass:'espFSconnectorUsedByUser=hdatcasusername'` Then copy the encrypted password into your `hdatcaspassword` parameter and enable the `hdatcaspasswordencrypted` parameter.
`hdatcaspasswordencrypted =true | false`	When true, `hdatcaspassword` is encrypted.
`latencyblksize= numevents`	Specifies the block size (in number of events) of memory to allocate for storing timestamps when in latency mode. Additional blocks are allocated as required. Required when you specify `latencymode`.
`unbufferedoutputstreams=true | false`	When `true` , specifies to create an unbuffered stream when writing to a file or socket. By default, streams are buffered.
`header= headerrow`	For a CSV subscriber, writes a header row that shows comma-separated field names. Set to "full" to include "opcode, flags," in header line.
`doubleprecision= number`	Specifies the number of fractional digits in the ASCII representation of a double. The default value is 6.
`hdatsecure=true | false`	When true, specifies to use SSH to connect to secure Hadoop plugins.

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

Publisher Usage

Table 45 *Required Keys*

Key-Value Pairs	Description
`type=pub`	Specifies a publisher adapter.
`url=` *pubsubURL*	Specifies the standard URL in the form **`dfESP://`** ***`host :port/ project/continuousquery /window`***.
`fsname=` *output*	Specifies the publisher input file or socket "*`host :port`*". Leave *host* blank to implement a server.
`fstype=binary | csv | xml | json | syslog | cef | asterixcat240 | opaquebinary`	Specifies the file system type. For more information about the `asterixcat240` type, see "ASTERIX Category 240 File and Socket Connector Notes".

Table 46 *Optional Keys*

Key-Value Pairs	Description
`dateformat=` *format*	Specifies the format of `DATE` and `STAMP` fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (`DATE`) or microseconds (`STAMP`) since epoch. The `dateformat` parameter accepts any time format that is supported by the UNIX `strftime` function.
`rate=` *transmitrate*	Specifies the rate to write files when latency mode is enabled.

Key-Value Pairs	Description
`rampupsecs=` *seconds*	Specifies the number of seconds to linearly ramp up from 0 to the transmit rate that you request with `rate`. The default is 0.
`gdconfig=` *file*	Specifies the guaranteed delivery configuration file.
`transport=native | solace | tervela | rabbitmq | kafka`	Specifies the transport type. If you specify `solace`, `tervela` , `rabbitmq`, or `kafka` transports instead of the default `native` transport, then use the required client configuration files specified in the description of the C++ C_dfESPpubsubSetPubsubLib() API call.
`loglevel=trace | debug | info | warn | error | fatal | off`	Sets the logging level for the adapter. This is the same range of logging levels that you can set in the `C_dfESPpubsubInit()` publish/subscribe API call and in the engine `initialize()` call. The default level is `warn`.
`logconfigfile=` *logconfigfile*	Specifies the log configuration file.
`tokenlocation=` *location*	Specifies the location of the file in the local filesystem that contains the OAuth token required for authentication by the publish/subscribe server.
`configfilesection=[` *section*`]`	Specifies the name of the section in **`/opt/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*`]`.
`latencymode=true | false`	When `true` , specifies latency mode.
`restartonerror= true | false`	When `true` , restarts the adapter if a fatal error is reported.
`transportconfigfile=` *file*	Specifies the publish/subscribe transport configuration file. For `solace` , the default is **`./solace.cfg`**. For `tervela` , the default is **`./client.config`**. For `rabbitmq` , the default is **`./rabbitmq.cfg`**. For `kafka` , the default is **`./kafka.cfg`**. **Note:** No transport configuration file is required for native transport.
`blocksize=` *size*	Sets the block size. The default value is 1.
`maxevents=` *number*	Specifies the maximum number of events to publish.

Key-Value Pairs	Description
`transactional=true | false`	When `true` , event blocks are transactional. By default, event blocks are normal.
`growinginputfile=true | false`	When `true` , read from the growing file
`prebuffer=true | false`	When `true` , buffer all event blocks prior to publishing.
`header=` *`numlines`*	For CSV publisher, the number of header lines to ignore.
`quiesceproject=true | false`	When `true` , quiesce project after all events are injected into the Source window.
`ignorecsvparseerrors=true | false`	When `true` , drop event when a field fails CSV parsing and continue.
`csvfielddelimiter=` *`character`*	Specifies the character that delimits CSV fields, default = ,
`noautogenfield=true | false`	When `true` , input events are missing the key field autogenerated by the Source window. **Note:** This parameter applies only when the format of input data is CSV.
`addcsvopcode=true | false`	When `true` , prepend opcode to input CSV events.
`addcsvflags=none | normal | partialupdate`	Specifies the event type to insert into input CSV event.
`publishwithupsert=true | false`	When `true` , build events with opcode = Upsert instead of insert.
`cefsyslogprefix=true | false`	When `true` , CEF events contain the syslog prefix. By default, they do not.
`repeatcount=` *`number`*	Specifies number of times to repeat, default = 0.
`stripnullsfromcsv=true | false`	When `true` , strip all nulls from input CSV events.
`cat240numthreads`	Specifies the thread pool size used to parse Asterix CAT240 data blocks. Specify a positive integer value. The default value is 1. This parameter is ignored unless `fstype=asterixcat240`. For every CAT240 data block that is received by the publisher, a thread is allocated from the thread pool to parse the data block. When all threads are in use, the data block is queued and parsed after a thread becomes available.

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

IMPORTANT This functionality is no longer supported or included in SAS Viya 3.5 revision 24w44. No updates to this functionality are available for earlier releases.

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 ,...*

IMPORTANT The default values of Boolean parameters for this adapter are **`false`**. You cannot manually reset a Boolean parameter from **`true`** to **`false`** with `-C`. To reset the value, simply remove the parameter from the command line.

Table 47 *Required Keys*

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

Table 48 *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.
`gdconfigfile= file`	Specifies the guaranteed delivery configuration file.
`pubsublib= native | solace | tervela | rabbitmq | kafka`	Specifies the transport type. When you specify the `solace`, `tervela`, `rabbitmq`, or `kafka` transports instead of the default `native` transport, use the required client configuration files specified at "Using the Java Publish/Subscribe API" in *SAS Event Stream Processing: Publish/Subscribe API Reference* .
`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.
`quiesceproject=true | false`	When true, quiesce the project after all events are injected into the Source window. The default value is false.

Key-Value Pair	Description
`publishwithupsert=true | false`	When true, build events with `opcode=Upsert`. The default value is false, and events are built with `opcode=Insert`. **`javaadapters.config`** parameter name: `publishwithupsert`
`transactional=true | false`	When true, event blocks are transactional. By default, they are normal.
`restartonerror=true | false`	When true, restarts the adapter when a fatal error is reported. The default value is false. **`javaadapters.config`** parameter name: `restartonerror`
`maxevents=` *number*	Specifies the maximum number of events to publish.
`transportconfigfile=` *file*	Specifies the full path to the publish/subscribe transport configuration file. The default file depends on the transport type specified with through the `pubsublib` parameter: For `solace` , the default file is **`solace.cfg`**. For `tervela` , the default file is **`client.config`**. For `rabbitmq` , the default file is **`rabbitmq.cfg`**. For `kafka` , the default file is **`kafka.cfg`**. **Note:** No transport configuration file is required for native transport.

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

IMPORTANT This functionality is no longer supported or included in SAS Viya 3.5 revision 24w44. No updates to this functionality are available for earlier releases.

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.

Note: It is recommended that you use Hadoop version 2.5.0 with the HDFS adapter. Other versions have not been tested with the adapter.

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 ,...*

IMPORTANT The default values of Boolean parameters for this adapter are **`false`**. You cannot manually reset a Boolean parameter from **`true`** to **`false`** with `-C`. To reset the value, simply remove the parameter from the command line.

Table 49 *Required Keys*

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

Table 50 *Optional Keys*

Key-Value Pair	Description
`hdfsblocksize=` *size*	Specifies the HDFS block size in MB. The default is 64MB.
`configfilesection=` *section*	Specifies the name of the section in **`/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/javaadapters.config`** (Linux) or **`%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\javaadapters.config`** (Windows) to parse for configuration parameters.
`dateformat=` *format*	Specifies the format of `DATE` and `STAMP` fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (`DATE`) or microseconds (`STAMP`) since epoch. The `dateformat` parameter accepts any time format that is supported by the UNIX `strftime` function.
`gdconfigfile=` *file*	Specifies the guaranteed delivery configuration file.
`jaasconf=` *file*	Specifies the location of the JAAS configuration on the local file system. This file is used for Kerberos authentication to the Hadoop grid. By default, there is no authentication.
`krb5conf=` *file*	Specifies the location of the Kerberos 5 configuration file on the local file system. This file is required for Kerberos authentication to the Hadoop grid. The default value is **`/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default /krb5.conf`** (Linux) or **`%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default \krb5.conf`** (Windows).
`pubsublib= native | solace | tervela | rabbitmq | kafka`	Specifies the transport type. When you specify the `solace`, `tervela`, `rabbitmq`, or `kafka` transports instead of the default `native` transport, use the required client configuration files specified at "Using the Java Publish/Subscribe API" in *SAS Event Stream Processing: Publish/Subscribe API Reference* .
`maxfilesize=` *size*	Specifies the output file periodicity in bytes.
`hdfsnumreplicas=` *number*	Specifies the HDFS number of replicas. The default value is 1.
`loglevel=severe | warning | info`	Specifies the application logging level. The default value is `warning` .

Key-Value Pair	Description
`tokenlocation=` *location*	Specifies the location of the file in the local file system that contains the OAuth token required for authentication by the publish/subscribe server.
`periodicity=` *seconds*	Specifies the output file periodicity in seconds. Invalid if data is not insert-only.
`opaquestringfield=` *field*	Specifies the subscribed window field from which to extract the output line.
`restartonerror=true | false`	When true, restarts the adapter if a fatal error is reported. The default value is false.
`transportconfigfile=` *file*	Specifies the full path to the publish/subscribe transport configuration file. The default file depends on the transport type specified with through the `pubsublib` parameter: For `solace` , the default file is **`solace.cfg`**. For `tervela` , the default file is **`client.config`**. For `rabbitmq` , the default file is **`rabbitmq.cfg`**. For `kafka` , the default file is **`kafka.cfg`**. **Note:** No transport configuration file is required for native transport.

Publisher usage:

dfesp_hdfs_publisher -C *key1* =*val1*, *key2* =*val2*, *key3* =*val3* , ...

Table 51 *Required Keys*

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

Table 52 *Optional Keys*

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

Key-Value Pair	Description
	microseconds (`STAMP`) since epoch. The `dateformat` parameter accepts any time format that is supported by the UNIX `strftime` function.
`ignorecsvparseerrors=true | false`	When true, drops an event when a field in an input CSV event cannot be parsed. An error is logged and publishing continues. The default value is false. **`javaadapters.config`** parameter name: `ignorecsvparseerrors`
`gdconfigfile=` *`file`*	Specifies the guaranteed delivery configuration file.
`jaasconf=` *`file`*	Specifies the location of the JAAS configuration on the local file system. This file is used for Kerberos authentication to the Hadoop grid. By default, there is no authentication.
`krb5conf=` *`file`*	Specifies the location of the Kerberos 5 configuration file on the local file system. This file is required for Kerberos authentication to the Hadoop grid. The default value is **`/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default /krb5.conf`** (Linux) or **`%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default \krb5.conf`** (Windows).
`pubsublib= native | solace | tervela | rabbitmq | kafka`	Specifies the transport type. When you specify the `solace`, `tervela`, `rabbitmq`, or `kafka` transports instead of the default `native` transport, use the required client configuration files specified at "Using the Java Publish/Subscribe API" in *SAS Event Stream Processing: Publish/Subscribe API Reference* .
`csvfielddelimiter=` *`delimiter`*	Specifies the character delimiter for field data in input CSV events. The default delimiter is the , character.
`noautogenfield=true | false`	When **`true`**, input events are missing the key field that is autogenerated by the Source window. The default value is false, and events contain all schema fields. **Note:** This parameter applies only when the format of input data is CSV.
`publishwithupsert=true | false`	When true, build events with `opcode=Upsert`. The default value is false, and events are built with `opcode=Insert`. **`javaadapters.config`** parameter name: `publishwithupsert`

Key-Value Pair	Description
`transactional=true | false`	When true, event blocks are transactional. The default value is false, and event blocks are normal.
`stripnullsfromcsv=true | false`	When true, strip all null characters from input CSV events. The default value is false.

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 'password' | openssl enc -e -aes-256-cbc -md md5 -a -salt\
 -pass pass:'espJMSadapterUsedByUser= userid'
```

Specify the encrypted *password* that you supplied for the `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 ,...*

> **IMPORTANT** The default values of Boolean parameters for this adapter are **`false`**. You cannot manually reset a Boolean parameter from **`true`** to **`false`** with `-C`. To reset the value, simply remove the parameter from the command line.

Table 53 *Required Keys*

Key-Value Pair	Description
`messagetype="BytesMessage" | "TextMessage" | "MapMessage"`	Specifies the JMS message type.
`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 54 *Optional Keys*

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

Key-Value Pair	Description
`dateformat=` *format*	Specifies the format of `DATE` and `STAMP` fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (`DATE`) or microseconds (`STAMP`) since epoch. The `dateformat` parameter accepts any time format that is supported by the UNIX `strftime` function.
`pwdencrypted=true | false`	When true, the JMS password is encrypted. The default value is false.
`protofile=` *file*	Specifies the .proto file that contains the message used for Google Protocol buffer support.
`gdconfigfile=` *file*	Specifies the Guaranteed Delivery configuration file.
`jmsuserid=` *userid*	Specifies the user ID for the JMS destination.
`jndidestname=` *name*	Specifies the JNDI name to look up for the JMS destination. The default value is `"jmsDestination"`. This option overrides settings in jndi.properties.
`pubsublib= native | solace | tervela | rabbitmq | kafka`	Specifies the transport type. When you specify the `solace`, `tervela`, `rabbitmq`, or `kafka` transports instead of the default `native` transport, use the required client configuration files specified at "Using the Java Publish/Subscribe API" in *SAS Event Stream Processing: Publish/Subscribe API Reference* .
`csvmsgpereventblock= true | false`	For CSV, a value of true specifies to send one message per event block. The default value is false: one message is sent per transactional event block and one message per event is sent in a non-transactional event block. In a multi-event message, CSV lines are delimited by the "line.separator" system property.
`csvmsgperevent= true | false`	For CSV, specifies to send one message per event. The default value is false: one message is sent per transactional event block and one message per event is sent in a non-transactional event block. In a multi-event message, CSV lines are delimited by the "line.separator" system property.
`loglevel=severe | warning | info`	Specifies the application logging level. The default value is `warning` .
`tokenlocation=` *location*	Specifies the location of the file in the local file system that contains the OAuth token required for authentication by the publish/subscribe server.

Key-Value Pair	Description
`jmspassword=` *password*	Specifies the password for the JMS destination.
`opaquestringfield=` *field*	Specifies the subscribed window field from which to extract the JMS TextMessage.
`restartonerror=true | false`	When true, restarts the adapter if a fatal error is reported. The default value is false.
`xmllib="axeda" | "turkcell"`	Specifies the JAR file derived from a third-party .xsd file that defines the XML contained in a JMS TextMessage. By default, the TextMessage contains CSV.
`transportconfigfile=` *file*	Specifies the full path to the publish/subscribe transport configuration file. The default file depends on the transport type specified with through the `pubsublib` parameter: For `solace` , the default file is **`solace.cfg`**. For `tervela` , the default file is **`client.config`**. For `rabbitmq` , the default file is **`rabbitmq.cfg`**. For `kafka` , the default file is **`kafka.cfg`**. **Note:** No transport configuration file is required for native transport.

Publisher Usage:

dfesp_jms_publisher -C *key1 =val1, key2 =val2, key3 =val3 ...*

Table 55 *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 56 *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 default value is false, and the opcode is Insert unless `-s` is enabled.

Key-Value Pair	Description
`pwdencrypted=true | false`	When true, the JMS password is encrypted. The default value is false.
`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. The default value is false.
`maxevents=` *number*	Specifies the maximum number of events to publish.
`pubsublib= native | solace | tervela | rabbitmq | kafka`	Specifies the transport type.When you specify the `solace`, `tervela`, `rabbitmq`, or `kafka` transports instead of the default `native` transport, use the required client configuration files specified at "Using the Java Publish/Subscribe API" in *SAS Event Stream Processing: Publish/Subscribe API Reference* .
`transportconfigfile=` *file*	Specifies the full path to the publish/subscribe transport configuration file. The default file depends on the transport type specified with through the `pubsublib` parameter: For `solace` , the default file is **`solace.cfg`**. For `tervela` , the default file is **`client.config`**. For `rabbitmq` , the default file is **`rabbitmq.cfg`**. For `kafka` , the default file is **`kafka.cfg`**. **Note:** No transport configuration file is required for native transport.
`jndifactname=` *name*	Specifies the JNDI name to look up for the JMS connection factory. The default value is `"ConnectionFactory"`. This option overrides settings in **`jndi.properties`** .
`blocksize=` *number*	Specifies the number of events per event block. The default value is 1.
`configfilesection=` *section*	Specifies the name of the section in **`/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/javaadapters.config`** (Linux) or **`%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default`**

Key-Value Pair	Description
	\javaadapters.config (Windows) to parse for configuration parameters.
`dateformat=` *format*	For messages of type TextMessage, specifies the format of datetime and timestamp fields. Specify a Java SimpleDateFormat pattern in quotation marks.
`ignorecsvparseerrors=true | false`	When true, drop an event and continue whenever a field fails CSV parsing. The default value is false.
`protofile=` *file*	Specifies the .proto file that contains the message used for Google Protocol buffer support.
`gdconfigfile=` *file*	Specifies the Guaranteed Delivery configuration file.
`jmsuserid=` *userid*	Specifies the user ID for the JMS destination.
`jndidestname=` *name*	Specifies the JNDI name to look up for the JMS destination. The default value is `"jmsDestination"`. This option overrides settings in **jndi.properties** .
`pubsublib= native | solace | tervela | rabbitmq | kafka`	Specifies the transport type. The default is `native`.
`csvfielddelimiter=` *delimiter*	Specifies the character delimiter for field data in input CSV events. The default delimiter is the , character.
`noautogenfield=true | false`	When **true**, input events are missing the key field that is autogenerated by the source window. The default value is false, and events contain all schema fields. **Note:** This parameter applies only when the format of input data is CSV.
`loglevel=severe | warning | info`	Specifies the application logging level. The default value is `warning` .
`jmspassword=` *password*	Specifies the password for the JMS destination.
`opaquestring=true | false`	When true, do not parse TextMessage. Instead, copy complete message to a ESP_UTF8STR field, where the Source window schema is assumed to be `"index:int64,message:string"` The default value is false.
`protomsg=` *message*	Specifies the message itself in the **.proto** file that is specified by the protofile parameter.

Key-Value Pair	Description
`publishwithupsert=true | false`	When true, build events with `opcode = upsert`. The default value is false, and `opcode=insert`.
`transactional=true | false`	When true, event blocks are transactional. The default value is false, and event blocks are normal.
`stripnullsfromcsv=true | false`	When true, strip all nulls from input CSV events. The default value is false.
`xmlib="axeda" | "turkcell"`	Specifies the JAR file derived from a third-party .xsd file that defines the XML contained in a JMS TextMessage. By default, the TextMessage contains CSV.

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

Note: If you are using Apache Kafka 4, you must use SAS Event Stream Processing 6.2 hot fix 70828 or higher.

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 except when a publisher has `kafkaconsumergroupid`. In that case, the partition is assigned by the broker.

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. When `kafkaconsumergroupid` is configured and `kafkainitialoffset` is not configured, the default initial offset is the last committed offset stored by the broker for this group ID. When the broker has no stored offset, the default starting offset is **`latest`**. To modify this default behavior, configure librdkafka parameter `auto.offset.reset` using the `kafkaglobalconfig` connector parameter.

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.

Configuring LibrdKafa and Zookeeper Libraries for Use with the Kafka Connector

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

Corresponding 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 57 *Required Parameters for the Kafka Subscriber Connector*

Parameter	Description
`type`	Specifies to subscribe. Must be set to "`sub`".
`kafkahostport`	Specifies one or more Kafka brokers in the following form: *`host:port ,host: port,....`*
`kafkatopic`	Specifies the Kafka topic.
`kafkatype`	Specifies **`binary`**, **`csv`** , **`json`**, **`protobuf`**, **`avro`**, or 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 '_'.

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 cluster as required to maintain message ID sequence.
snapshot	Specifies whether to send snapshot data. The only supported value is false . Subscriber clients reading messages sent by this subscriber can control their snapshot by configuring an appropriate initial offset into the Kafka partition.

Table 58 *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 **.proto** file that contains the Google Protocol Buffers message definition used to convert event blocks to protobuf messages. When you specify this parameter, you must also specify the protomsg parameter. This parameter is supported only when kafkatype is set to **protobuf**; it is otherwise ignored.
protomsg	Specifies the name of a Google Protocol Buffers message in the **.proto** file that you specified with the protofile parameter. Event blocks are converted into this message. This parameter is supported only when kafkatype is set to **protobuf**; it is otherwise ignored.
csvincludeschema	When kafkatype=CSV , specifies when to prepend output CSV data with the window's serialized schema. Valid values are never, once, and pereventblock. The default value is " never".

Parameter	Description
useclientmsgid	Uses the client-generated message ID instead of the engine-generated message ID when performing a failover operation and extracting a message ID from an event block.
configfilesection	Specifies the name of the section in **/opt/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*].
zookeeperhostport	Specifies the Zookeeper server in the form *host* :*port*.
kafkaglobalconfig	Specifies a semicolon-separated list of key=value strings to configure librdkafka global configuration values.
kafkapartition	Specifies the Kafka partition. Specify -1 to use all partitions in the topic. This value is not supported when hot failover is enabled. Required when kafkaconsumergroupid is not configured. Disallowed when kafkaconsumergroupid is configured.
kafkatopicconfig	Specifies a semicolon-separated list of key=value strings to configure librdkafka topic configuration values.
csvmsgperevent	For CSV, specifies to send one message per event. The default is one message per transactional event block or else one message per event.
csvmsgpereventblock	For CSV, specifies to send one message per event block. The default is one message per transactional event block or else one message per event.
doubleprecision	Specifies the number of fractional digits in the ASCII representation of a double. The default value is 6.
avroschemaregistryurl	Specifies the URL of the Apache Avro schema registry. This parameter is supported only when kafkatype is set to **avro**; it is otherwise ignored. See also "Publish/Subscribe API Support for Apache Avro Messaging" in *SAS Event Stream Processing: Publish/Subscribe API Reference*.
avroschemadefinition	Specifies the path to a file that contains an Apache Avro schema definition in JSON format. This schema is copied to the schema registry that is configured in the avroschemaregistryurl parameter. The default value is an Apache Avro schema that represents the subscribed window schema plus an opcode field. This definition is

Parameter	Description
	also copied to the schema registry. This parameter is supported only when `kafkatype` is set to **`avro`**; it is otherwise ignored.
`avroschemaname`	Specifies the name of an Apache Avro schema to copy from the schema registry that is configured in the `avroschemaregistryurl` parameter. The default value is an Apache Avro schema that represents the subscribed window schema plus an opcode field. This name is copied to the schema registry. This parameter is supported only when `kafkatype` is set to **`avro`**; it is otherwise ignored.
`fieldtokafkakey`	Specifies to copy the contents of the specified window string field to the Kafka message key. **Note:** This key is not valid when `kafkatype` is set to **`binary`** or **`json`**. If `kafkatype` is **`csv`**, then `csvmsgperevent` must also be configured in order for `fieldtokafkakey` to take effect.
`avroschemanoopcode`	Specifies to not include the event opcode in outbound Apache Avro schema and messages. This parameter is supported only when `kafkatype` is set to **`avro`**.
`useoffsetmsgid`	When performing a failover operation and extracting a message ID from an event block, use the offset of the consumed Kafka message instead of the engine-generated message ID.
`zookeeperloglevel`	Specifies the Zookeeper library log level. Valid values are **`debug`**, **`info`**, **`warn`**, and **`error`**. The default value is **`warn`**.
`kafkaheaderfieldnames`	Specifies a comma separated list of window schema field names. The CSV representation of the contents of those fields are copied to Kafka message headers. The window field values are then converted to NULL values, and thus are not included in Kafka message payloads. For every configured field, a header is added to every Kafka message. The header key is the field name and the header value is the CSV representation of the field value. If the field value is NULL, then no header is created. If the Kafka message represents multiple events, then the header aggregates all non-NULL values from the configured field, separated by commas. By default, Kafka message headers are unused.

Parameters for the Kafka Publisher Connector

Table 59 *Required Parameters for the Kafka Publisher Connector*

Parameter	Description
`type`	Specifies to publish. Must be set to "`pub`".
`kafkahostport`	Specifies one or more Kafka brokers in the following form: *`host:port ,host: port,….`*
`kafkatopic`	Specifies the Kafka topic.
`kafkatype`	Specifies **`binary`**, **`csv`** , **`json`**, **`protobuf`**, **`avro`** , or **`opaquestring`**. When the `kafkatype` is **`opaquestring`**, the Source window schema is assumed to be `"index:int64,message:string"`.
`urlhostport`	Specifies the *`host: port`* field in the metadata topic that is subscribed to on start-up. This combination fields metadata requests. To meet Kafka topic naming requirements, replace ':' with '_'.

Table 60 *Optional Parameters for the Kafka Publisher Connector*

Parameter	Description
`transactional`	When `kafkatype=csv` or `kafkatype=opaquestring`, sets the event block type to transactional. The default event block type is normal.
`blocksize`	When `kafkatype=csv` or `kafkatype=opaquestring`, specifies the number of events to include in a published event block. The default value is 1.
`dateformat`	Specifies the format of `DATE` and `STAMP` fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (`DATE`) or microseconds (`STAMP`) since epoch. The `dateformat` parameter accepts any time format that is supported by the UNIX `strftime` function.
`ignorecsvparseerrors`	Specifies that when a field in an input CSV event cannot be parsed, the event is dropped, an error is logged, and publishing continues.
`ignoreparseerrors`	Specifies that when a field in an input CSV, JSON, or Protobuf event cannot be parsed, the event is dropped, an error is logged, and publishing continues.
`protofile`	Specifies the **`.proto`** file that contains the Google Protocol Buffers message definition used to convert event blocks to protobuf

Parameter	Description
	messages. When you specify this parameter, you must also specify the `protomsg` parameter. This parameter is supported only when `kafkatype` is set to **`protobuf`**; it is otherwise ignored.
`protomsg`	Specifies the name of a Google Protocol Buffers message in the **`.proto`** file that you specified with the protofile parameter. Event blocks are converted into this message. This parameter is supported only when `kafkatype` is set to **`protobuf`**; it is otherwise ignored.
`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`*`]`.
`csvfielddelimiter`	Specifies the character delimiter for field data in input CSV events. The default delimiter is the `,` character.
`noautogenfield`	Specifies that input events are missing the key field that is autogenerated by the source window. **Note:** This parameter applies only when the format of input data is CSV.
`publishwithupsert`	Specifies to build events with `opcode = Upsert` instead of `opcode = Insert`.
`kafkainitialoffset`	Specifies the offset from which to begin consuming messages from the Kafka topic and partition. Valid values are `smallest`, `largest`, or an integer. The default value is `smallest`.
`kafkapartition`	Specifies the Kafka partition. Specify a value of **`-1`** in order to use all partitions in the topic. This parameter is required when `kafkaconsumregroupid` is not configured, and not allowed when `kafkaconsumregroupid` is configured.
`addcsvopcode`	Prepends an opcode and comma to write CSV events. The opcode is Insert unless `publishwithupsert` is enabled.
`addcsvflags`	Specifies the event type to insert into input CSV events (with a comma). Valid values are `normal` and `partialupdate`.
`kafkaglobalconfig`	Specifies a semicolon-separated list of `key=value` strings to configure `librdkafka` global configuration values.

Parameter	Description
kafkatopicconfig	Specifies a semicolon-separated list of key=value strings to configure librdkafka topic configuration values.
useclientmsgid	If the Source window has been restored from a persist to disk, then 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 supported only when kafkatype is set to **avro**; it is otherwise ignored.
keyfromkafkakey	Specifies to copy the Kafka message key to the window key field. This field must be type=string. This parameter is supported only when kafkatype is set to **protobuf**; it is otherwise ignored.
kakaconsumergroupid	Specifies the group ID for this Kafka container. The default value is a randomly generated string. This parameter is not supported when hot failover is enabled.
useoffsetmsgid	When reading a message from Kafka, copy the message offset into the resulting event block. This message offset is used when failover is enabled.
offsetfieldname	Specifies the name of an INT64 field in the Source window schema into which to copy the Kafka message offset. **Note:** This parameter is not supported when the kafkatype is set to **binary**.
partitionfieldname	Specifies the name of an INT32 field in the Source window schema into which to copy the Kafka message partition. **Note:** This parameter is not supported when the kafkatype is set to **binary**.
avronestingseparator	Specifies the character that separates nested Apache Avro fields in Source window fields. The default value is '_'.
kafkaheaderfieldnames	Specifies a comma-separated list of field names in the current window schema. The contents of the specified fields are copied from headers in Kafka messages instead of the message payloads. Configured fields must be of type STRING or BLOB. For every configured field, the field value is copied from a header whose key matches the field name. If there are multiple such headers in a single message, then all values are copied to the

Parameter	Description
	window field and separated by commas. In this case, the field must be of type STRING and not BLOB. If a received message has no header that matches a configured field name, then that field value is NULL. If a received message has a header that matches a configured field name and the message payload also contains data that is intended for that field, then the header value takes precedence. By default, Kafka message headers are ignored.

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,...*

Subscriber Usage

Table 61 *Required Keys*

Key-Value Pair	Description
`type=sub`	Specifies a subscriber adapter.
`url=` *pubsubURL*	Specifies the standard URL in the form **`dfESP://`** ***`host :port/ project/continuousquery /window`***. Append the following for subscribers:`?snapshot=false`. When `?snapshot=true`, the subscriber receives a collection of Insert events that are contained in the window at that point in time. The subscriber then receives a stream of events produced from the time of the snapshot onward. Those subsequent events can be Inserts, Updates, or Deletes. **Note:** `?snapshot=true` is invalid.

Key-Value Pair	Description
	Append `?collapse=true | false` or `?rmretdel=true | false` for subscribers if needed.
`kafkahostport= broker`	Specifies one or more Kafka brokers in the following form: `" host:port ,host: port,..."`.
`kafkatopic= topic`	Specifies the Kafka topic.
`kafkapartition= partition`	Specifies the Kafka partition. Specify `-1` to use all partitions in the topic. This value is not supported when hot failover is enabled.
`urlhostport= field`	Specifies the `host: port` field in the metadata topic to which the adapter subscribes (replace ':' with '_')
`numbufferedmsgs= number`	Specifies the maximum number of messages buffered by a standby subscriber connector.
`kafkatype= binary | csv | json | protobuf | avro`	Specifies the message format. For subscribers, the name of a string field in the subscribed window schema is also supported.

Table 62 Optional Keys

Key-Value Pair	Description
`dateformat= format`	Specifies the format of `DATE` and `STAMP` fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (`DATE`) or microseconds (`STAMP`) since epoch. The `dateformat` parameter accepts any time format that is supported by the UNIX `strftime` function.
`protofile= file`	Specifies the **`.proto`** file to be used for Google protocol buffer support. This key is ignored when `kafkatype` is not set to **`protobuf`**.
`protomsg= message`	Specifies the message itself in the **`.proto`** file that is specified by the protofile parameter. This key is ignored when `kafkatype` is not set to **`protobuf`**.
`gdconfig= file`	Specifies the guaranteed delivery configuration file.
`transport=native | solace | tervela | rabbitmq | kafka`	Specifies the transport type. If you specify `solace`, `tervela` , `rabbitmq`, or `kafka` transports instead of the default `native` transport, use the required client

Key-Value Pair	Description
	configuration files specified in the description of the C++ C_dfESPpubsubSetPubsubLib() API call.
`loglevel=trace | debug | info | warn | error | fatal | off`	Sets the logging level for the adapter. This is the same range of logging levels that you can set in the `C_dfESPpubsubInit()` publish/subscribe API call and in the engine `initialize()` call. The default level is `warn`.
`logconfigfile=` *file*	Specifies the log configuration file.
`tokenlocation=` *location*	Specifies the location of the file in the local filesystem that contains the OAuth token required for authentication by the publish/subscribe server.
`configfilesection=[` *section*`]`	Specifies the name of the section in **`/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config`** (Linux) or **`%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.config`** (Windows) to parse for configuration parameters. Specify the value as `[`*configfilesection*`]`.
`kafkaglobalconfig=` *list*	Specifies a semicolon-separated list of "`key=value`" strings to configure `librdkafka` global configuration values.
`kafkatopicconfig=` *list*	Specifies a semicolon-separated list of "`key=value`" strings to configure `librdkafka` topic configuration values.
`restartonerror=true | false`	When `true`, specifies to restart the adapter if a fatal error is reported.
`transportconfigfile=` *file*	Specifies the publish/subscribe transport configuration file. For `solace`, the default is **`./solace.cfg`**. For `tervela`, the default is **`./client.config`**. For `rabbitmq`, the default is **`./rabbitmq.cfg`**. For `kafka`, the default is **`./kafka.cfg`**. **Note:** No transport configuration file is required for native transport.
`csvincludeschema=never | once | pereventblock`	When `kafkatype=CSV`, specifies when to prepend output CSV data with the window's serialized schema. The default value is "`never`".
`csvmsgperevent= true | false`	When true, for CSV, specifies to send one message per event. The default is one message per transactional event block or else one message per event.

Key-Value Pair	Description
`csvmsgpereventblock=true | false`	When true, for CSV, specifies to send one message per event block. The default is one message per transactional event block or else one message per event.
`doubleprecision=` *`number`*	Specifies the number of fractional digits in the ASCII representation of a double. The default value is 6.
`avroschemaregistryurl=` *`url`*	Specifies the URL of the Apache Avro schema registry. This parameter is ignored when `kafkatype` is not set to **`avro`**.
`avroschemadefinition=` *`schema`*	Specifies the path to a file that contains an Apache Avro schema definition in JSON format. This schema is copied to the schema registry configured in the `avroschemaregistryurl` parameter. The default value is an Apache Avro schema that represents the subscribed window schema plus an opcode field. This schema is also copied to the schema registry. This parameter is ignored when `kafkatype` is not set to **`avro`**.
`avroschemaname=` *`name`*	Specifies the name of an Apache Avro schema to copy from the schema registry configured in the `avroschemaregistryurl` parameter. The default value is an Apache Avro schema that represents the subscribed window schema plus an opcode field. This name is copied to the schema registry. This parameter is ignored when `kafkatype` is not set to **`avro`**.
`fieldtokafkakey=` *`fieldname`*	Specifies to copy the contents of the specified window string field to the Kafka message key. **Note:** This key is not valid when `kafkatype` is set to **`binary`** or **`json`**. If `kafkatype` is **`csv`**, then `csvmsgperevent` must also be configured in order for `fieldtokafkakey` to take effect.
`avroschemanoopcode= true | false`	When true, specifies to not include the event opcode in outbound Apache Avro schema and messages. This key is ignored when `kafkatype` is not set to **`avro`**.
`kafkaheaderfieldnames=`*`list`*	Specifies a comma separated list of window schema field names. The CSV representation of the contents of those fields are copied to Kafka message headers. The window field values are then converted to NULL values, and thus are not included in Kafka message payloads. For every configured field, a header is added to every Kafka message. The header key is the field name and the header value is the CSV representation of the field value. If the field value is NULL, then no header is created. If the Kafka message represents multiple events, then the header

Key-Value Pair	Description
	aggregates all non-NULL values from the configured field, separated by commas. By default, Kafka message headers are ignored.

Publisher Usage

Table 63 *Required Keys*

Key-Value Pair	Description
`type=pub`	Specifies a publisher adapter
`url=` *pubsubURL*	Specifies the standard URL in the form **`dfESP://`** ***`host :port/ project/continuousquery /window`***.
`kafkahostport=` *broker*	Specifies one or more Kafka brokers in the following form: `"`*`host:port ,host: port,...`*`"`.
`kafkatopic=` *topic*	Specifies the Kafka topic.
`urlhostport=` *field*	Specifies the *`host: port`* field in the metadata topic to which the adapter subscribes (replace ':' with '_')
`kafkatype=binary | csv | json | protobuf | avro | opaquestring`	Specifies the message format. `opaquestring` is valid only for publishers. For `opaquestring`, the Source window schema is assumed to be `"index:int64,message:string"` .

Table 64 *Optional Keys*

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

Key-Value Pair	Description
protomsg= *message*	Specifies the message itself in the **.proto** file that is specified by the protofile parameter. This key is ignored when kafkatype is not set to **protobuf**.
gdconfig= *file*	Specifies the guaranteed delivery configuration file.
transport=native | solace | tervela | rabbitmq | kafka	Specifies the transport type. If you specify solace, tervela , rabbitmq, or kafka transports instead of the default native transport, then use the required client configuration files specified in the description of the C++ C_dfESPpubsubSetPubsubLib() API call.
loglevel=trace | debug | info | warn | error | fatal | off	Sets the logging level for the adapter. This is the same range of logging levels that you can set in the C_dfESPpubsubInit() publish/subscribe API call and in the engine initialize() call. The default level is warn.
logconfigfile= *file*	Specifies the log configuration file.
tokenlocation= *location*	Specifies the location of the file in the local filesystem that contains the OAuth token required for authentication by the publish/subscribe server.
configfilesection=[*section*]	Specifies the name of the section in **/opt/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*].
kafkapartition= *partition*	Specifies the Kafka partition. Specify a value of **-1** to use all partitions in the topic. This key is required when kafkaconsumergroupid is not configured. It is not allowed when kafkaconsumergroupid is configured.
kafkaglobalconfig= *list*	Specifies a semicolon-separated list of "key=value" strings to configure librdkafka global configuration values.
kafkatopicconfig= *list*	Specifies a semicolon-separated list of "key=value" strings to configure librdkafka topic configuration values.
restartonerror=true | false	When true , specifies to restart the adapter if a fatal error is reported.
transportconfigfile= *file*	Specifies the publish/subscribe transport configuration file. For solace , the default is **./solace.cfg**.

Key-Value Pair	Description
	For `tervela` , the default is **`./client.config`**. For `rabbitmq` , the default is **`./rabbitmq.cfg`**. For `kafka` , the default is **`./kafka.cfg`**. **Note:** No transport configuration file is required for native transport.
`blocksize=` *`size`*	Sets the block size. The default value is 1.
`transactional=true | false`	When `true` , events blocks are transactional. By default, event blocks are normal.
`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.
`ignoreparseerrors=true | false`	Specifies that when a field in an input CSV, JSON, or Protobuf event cannot be parsed, the event is dropped, an error is logged, and publishing continues.
`csvfielddelimiter=` *`delimiter`*	Specifies the character delimiter for field data in input CSV events. The default delimiter is the `,` character.
`noautogenfield=true | false`	When `true` , specifies that input events are missing the key field that is autogenerated by the Source window. **Note:** This parameter applies only when the format of input data is CSV.
`publishwithupsert=true | false`	When `true` , build events with opcode = Upsert instead of Insert.
`kafkainitialoffset=` *`offset`*	Specifies the offset from which to begin consuming from the Kafka partition. Valid values are "smallest", "largest", or an integer. The default is "smallest".
`addcsvopcode=true | false`	When `true` , prepends an opcode and comma to write CSV events. The opcode is Insert unless `publishwithupsert` is `true` .
`addcsvflags=none | normal | partialupdate`	Specifies the event type to Insert into input CSV events (with comma).
`maxevents=` *`number`*	Specifies the maximum number of events to publish.
`quiesceproject=true | false`	When `true` , quiesces the project after all events are injected into the Source window.

Key-Value Pair	Description
`stripnullsfromcsv=true | false`	When `true` , strip all nulls from input CSV events.
`avroschemaregistryurl= url`	Specifies the URL of the Apache Avro schema registry. This parameter is ignored when `kafkatype` is not set to **`avro`**. See also "Functions to Support JSON Objects" in *SAS Event Stream Processing: Publish/Subscribe API Reference*.
`keyfromkafkakey=true | false`	When set to **`true`**, specifies to copy the Kafka message key to the window key field. This field must be `type=string`. This key is ignored when `kafkatype` is not set to **`protobuf`**.
`kafkaconsumergroupid= string`	Specifies the group ID for this Kafka consumer. The default value is a randomly generated string. This parameter is not supported when hot failover is enabled.
`avroschemanoopcode= true | false`	When set to true, specifies to not include the event opcode in outbound Apache Avro schema and messages. This parameter is supported only when `kafkatype` is set to **`avro`**; it is otherwise ignored.
`offsetfieldname= name`	Specifies the name of an INT64 field in the Source window schema into which to copy the Kafka message offset. **Note:** This parameter is not supported when the `kafkatype` is set to **`binary`**.
`partitionfieldname= name`	Specifies the name of an INT32 field in the Source window schema into which to copy the Kafka message partition. **Note:** This parameter is not supported when the `kafkatype` is set to **`binary`**.
`avronestingseparator=character`	Specifies the character that separates nested Apache Avro fields in Source window fields. The default value is `'_'`.
`kafkaheaderfieldnames=list`	Specifies a comma-separated list of window schema field names. The contents of the specified fields are copied from headers in Kafka messages instead of the message payloads. Configured fields must be of type STRING or BLOB. For every configured field, the field value is copied from a header whose key matches the field name. If there are multiple such headers in a single message, then all values are copied to the window field and separated by commas. In this case, the field must be of type STRING and not BLOB. If a received message has no header matching a configured field name, then that field value is NULL. If a received message has a header matching a configured field name and the message payload also contains data that is

Key-Value Pair	Description
	intended for that field, then the header value takes precedence. By default, Kafka message headers are ignored. By default, Kafka message headers are unused.

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 65 *Required Parameters for the Kinesis Subscriber Connector*

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

Table 66 *Required Parameters for the Kinesis Publisher Connector*

Parameter	Description
`type`	Specifies to publish. Must be **`pub`**.
`kinesistype`	Specifies **`binary`**, **`csv`** , **`json`**, **`protobuf`**, **`avro`** , or **`opaquestring`**. When the `kinesistype` is **`opaquestring`**, the Source window schema is assumed to be `"index:int64,message:string"`.
`streamname`	Specifies the Amazon Kinesis Data Stream name.

Parameter	Description
awsregion	Specifies the Amazon AWS region.
sharditeratortype	Valid values are **trimhorizon**, **latest** , **atsequencenumber**, **aftersequencenumber**, or **attimestamp**.

Table 67 *Optional Parameters for the Kinesis Subscriber Connector*

Parameter	Description
collapse	Enables conversion of UPDATE_BLOCK events to make subscriber output publishable.
rmretdel	Specifies to remove all delete events from event blocks received by a subscriber that were introduced by a window retention policy.
dateformat	Specifies the format of DATE and STAMP fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (DATE) or microseconds (STAMP) since epoch. The dateformat parameter accepts any time format that is supported by the UNIX strftime function.
protofile	Specifies the **.proto** file that contains the Google Protocol Buffers message definition used to convert event blocks to protobuf messages. When you specify this parameter, you must also specify the protomsg parameter. This parameter is ignored when kinesistype is not set to **protobuf**.
protomsg	Specifies the name of a Google Protocol Buffers message in the **.proto** file that you specified with the protofile parameter. Event blocks are converted into this message. This parameter is ignored when kinesistype is not set to **protobuf**.
csvincludeschema	When kinesistype=csv , specifies when to prepend output CSV data with the window's serialized schema. Valid values are **never**, **once**, and **pereventblock**. The default value is **never**.
configfilesection	Specifies the name of the section in **/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config** (Linux) or **%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.config** (Windows) to parse for configuration parameters.
csvmsgperevent	For CSV, specifies to send one message per event. The default is one message per transactional event block or else one message per event.

Parameter	Description
csvmsgpereventblock	For CSV, specifies to send one message per event block. The default is one message per transactional event block or else one message per event.
doubleprecision	Specifies the number of fractional digits in the ASCII representation of a double. The default value is 6.
avroschemaregistryurl	Specifies the URL of the Apache Avro schema registry. This parameter is ignored when kinesistype is not set to **avro**. See also "Functions to Support Apache Avro Objects" in *SAS Event Stream Processing: Publish/Subscribe API Reference*.
avroschemadefinition	Specifies the path to a file that contains an Apache Avro schema definition in JSON format. This schema is copied to the schema registry that is configured in the avroschemaregistryurl parameter. The default value is an Apache Avro schema that represents the subscribed window schema plus an opcode field. This definition is also copied to the schema registry. This parameter is ignored when kinesistype is not set to **avro**.
avroschemaname	Specifies the name of an Apache Avro schema to copy from the schema registry that is configured in the avroschemaregistryurl parameter. The default value is an Apache Avro schema that represents the subscribed window schema plus an opcode field. This name is copied to the schema registry. This parameter is ignored when kinesistype is not set to **avro**.
awsaccesskeyid	Specifies the AWS access key ID credential. Required when awssecretkey and awssessiontoken are configured.
awssecretkey	Specifies the AWS secret key credential. Required when awsaccesskeyid and awssessiontoken are configured.
awssessiontoken	Specifies the AWS session token credential. This parameter is required when awsaccesskeyid and awssecretkey are configured.
partitionkeyrange	Specifies the number of partition keys used when writing output records. The default value is 1.
avroschemanoopcode	Specifies to not include the event opcode in outbound Apache Avro schema and messages. This parameter is supported only when kafkatype is set to **avro**; it is otherwise ignored.
stsassumerolean	Specifies the ARN used to perform an AWS Security Token Service (STS) Assume Role. Required when stsassumerolesessionname and stsassumeroleexternalid are configured.

Parameter	Description
stsassumerolesessionname	Specifies the session name used to perform an AWS Security Token Service (STS) Assume Role. Required when stsassumerolearn and stsassumeroleexternalid are configured.
stsassumeroleexternalid	Specifies the external ID used to perform an AWS Security Token Service (STS) Assume Role. Required when stsassumerolearn and stsassumerolesessionname are configured.

Table 68 *Optional Parameters for the Kinesis Publisher Connector*

Parameter	Description
transactional	When kinesistype=csv , sets the event block type to transactional. The default event block type is normal.
blocksize	When kinesistype=csv , specifies the number of events to include in a published event block. The default value is 1.
dateformat	Specifies the format of DATE and STAMP fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (DATE) or microseconds (STAMP) since epoch. The dateformat parameter accepts any time format that is supported by the UNIX strftime function.
ignorecsvparseerrors	When a field in an input CSV event cannot be parsed, this specifies to drop the event, log an error, and continue publishing.
protofile	Specifies the **.proto** file that contains the Google Protocol Buffers message definition used to convert event blocks to protobuf messages. When you specify this parameter, you must also specify the protomsg parameter. This parameter is ignored when kinesistype is not set to **protobuf**.
protomsg	Specifies the name of a Google Protocol Buffers message in the **.proto** file that you specified with the protofile parameter. Event blocks are converted into this message. This parameter is ignored when kinesistype is not set to **protobuf**.
configfilesection	Specifies the name of the section in **/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config** (Linux) or **%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.config** (Windows) to parse for configuration parameters.
csvfielddelimiter	Specifies the character delimiter for field data in input CSV events. The default delimiter is the , character.

Parameter	Description
noautogenfield	Specifies that input events are missing the key field that is autogenerated by the source window. **Note:** This parameter applies only when the format of input data is CSV.
publishwithupsert	Specifies to build events with opcode = Upsert instead of opcode = Insert.
addcsvopcode	Prepends an opcode and comma to write CSV events. The opcode is Insert unless publishwithupsert is enabled.
addcsvflags	Specifies the event type to insert into input CSV events (with a comma). Valid types are normal and partial-update.
maxevents	Specifies the maximum number of events to publish.
stripnullsfromcsv	Strip all nulls from input CSV events.
avroschemaregistryurl	Specifies the URL of the Apache Avro schema registry. This parameter is ignored when kinesistype is not set to **avro**. See also "Functions to Support Apache Avro Objects" in *SAS Event Stream Processing: Publish/Subscribe API Reference*.
awsaccesskeyid	Specifies the AWS access key ID credential. Required when awssecretkey and awssessiontoken are configured.
awssecretkey	Specifies the AWS secret key credential. Required when awsaccesskeyid and awssessiontoken are configured.
awssessiontoken	Specifies the AWS session token credential. This parameter is required when awsaccesskeyid and awssecretkey are 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 sharditeratortype is set to **atsequencenumber** or **aftersequencenumber**.
shardtimestamp	Specifies the timestamp to use when sharditeratortype is set to **attimestamp**. See the dateformat parameter description for the required format.
maxrecordspershard	Specifies the maximum number of records to read from the shard per GetRecords() call. The default value is to read all available records.

Parameter	Description
`stsassumerolean`	Specifies the ARN used to perform an AWS Security Token Service (STS) Assume Role. Required when `stsassumerolesessionname` and `stsassumeroleexternalid` are configured.
`stsassumerolesessionname`	Specifies the session name used to perform an AWS Security Token Service (STS) Assume Role. Required when `stsassumerolearn` and `stsassumeroleexternalid` are configured.
`stsassumeroleexternalid`	Specifies the external ID used to perform an AWS Security Token Service (STS) Assume Role. Required when `stsassumerolearn` and `stsassumerolesessionname` are configured.

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 ,...*

Subscriber Usage

Table 69 *Required Keys*

Key-Value Pair	Description
`type=sub`	Specifies a subscriber adapter.
`url=` *pubsubURL*	Specifies the standard URL in the form **`dfESP://host :port/ project/continuousquery /window`**. Append the following for subscribers: **`?snapshot=true`** | **`false`** . When **`?snapshot=true`**, the subscriber receives a collection of Insert events that are contained in the window at that point in time. The subscriber then receives a stream of events produced from the time of the snapshot onward. Those subsequent events can be Inserts, Updates, or Deletes. Append **`?collapse=true`** | **`false`** or **`?rmretdel=true`** | **`false`** for subscribers if needed.

Key-Value Pair	Description
`kinesistype= binary | csv | json | protobuf | avro`	Specifies the message format. For subscribers, the name of a string field in the subscribed window schema is also supported.
`streamname= name`	Specifies the Amazon Kinesis Data Stream name.
`awsregion= region`	Specifies the Amazon AWS region.
`partitionkey= key`	Specifies a string used to map output records to a specific shard. Configure `partitionkeyrange` to write to multiple shards.

Table 70 *Optional Keys*

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

Key-Value Pair	Description
`configfilesection=[ sectio n]`	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.
`restartonerror=true | false`	When true, specifies to restart the adapter if a fatal error is reported.
`transportconfigfile= file`	Specifies the publish/subscribe transport configuration file. For Solace, the default is **`./solace.cfg`**. For Tervela, the default is **`./client.config`**. For RabbitMQ, the default is **`./rabbitmq.cfg`**. For Kafka, the default is `./kafka.cfg`. **Note:** No transport configuration file is required for native transport.
`csvincludeschema=never | once | pereventblock`	When `kinesistype=csv` , specifies when to prepend output CSV data with the window's serialized schema. The default value is `never`.
`csvmsgperevent= true | false`	When true, for CSV, specifies to send one message per event. The default is one message per transactional event block or else one message per event.
`csvmsgpereventblock=true | false`	When true, for CSV, specifies to send one message per event block. The default is one message per transactional event block or else one message per event.
`doubleprecision= number`	Specifies the number of fractional digits in the ASCII representation of a double. The default value is 6.
`avroschemaregistryurl= url`	Specifies the URL of the Apache Avro schema registry. This parameter is ignored when `kinesistype` is not set to **`avro`**. See also "Functions to Support Apache Avro Objects" in *SAS Event Stream Processing: Publish/Subscribe API Reference*.
`awsaccesskeyid= id`	Specifies the AWS access key ID credential. This key is required when `awssecretkey` and `awssessiontoken` are configured.
`awssecretkey= key`	Specifies the AWS secret key credential. This key is required when `awsaccesskeyid` and `awssessiontoken` are configured.
`awssessiontoken= token`	Specifies the AWS session token credential. This key is required when `awsaccesskeyid` and `awssecretkey` are 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

Key-Value Pair	Description
	registry configured in the `avroschemaregistryurl` parameter. The default value is an Apache Avro schema that represents the subscribed window schema plus an opcode field. This schema is also copied to the schema registry. This parameter is ignored when `kinesistype` is not set to **avro**.
`avroschemaname=` *name*	Specifies the name of an Apache Avro schema to copy from the schema registry configured in the `avroschemaregistryurl` parameter. The default value is an Apache Avro schema that represents the subscribed window schema plus an opcode field. This name is copied to the schema registry. This parameter is ignored when `kinesistype` is not set to **avro**.
`partitionkeyrange=` *number*	Specifies the number of partition keys used when writing output records. The default value is 1.
`acroschemanoopcode= true | false`	When set to true, specifies to not include the event opcode in outbound Apache Avro schema and messages. This parameter is supported only when `kafkatype` is set to **avro**; it is otherwise ignored.
`stsassumerolean=`*arn*	Specifies the Amazon Resource Name (ARN) used to perform an Amazon Web Service (AWS) Security Token Service (STS) Assume Role. This key is required when `stsassumerolesessionname` and `stsassumeroleexternalid` are configured.
`stsassumerolesessionname=`*name*	Specifies the session name used to perform an AWS Security Token Service (STS) Assume Role. This key is required when `stsassumerolearn` and `stsassumeroleexternalid` are configured.
`stsassumeroleexternalid=`*id*	Specifies the external ID used to perform an AWS Security Token Service (STS) Assume Role. This key is required when `stsassumerolearn` and `stsassumerolesessionname` are configured.

Publisher Usage

Table 71 *Required Keys*

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

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

Table 72 *Optional Keys*

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

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 **/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config** (Linux) or **%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.config** (Windows) to parse for configuration parameters.
`restartonerror=true | false`	When **true** , specifies to restart the adapter if a fatal error is reported.
`transportconfigfile=` *file*	Specifies the publish/subscribe transport configuration file. For Solace, the default is **./solace.cfg**. For Tervela, the default is **./client.config**. For RabbitMQ, the default is **./rabbitmq.cfg**. For Kafka, the default is `./kafka.cfg`. **Note:** No transport configuration file is required for native transport.
`awsaccesskeyid=` *id*	Specifies the AWS access key ID credential. This key is required when `awssecretkey` and `awssessiontoken` are configured.
`awssecretkey=` *key*	Specifies the AWS secret key credential. This key is required when `awsaccesskeyid` and `awssessiontoken` are configured.
`awssessiontoken=` *token*	Specifies the AWS session token credential. This key is required when `awsaccesskeyid` and `awssecretkey` are configured.
`blocksize=` *size*	Sets the block size. The default value is 1.
`transactional=true | false`	When true, event blocks are transactional. By default, event blocks are normal.
`ignorecsvparseerrors=true | false`	When a field in an input CSV event cannot be parsed, this specifies to drop the event, log an error, and continue publishing.
`csvfielddelimiter=` *delimiter*	Specifies the character delimiter for field data in input CSV events. The default delimiter is the `,` character.
`noautogenfield=true | false`	When **true** , specifies that input events are missing the key field that is autogenerated by the Source window. **Note:** This parameter applies only when the format of input data is CSV.

Key-Value Pair	Description
`publishwithupsert=true | false`	When **`true`** , build events with `opcode = Upsert` instead of `Insert`.
`addcsvopcode=true | false`	When **`true`** , prepends an opcode and comma to write CSV events. The opcode is Insert unless `publishwithupsert` is **`true`**.
`addcsvflags=none | normal | partialupdate`	Specifies the event type to insert into input CSV events (with comma).
`maxevents=` *number*	Specifies the maximum number of events to publish.
`quiesceproject=true | false`	When **`true`** , quiesces the project after all events are injected into the Source window.
`stripnullsfromcsv=true | false`	When **`true`** , strip all nulls from input CSV events.
`avroschemaregistryurl=` *url*	Specifies the URL of the Apache Avro schema registry. This parameter is ignored when `kinesistype` is not set to **`avro`**. See also "Functions to Support Apache Avro Objects" in *SAS Event Stream Processing: Publish/Subscribe API Reference*.
`shardids=` *list*	Specifies a comma separated list of shard IDs to read from. The default value is to read from all shards in the stream.
`shardsequencenumber=` *number*	Specifies the sequence number to use when `sharditeratortype` is set to **`atsequencenumber`** or **`aftersequencenumber`**.
`shardtimestamp=` *timestamp*	Specifies the timestamp to use when `sharditeratortype` is set to **`attimestamp`**. See the `dateformat` parameter description for the required format.
`maxrecordspershard=` *number*	Specifies the maximum number of records to read from the shard per `GetRecords()` call. The default value is to read all available records.
`stsassumerolean=`*arn*	Specifies the Amazon Resource Name (ARN) used to perform an Amazon Web Service (AWS) Security Token Service (STS) Assume Role. This key is required when `stsassumerolesessionname` and `stsassumeroleexternalid` are configured.
`stsassumerolesessionname=`*name*	Specifies the session name used to perform an AWS Security Token Service (STS) Assume Role. This key is required when `stsassumerolearn` and `stsassumeroleexternalid` are configured.
`stsassumeroleexternalid=`*id*	Specifies the external ID used to perform an AWS Security Token Service (STS) Assume Role. This key is required when `stsassumerolearn` and `stsassumerolesessionname` are configured.

Using the SAS LASR Analytic Server Adapter

IMPORTANT This functionality is no longer supported or included in SAS Viya 3.5 revision 24w44. No updates to this functionality are available for earlier releases.

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 -md md5 -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* ,...

IMPORTANT The default values of Boolean parameters for this adapter are **`false`**. You cannot manually reset a Boolean parameter from **`true`** to **`false`** with `-C`. To reset the value, simply remove the parameter from the command line.

Table 73 *Required Keys*

Key-Value Pair	Description
`lasrurl=` *`url`*	Specifies the SAS LASR Analytic Server connection URL in the form " *host:port*"
`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><? rmretdel=true | false>". **Note:** Multiple URLs, separated by commas, are permitted.
`type=sub`	Specifies a subscriber 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 74 *Optional Keys*

Key-Value Pair	Description
`autocommit=` *number*	Specifies the number of observations to commit to the server. A value < 0 specifies the time in seconds between auto-commit operations. A value > 0 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.
`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.
`pubsublib= native | solace | tervela | rabbitmq | kafka`	Specifies the transport type. When you specify the `solace`, `tervela`, `rabbitmq`, or `kafka` transports instead of the default `native` transport, use the required client configuration files specified at "Using the Java Publish/Subscribe API" in *SAS Event Stream Processing: Publish/Subscribe API Reference* .
`pwdencrypted= true | false`	When true, the SAS LASR Analytic Server password is encrypted. The default value is false.
`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.
`nostrictschema= true | false`	When true, the adapter does not observe strict SAS LASR Analytic Server adapter schema. The default value is false, and strict schema is observed. **`javaadapters.config`** parameter name: `nostrictschema`
`stimeout=` *seconds*	Specifies the time out interval. The default value is **`300`**.
`authurl=` *url*	Specifies the URL for SAS LASR Analytic Server authentication services.
`restartonerror= true | false`	When true, restarts the adapter when a fatal error is reported. The default value is false. **`javaadapters.config`** parameter name: `restartonerror`
`tklasrkey=` *path*	Specifies the path to **`tklasrkey.sh`** for Linux or **`tklasrkey.bat`** for Microsoft Windows. By default, these files are located in **`$DFESP_HOME/bin`** .

Key-Value Pair	Description
`authusername= username`	Specifies the user name for SAS LASR Analytic Server authentication services.
`authpassword= password`	Specifies the password for SAS LASR Analytic Server authentication services.
`obstoanalyze= number`	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 `-s`, this option is ignored.
`blocksize= size`	Specifies the buffer size (number of observations) to be flushed to the server.
`dateformat= format`	Specifies the format of `DATE` and `STAMP` fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (`DATE`) or microseconds (`STAMP`) since epoch. The `dateformat` parameter accepts any time format that is supported by the UNIX `strftime` function.
`gdconfigfile= file`	Specifies the guaranteed delivery configuration file.
`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. The default value is false, and the adapter assumes a LASR table already exists. **`javaadapters.config`** parameter name: `newtable`
`schema= schema`	Specifies the explicit SAS LASR Analytic Server schema in the form *rowname1* :*sastype1*: *sasformat1*: *label1*,...*rownameN* :*sastypeN*: *sasformatN*: *labelN*. **Note:** When you omit the *sasformat*, no SAS format is applied to the row. When you omit the *label*, no label is applied to the row. If you previously specified `nostrictschema=true`, this key is ignored. **Note:** If the *schema* includes a comma, then you must enclose the entire string in escaped double quotation marks. You must also escape spaces, dollar sign, and parenthesis characters in the string. For example: `schema=\"ID:num,column_1:char\(5\):\$5.:column\ 1\"`
`transportconfigfile= file`	Specifies the full path to the publish/subscribe transport configuration file. The default file depends on the

Key-Value Pair	Description
	transport type specified with through the `pubsublib` parameter: For `solace` , the default file is **`solace.cfg`**. For `tervela` , the default file is **`client.config`**. For `rabbitmq` , the default file is **`rabbitmq.cfg`**. For `kafka` , the default file is **`kafka.cfg`**. **Note:** No transport configuration file is required for native transport.
`bufsize=` *mb*	Specifies the size (in megabytes) of the buffered input-output stream that is associated with the socket connection to the ESP server.

Publisher Usage:

dfesp_lasr_adapter -C *key1* =*val1*, *key2* =*val2*, *key3* =*val3* , ...

Table 75 *Required Keys*

Key-Value Pair	Description
`lasrurl=` *url*	Specifies the SAS LASR Analytic Server connection URL in the form " *host:port*"
`url=` *url*	Specifies the event stream processing standard URL in the following format: "dfESP:// *host:port* /*project*/ *contquery*/*window* ". **Note:** 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 76 *Optional Keys*

Key-Value Pair	Description
`configfilesection=` *name*	Specifies the name of the section in **`/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/javaadapters.config`** (Linux) or **`%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default`**

Key-Value Pair	Description
	\javaadapters.config (Windows) to parse for configuration parameters.
`fetchallcached=true | false`	When true, enables caching as data is fetched from a table. The default value is false, and caching is disabled. **javaadapters.config** parameter name: `fetchallcached`
`pubsublib= native | solace | tervela | rabbitmq | kafka`	Specifies the transport type. When you specify the `solace`, `tervela`, `rabbitmq`, or `kafka` transports instead of the default `native` transport, use the required client configuration files specified at "Using the Java Publish/Subscribe API" in *SAS Event Stream Processing: Publish/Subscribe API Reference* .
`pwdencrypted= true | false`	When true, the SAS LASR Analytic Server password is encrypted. By default, it is not.
`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.
`cachedaction=" action"`	Specifies the action to get cached results from the LASR table. For example, specify `cachedaction="fetch field1 / to=100 where field2=2".`
`nostrictschema= true | false`	When true, the adapter does not observe strict SAS LASR Analytic Server adapter schema. The default value is false, and strict schema is observed. **javaadapters.config** parameter name: `nostrictschema`
`stimeout= seconds`	Specifies the time out interval. The default value is **300**.
`authurl= url`	Specifies the URL for SAS LASR Analytic Server authentication services.
`restartonerror= true | false`	When true, restarts the adapter when a fatal error is reported. The default value is false. **javaadapters.config** parameter name: `restartonerror`
`maxevents= number`	Specifies the maximum number of events to publish.
`tklasrkey= path`	Specifies the path to **tklasrkey.sh** for Linux or **tklasrkey.bat** for Microsoft Windows. By default, these files are located in **$DFESP_HOME/bin** .
`authusername= username`	Specifies the user name for SAS LASR Analytic Server authentication services.

Key-Value Pair	Description
`authpassword=` *password*	Specifies the password for SAS LASR Analytic Server authentication services.
`blocksize=` *size*	Specifies the buffer size (number of observations) to be flushed to the server.
`quiesceproject=true | false`	When true, quiesces the project after all events are injected into the Source window. The default value is false. **`javaadapters.config`** parameter name: `quiesceproject`
`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.
`transactional=true | false`	When true, specifies that event blocks are transactional. The default value is false, and event blocks are normal. **`javaadapters.config`** parameter name:`transactional`
`gdconfigfile=` *file*	Specifies the guaranteed delivery configuration file.
`loglevel=severe | warning | info`	Specifies the application logging level. The default level is `warning` .
`action="` *action*`"`	Specifies the action to get cached results from the LASR table. For example, specify `action="fetch field1 / to=100 where field2=2"`.
`publishwithupsert=true | false`	When true, builds events with `opcode=Upsert`. The default value is false, and events are built with `opcode=Insert`. **`javaadapters.config`** parameter name: `publishwithupsert`
`transportconfigfile=` *file*	Specifies the full path to the publish/subscribe transport configuration file. The default file depends on the transport type specified with through the `pubsublib` parameter: For `solace` , the default file is **`solace.cfg`**. For `tervela` , the default file is **`client.config`**. For `rabbitmq` , the default file is **`rabbitmq.cfg`**. For `kafka` , the default file is **`kafka.cfg`**. **Note:** No transport configuration file is required for native transport.

Key-Value Pair	Description
`bufsize= mb`	Specifies the size (in megabytes) of the buffered input-output stream that is associated with the socket connection to the ESP server.

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 77 *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 https://sourceforge.net/projects/modrssim/.

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 78 *Required Parameters for the Modbus Publisher Connector*

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

Table 79 *Optional Parameters for the Modbus Publisher Connector*

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

Table 80 *Required Parameters for the Modbus Subscriber Connector*

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

Table 81 *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, ...*

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 82 *Required Keys*

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

Table 83 *Optional Keys*

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

Key-Value Pair	Description
	Note: No transport configuration file is required for native transport.
`tokenlocation=` *location*	Specifies the location of the file in the local filesystem that contains the OAuth token required for authentication by the publish/subscribe server.
`configfilesection=[` *section*`]`	Specifies the name of the section in **`/opt/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*`]`.

Publisher Usage

Table 84 *Required Keys*

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

Table 85 *Optional Keys*

Key-Value Pairs	Description
`restartonerror=true | false`	When `true` , restarts the adapter if a fatal error is reported.
`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

Key-Value Pairs	Description
	configuration files specified in the description of the C++ C_dfESPpubsubSetPubsubLib() API call.
`transportconfigfile= file`	Specifies the publish/subscribe transport configuration file. For `solace` , the default is **`./solace.cfg`**. For `tervela` , the default is **`./client.config`**. For `rabbitmq` , the default is **`./rabbitmq.cfg`**. For `kafka` , the default is **`./kafka.cfg`**. **Note:** No transport configuration file is required for native transport.
`tokenlocation= location`	Specifies the location of the file in the local filesystem that contains the OAuth token required for authentication by the publish/subscribe server.
`configfilesection=[ section]`	Specifies the name of the section in **`/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config`** (Linux) or **`%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.config`** (Windows) to parse for configuration parameters. Specify the value as `[configfilesection ]`.
`blocksize= size`	Specifies the event block size. The default value is 1.
`maxevents= maxevents`	Specifies the maximum number of events to publish.
`interval= seconds`	Specifies the Modbus polling interval, in seconds. The default is `10` .
`coils= coils`	Specifies the Modbus coils to read.
`inputs= inputs`	Specifies the Modbus inputs to read.
`inputRegisters= registers`	Specifies the Modbus input registers to read.
`holdingRegisters= hregisters`	Specifies the Modbus holding registers to read.

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 -md md5 -a -salt\
```

```
-pass pass:'espMQTTconnectorUsedByUser=mqttuserid '
```

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

Table 86 *Required Parameters for Subscriber MQTT Connectors*

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

Table 87 *Optional Parameters for Subscriber MQTT Connectors*

Parameter	Description
`mqttuserid`	Specifies the user name required to authenticate the connector's session with the MQTT server.
`mqttpassword`	Specifies the password associated with `mqttuserid`.
`mqttport`	Specifies the MQTT server port. The default is 1883.

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

Parameter	Description
	This parameter is ignored when `mqttmsgtype` is not set to **`protobuf`**.
`mqttssl`	Specifies to use TLS to connect to the MQTT broker. The default is "`false` ". In order to use TLS, the SAS Event Stream Processing encryption overlay must be installed.
`mqttsslcafile`	If `mqttssl=true` , specifies the path to a file containing the PEM encoded trusted CA certificate files. Either `mqttsslcafile` or `mqttsslcapath` must be specified.
`mqttsslcapath`	If `mqttssl=true` , specifies the path to a directory containing the PEM encoded trusted CA certificate files. See **`mosquitto.conf`** for more details about configuring this directory. Either `mqttsslcafile` or `mqttsslcapath` must be specified.
`mqttsslcertfile`	If `mqttssl=true` , specifies the path to a file containing the PEM encoded certificate file for this client. Both `mqttsslcertfile` and `mqttsslkeyfile` must be provided if one of them is.
`mqttsslkeyfile`	If `mqttssl=true` , specifies the path to a file containing the PEM encoded private key for this client. Both `mqttsslcertfile` and `mqttsslkeyfile` must be provided if one of them is.
`mqttsslpassword`	If `mqttssl=true` , and if key file is encrypted, specifies the password for decryption.
`csvincludeschema`	Specifies `never` , `once`, or `pereventblock`. The default value is " `never`". When `mqttmsgtype = CSV`, prepend output CSV with the window's serialized schema.
`configfilesection`	Specifies the name of the section in **`/opt/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`*` ]`.
`mqttpasswordencrypted`	Specifies that `mqttpassword` is encrypted.
`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.

Parameter	Description
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 88 *Required Parameters for Publisher MQTT Connectors*

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

Table 89 *Optional Parameters for Publisher MQTT Connectors*

Parameter	Description
mqttuserid	Specifies the user name required to authenticate the connector's session with the MQTT server.
mqttpassword	Specifies the password associated with mqttuserid.
mqttport	Specifies the MQTT server port. Default is 1883.
mqttacceptretainedmsg	Sets to true to accept to receive retained message. The default is "false ".

Parameter	Description
mqttcleansession	Set to true to instruct the MQTT Server to clean all messages and subscriptions on disconnect, false to instruct it to keep them. The default is "true ".
mqttkeepaliveinterval	Specifies the number of seconds after which the broker should send a PING message to the client if no other messages have been exchanged in that time. The default is 10.
publishwithupsert	Builds events with opcode=Upsert instead of Insert.
transactional	When mqttmsgtype=CSV , sets the event block type to transactional. The default event block type is normal.
blocksize	When mqttmsgtype=CSV , specifies the number of events to include in a published event block. The default value is 1.
ignorecsvparseerrors	Specifies that when a field in an input CSV event cannot be parsed, the event is dropped, an error is logged, and publishing continues.
csvfielddelimiter	Specifies the character delimiter for field data in input CSV events. The default delimiter is the , character.
noautogenfield	Specifies that input events are missing the key field that is automatically generated by the Source window. **Note:** This parameter applies only when the format of input data is CSV.
dateformat	Specifies the format of DATE and STAMP fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (DATE) or microseconds (STAMP) since epoch. The dateformat parameter accepts any time format that is supported by the UNIX strftime function.
protofile	Specifies the **.proto** file that contains the Google Protocol Buffers message definition used to convert event blocks to protobuf messages. When you specify this parameter, you must also specify the protomsg parameter. This parameter is ignored when mqttmsgtype is not set to **protobuf**.
protomsg	Specifies the name of a Google Protocol Buffers message in the **.proto** file that you specified with the protofile parameter. Event blocks are converted into this message. This parameter is ignored when mqttmsgtype is not set to **protobuf**.

Parameter	Description
mqttssl	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.
mqttsslcafile	If mqttssl=true , specifies the path to a file containing the PEM encoded trusted CA certificate files. Either mqttsslcafile or mqttsslcapath must be specified.
mqttsslcapath	If mqttssl=true , specifies the path to a directory containing the PEM encoded trusted CA certificate files. See **mosquitto.conf** for more details about configuring this directory. Either mqttsslcafile or mqttsslcapath must be specified.
mqttsslcertfile	If mqttssl=true , specifies the path to a file containing the PEM encoded certificate file for this client. Both mqttsslcertfile and mqttsslkeyfile must be provided if one of them is.
mqttsslkeyfile	If mqttssl=true , specifies the path to a file containing the PEM encoded private key for this client. Both mqttsslcertfile and mqttsslkeyfile must be provided if one of them is.
mqttsslpassword	If mqttssl=true , and if key file is encrypted, specifies the password for decryption.
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*].
mqttpasswordencrypted	Specifies that mqttpassword is encrypted.
maxevents	Specifies the maximum number of events to publish.
addcsvopcode	Prepends an opcode and comma to input CSV events. The opcode is Insert unless publishwithupsert is enabled.
addcsvflags	Specifies the event type to insert into input CSV events (with a comma). Valid values are normal and partialupdate .
stripnullsfromcsv	Strip all nulls from input CSV events.

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: The MQTT adapter is not supported on Windows systems.

Subscriber Usage

Table 90 *Required Keys*

Key-Value Pair	Description
`type=sub`	Specifies a subscriber adapter.
`url=` *pubsubURL*	Specifies the standard URL in the form **`dfESP://`** ***`host :port/ project/continuousquery /window`***. Append `?snapshot=true |false` for subscribers. Append`?collapse=true | false` or `?rmretdel=true | false` or both for subscribers if needed.
`mqtthost=` *hostname*	Specifies the MQTT server host name.
`mqtttopic=` *topic*	Specifies the MQTT topic. An MQTT topic is an UTF-8 string that the broker uses to filter messages for each connected client. The topic consists of one or more topic levels. Each topic level is separated by a forward slash (topic level separator). If the configured string matches the name of a field in the subscribed window schema and that field is a string field, then the contents of that field are used as a dynamic topic to publish events to.
`mqttclientid=` *string*	Specifies the string to use as the MQTT Client ID. If NULL, a random client ID is generated. If NULL, `mqttdonotcleansession` must be `false`. Must be unique among all clients connected to the MQTT server.

Key-Value Pair	Description
`mqttqos=0 | 1 | 2`	Specifies the requested Quality of Service.
`mqttmsgtype=binary | csv | json | protobuf | opaquestringfield`	Specifies the message format or the name of a string field in the subscribed window schema.

Table 91 *Optional Keys*

Key-Value Pair	Description
`mqttuserid=` *username*	Specifies the user name required to authenticate the session with the MQTT server.
`mqttpassword=` *password*	Specifies the password associated with `mqttuserid`.
`mqttport=` *port*	Specifies the MQTT server port. The default is 1883.
`mqttdonotcleansession=true | false`	When `true` , instructs the MQTT Server to keep all messages and subscriptions on disconnect (instead of cleaning them). The default is "`false` ".
`mqttkeepaliveinterval=` *seconds*	Specifies the number of seconds after which the broker should send a PING message to the client if no other messages have been exchanged in that time. The default is 10 seconds.
`mqttssl=true | false`	When `true` , uses 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.
`mqttsslcafile=` *path*	When `mqttssl=true` , specifies the path to a file containing the PEM encoded trusted CA certificate files. Either **`mqttsslcafile`** or **`mqttsslcapath`** must be specified.
`mqttsslcapath=` *path*	When `mqttssl=true` , specifies the path to a file containing the PEM encoded trusted CA certificate files. Either **`mqttsslcafile`** or **`mqttsslcapath`** must be specified. See mosquitto.conf for more information about configuring this directory.
`mqttsslcertfile=` *path*	When `mqttssl=true` , specifies the path to a file containing the encoded certificate file for this client. Both **`mqttsslcertfile`** and **`mqttsslkeyfile`** must be provided if one of them is.

Key-Value Pair	Description
mqttsslkeyfile= *path*	When mqttssl=true , specifies the path to a file containing the encoded private key for this client. Both **mqttsslcertfile** and **mqttsslkeyfile** must be provided if one of them is.
mqttsslpassword= *password*	When mqttssl=true and **mqttsslkeyfile** is encrypted, specifies the password for decryption.
dateformat= *format*	Specifies the format of DATE and STAMP fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (DATE) or microseconds (STAMP) since epoch. The dateformat parameter accepts any time format that is supported by the UNIX strftime function.
protofile= *file*	Specifies the **.proto** file to be used for Google protocol buffer support. This key is ignored when mqttmsgtype is not set to **protobuf**.
protomsg= *message*	Specifies the message itself in the **.proto** file that is specified by the protofile parameter. This key is ignored when mqttmsgtype is not set to **protobuf**.
gdconfig= *file*	Specifies the guaranteed delivery configuration file.
transport=native | solace | tervela | rabbitmq | kafka	Specifies the transport type. If you specify solace, tervela , rabbitmq, or kafka transports instead of the default native transport, then use the required client configuration files specified in the description of the C++ C_dfESPpubsubSetPubsubLib() API call.
loglevel=trace | debug | info | warn | error | fatal | off	Sets the logging level for the adapter. This is the same range of logging levels that you can set in the C_dfESPpubsubInit() publish/subscribe API call and in the engine initialize() call. The default level is warn.
logconfigfile= *file*	Specifies the log configuration file.
tokenlocation= *location*	Specifies the location of the file in the local file system that contains the OAuth token that is required for authentication by the publish/subscribe server.
configfilesection=[*section*]	Specifies the name of the section in **/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config** (Linux) or **%ProgramData%\SAS\Viya**

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

Publisher Usage

Table 92 *Required Keys*

Key-Value Pair	Description
`type=pub`	Specifies a publisher adapter.
`url=` *pubsubURL*	Specifies the standard URL in the form **`dfESP://`** ***`host :port/ project/continuousquery /window`***.
`mqtthost=` *hostname*	Specifies the MQTT server host name.
`mqtttopic=` *topic*	Specifies the MQTT topic. An MQTT topic is an UTF-8 string that the broker uses to filter messages for each connected client. The topic consists of one or more topic levels. Each topic level is separated by a forward slash (topic level separator).
`mqttclientid=string`	Specifies the string to use as the MQTT Client ID. If NULL, a random client ID is generated. If NULL, `mqttdonotcleansession` must be `false`. Must be unique among all clients connected to the MQTT server.
`mqttqos=0 | 1 | 2`	Specifies the requested Quality of Service.
`mqttmsgtype=binary | csv | json | protobuf |` *opaquestring*	Specifies the message format. For *opaquestring*, when the Source window schema has three fields, it is assumed to be `"index:int64,topic:string,message:string"`. When the Source window schema has two fields, it is assumed to be `"index:int64,message:string"`.

Table 93 *Optional Keys*

Key-Value Pair	Description
`mqttuserid=` *username*	Specifies the user name required to authenticate the session with the MQTT server.
`mqttpassword=` *password*	Specifies the password associated with `mqttuserid`.
`mqttport=` *port*	Specifies the MQTT server port. The default is 1883.

Key-Value Pair	Description
`mqttdonotcleansession=true | false`	When `true` , instructs the MQTT Server to keep all messages and subscriptions on disconnect (instead of cleaning them). The default is "`false` ".
`mqttkeepaliveinterval=` *seconds*	Specifies the number of seconds after which the broker should send a PING message to the client if no other messages have been exchanged in that time. The default is 10 seconds.
`mqttssl=true | false`	When `true` , uses TLS to connect to the MQTT broker. The default is " `false`". In order to use TLS, the SAS Event Stream Processing encryption overlay must be installed.
`mqttsslcafile=` *path*	When `mqttssl=true` , specifies the path to a file containing the PEM encoded trusted CA certificate files. Either **`mqttsslcafile`** or **`mqttsslcapath`** must be specified.
`mqttsslcapath=` *path*	When `mqttssl=true` , specifies the path to a file containing the PEM encoded trusted CA certificate files. Either **`mqttsslcafile`** or **`mqttsslcapath`** must be specified. See mosquitto.conf for more information about configuring this directory.
`mqttsslcertfile=` *path*	When `mqttssl=true` , specifies the path to a file containing the encoded certificate file for this client. Both **`mqttsslcertfile`** and **`mqttsslkeyfile`** must be provided if one of them is.
`mqttsslkeyfile=` *path*	When `mqttssl=true` , specifies the path to a file containing the encoded private key for this client. Both **`mqttsslcertfile`** and **`mqttsslkeyfile`** must be provided if one of them is.
`mqttsslpassword=` *password*	When `mqttssl=true` and **`mqttsslkeyfile`** is encrypted, specifies the password for decryption.
`dateformat=` *format*	Specifies the format of `DATE` and `STAMP` fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (`DATE`) or microseconds (`STAMP`) since epoch. The `dateformat` parameter accepts any time format that is supported by the UNIX `strftime` function.
`protofile=` *file*	Specifies the **`.proto`** file to be used for Google protocol buffer support.

Key-Value Pair	Description
	This key is ignored when `mqttmsgtype` is not set to **protobuf**.
`protomsg=` *message*	Specifies the message itself in the **.proto** file that is specified by the protofile parameter. This key is ignored when `mqttmsgtype` is not set to **protobuf**.
`gdconfig=` *file*	Specifies the guaranteed delivery configuration file.
`transport=native | solace | tervela | rabbitmq | kafka`	Specifies the transport type. If you specify `solace`, `tervela` , `rabbitmq`, or `kafka` transports instead of the default `native` transport, then use the required client configuration files specified in the description of the C++ C_dfESPpubsubSetPubsubLib() API call.
`loglevel=trace | debug | info | warn | error | fatal | off`	Sets the logging level for the adapter. This is the same range of logging levels that you can set in the `C_dfESPpubsubInit()` publish/subscribe API call and in the engine `initialize()` call. The default level is `warn`.
`logconfigfile=` *file*	Specifies the log configuration file.
`tokenlocation=` *location*	Specifies the location of the file in the local file system that contains the OAuth token that is required for authentication by the publish/subscribe server.
`configfilesection=[` *section*`]`	Specifies the name of the section in **/opt/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*`]`.
`mqttpasswordencrypted=true | false`	When `true` , `mqttpassword` is encrypted.
`restartonerror=true | false`	When `true` , specifies to restart the adapter if a fatal error is reported.
`transportconfigfile=` *file*	Specifies the publish/subscribe transport configuration file. For `solace` , the default is **./solace.cfg**. For `tervela` , the default is **./client.config**. For `rabbitmq` , the default is **./rabbitmq.cfg**. For `kafka` , the default is **./kafka.cfg**.

Key-Value Pair	Description
	Note: No transport configuration file is required for native transport.
`mqttacceptretainedmsg= true | false`	When `true` , enables receiving of retained messages. The default is " `false`".
`blocksize= size`	Sets the block size. The default value is 1.
`transactional=true | false`	When `true` , event blocks are transactional. By default, event blocks are normal.
`ignorecsvparseerrors=true | false`	Specifies that when a field in an input CSV event cannot be parsed, the event is dropped, an error is logged, and publishing continues.
`csvfielddelimiter= delimeter`	Specifies the character delimiter for field data in input CSV events. The default delimiter is the , character.
`noautogenfield=true | false`	When `true` , specifies that input events are missing the key field that is autogenerated by the Source window. **Note:** This parameter applies only when the format of input data is CSV.
`publishwithupsert=true | false`	When `true` , build events with opcode = Upsert instead of Insert.
`addcsvopcode= true | false`	When `true` , prepends an opcode and comma to write CSV events. The opcode is Insert unless `publishwithupsert` is `true` .
`addcsvflags=none | normal | partialupdate`	Specifies the event type to Insert into input CSV events (with comma).
`maxevents= number`	Specifies the maximum number of events to publish.
`quiesceproject=true | false`	When `true` , quiesces the project after all events are injected into the Source window.
`stripnullsfromcsv=true | false`	When `true` , strip all nulls from input CSV events.

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

IMPORTANT This functionality is no longer supported or included in SAS Viya 3.5 revision 24w44. No updates to this functionality are available for earlier releases.

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 94 *Required Parameters of the Nurego Connector*

Parameter	Description
`type`	Specifies to subscribe. Must be "`sub`".
`snapshot`	Specifies whether to send snapshot data. When `true`, the subscriber receives a collection of Insert events that are contained in the window at that point in time. The subscriber then receives a stream of events produced from the time of the snapshot onward. Those subsequent events can be Inserts, Updates, or Deletes.
`serviceurl`	Specifies the target Nurego REST service URL.

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

Table 95 *Optional Parameters of the Nurego Connector*

Parameter	Description
`certpassword`	Specifies the password associated with the client certificate that is configured in certificate.
`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
`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`*` ]`.

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

Overview

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 96 *Supported Mappings for a Subscriber:*

ESP type	OPC-UA type
int32	INT32
	UINT32
int64	INT64
	UINT64
double	DOUBLE
array(dbl)	XVTYPE
string	STRING

ESP type	OPC-UA type
date	DATETIME
stamp	DATETIME
money	Not supported

Table 97 *Supported Mappings for a Publisher:*

ESP type	OPC-UA type
int32	INT32
	UINT32
	INT16
	UINT16
	BOOLEAN
int64	INT64
	UINT64 **Note:** This type is supported only when the value falls within the range of (0 - (2^32) - 1). SAS Event Stream Processing supports only signed integers.
	INT32 **Note:** This type is supported only when the value falls within the range of (0 - (2^16) – 1). SAS Event Stream Processing supports only signed integers.
	UINT32
	INT16
	UINT16
	BOOLEAN
	INTEGERID
double	DOUBLE
	FLOAT
	BOOLEAN

ESP type	OPC-UA type
array(dbl)	XVTYPE
	array of DOUBLE
	array of FLOAT
	array of BOOLEAN
array(i32)	array of INT32
	array of UINT32
	array of INT16
	array of UINT16
	array of BOOLEAN
array(i64)	array of INT32
	array of INT64
	array of UINT64
	array of UINT32
	array of INT16
	array of UINT16
	array of BOOLEAN
string	STRING
	GUID
	array of GUID
date	DATETIME
	UTCTIME
stamp	DATETIME
	UTCTIME

ESP type	OPC-UA type
money	Not supported

SAS Event Stream Processing also supports OPC-UA abstract data types. For publishers, the mapping of an abstract data type to an ESP data type is performed using the OPC-UA subtype that is received when a value to be written to the Source window is received. The current mappings are valid but they use the subtype of the value instead of the base type of the node. Whenever there is a type mismatch, the publisher logs a warning and writes a null to the target Source window field instead of logging an error and quitting.

Table 98 *Required Parameters for OPC-UA Connectors*

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

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 when no other field is appropriate.

Subscriber Usage

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 99 *Optional Parameters for Subscriber OPC-UA Connectors*

Parameter	Description
`opcuanamespaceuri`	Specifies the OPC-UA server namespace URI. The default is the namespace at `index=0` .
`opcuausername`	Specifies the OPC-UA user name. By default, there is none.
`opcuapassword`	Specifies the OPC-UA password. By default, there is none.

Parameter	Description
	Note: When required, you can configure this parameter with an encrypted password. You can encrypt a version of the password with OpenSSL, which must be installed on your system. When 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 'opcpwd' | openssl enc -e -aes-256-cbc -md md5\ -a -salt -pass pass:'espOPCUAconnectorUsedByUser=opcuid'` Then copy the encrypted password into your `opcuapassword` parameter and enable the `opcuapasswordencrypted` parameter.
`opcuapasswordencrypted`	Specifies that `opcuapassword` is encrypted.
`opcuasecuritymode`	Specifies the OPC-UA message security. Valid values are **`none`**, **`sign`**, and **`signandencrypt`**. The default value is **`signandencrypt`**.
`opcuasecuritypolicy`	Specifies an OPC-UA security policy. Valid values are **`basic128rsa15`**, **`basic256`**, **`basic256sha256`**, and **`aes128sha256rsaoaep`**. By default, there is no security policy. **Note:** In order to use this parameter, you must configure `opcuacert` and `opcuakey`. **Note:** This parameter is ignored when you connect to a non-SSL OPC-UA endpoint.
`opcuanodeids`	Specifies a comma-separated list of Node IDs to map to window schema fields, in the form `<identifier type>_<identifier>`. The list size must be equal to the number of fields in the subscribed window schema. Window field names are in Node ID form by default.
`configfilesection`	Specifies the name of the section in **`/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config`** (Linux) or **`%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.config`** (Windows) to

Parameter	Description
	parse for configuration parameters. Specify the value as `[configfilesection ]`.
`opcuanodeiddelimiter`	Specifies the character used to delimit fields in the `opcuanodeids` parameter. The default value is ','.
`opcuaapplicationuri`	Specifies a globally unique identifier for the client instance.
`opcuacert`	Specifies the path to the valid certificate file (DER format).
`opcuakey`	Specifies the path to the valid key file (DER format).

Publisher Usage

The publisher publishes an event when the value of the Value attribute of an OPC-UA node in the Source window schema changes. When 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 publish node attributes other than Value. To do this, append to the node ID:

- `._value` – to get the value. This is the default behavior if no attribute is specified.
- `._serverTimestamp` – to get the `serverTimestamp` attribute. ESP type: STAMP.
- `._sourceTimestamp` – to get the `sourceTimestamp` attribute. ESP type: STAMP.
- `._status` – to get the `status` attribute. ESP type: INT32.

For example, specifying `s_Demo.Dynamic.Scalar.String._serverTimestamp` as a node ID publishes the server timestamp of the node `s_Demo.Dynamic.Scalar.String`. This works both when IDs are supplied by the `opcuanodeids` parameter and when they are supplied by the window field names.

You can 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.

When either `opcuacert` or `opcuakey` parameters are supplied, secure mode is enabled. When secure mode is enabled, the parameters `opcuacert`, `opcuakey` and `opcuaapplicationuri` must be supplied. The message security mode is switched to SignAndEncrypt. Before proceeding, ensure that the server has an endpoint to support this mode.

The rules for valid certificate and key files are strict. First, both of these files must be in DER format. The key must be generated using RSA and cannot be password protected. The certificate file must be generated according to the x509 format. Critically, a URI must be provided In the Subject Alternative Names of the certificate. This URI must match the value of the `opcuaapplicationuri` parameter. Given this strictness, it might be useful to use a sample OpenSSL configuration file to generate the certificate and key.

When your server has a trust list, make sure the client's certificate has been added to the trust list.

Table 100 *Optional Parameters for Publisher OPC-UA Connectors*

Parameter	Description
`opcuanamespaceuri`	Specifies the OPC-UA server namespace URI. The default is the namespace at `index=0` .
`opcuausername`	Specifies the OPC-UA user name. By default, there is none.

`opcuapassword`	Specifies the OPC-UA password. By default, there is none.
	Note: When required, you can configure this parameter with an encrypted password. You can encrypt a version of the password with OpenSSL, which must be installed on your system. When 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 'opcpwd' | openssl enc -e -aes-256-cbc -md md5\`
	`-a -salt -pass pass:'espOPCUAconnectorUsedByUser=opcuid'`
	Then copy the encrypted password into your `opcuapassword` parameter and enable the `opcuapasswordencrypted` parameter.

`opcuapasswordencrypted`	Specifies that `opcuapassword` is encrypted.
`opcuasecuritymode`	Specifies the OPC-UA message security. Valid values are **`none`**, **`sign`**, and **`signandencrypt`**. The default value is **`signandencrypt`**.
`opcuasecuritypolicy`	Specifies an OPC-UA security policy. Valid values are **`basic128rsa15`**, **`basic256`**, **`basic256sha256`**, and **`aes128sha256rsaoaep`**. By default, there is no security policy. **Note:** In order to use this parameter, you must configure `opcuacert` and `opcuakey`. **Note:** This parameter is ignored when you connect to a non-SSL OPC-UA endpoint.
`opcuanodeids`	Specifies a comma-separated list of Node IDs to map to window schema fields, in the form `<identifier type>_<identifier>`. The list size must be equal to the number of fields in the subscribed window schema - 1. Window field names are in Node ID form by default.
`publishinterval`	Specifies the interval (in seconds) at which all current Node ID values are published into the Source window regardless of whether the values have changed. By default, values are published when they change.
`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 **`/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`*` ]`.
`publishwithupsert`	Builds events with `opcode=Upsert` instead of `Insert`.
`maxevents`	Specifies the maximum number of events to publish.
`opcuanodeiddelimiter`	Specifies the character used to delimit fields in the `opcuanodeids` parameter. The default value is ','.
`opcuaapplicationuri`	Specifies a globally unique identifier for the client instance.
`opcuacert`	Specifies the path to the valid certificate file (DER format).
`opcuakey`	Specifies the path to the valid key file (DER format).

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, ...*

Subscriber Usage

Table 101 *Required Keys*

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

Key-Value Pair	Description
	Append `?snapshot=true |false` for subscribers.
`opcuaendpoint= endpoint`	Specifies the OPC-UA server endpoint, without the `opc.tcp://` prefix.

Table 102 *Optional Keys*

Key-Value Pair	Description
`opcuanamespaceuri= uri`	Specifies the OPC-UA server namespace URI. The default is the namespace at `index=0` .
`opcuausername= username`	Specifies OPC-UA user name. By default, there is none.
`opcuapassword= password`	Specifies the OPC-UA password. By default, there is none. **Note:** When required, you can configure this parameter with an encrypted password. The encrypted version of the password can be generated using OpenSSL, which must be installed on your system. When 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 'opcpwd' | openssl enc -e -aes-256-cbc -md md5\ -a -salt -pass pass:'espOPCUAconnectorUsedByUser=opcuid'` Then copy the encrypted password into your `opcuapassword` parameter and enable the `opcuapasswordencrypted` parameter.
`opcuapasswordencrypted=true | false`	When **`true`**, `opcuapassword` is encrypted.
`opcuasecuritymode= none | sign | signandencrypt`	Specifies the OPC-UA message security. The default value is **`signandencrypt`**.
`opcuasecuritypolicy = basic128rsa15 | basic256 | basic256sha256 | aes128sha256rsaoaep`	Specifies an OPC-UA security policy. By default there is no security policy. **Note:** In order to use this key, you must configure `opcuacert` and `opcuakey`. **Note:** This key is ignored when you connect to a non-SSL OPC-UA endpoint.
`opcuanodeids= list`	Specifies a comma–separated list of Node IDs to map to window schema fields, in the form `<identifier type>_<identifier>`. The default assumes that window field names are in Node ID form.

Key-Value Pair	Description
`gdconfig= file`	Specifies the guaranteed delivery configuration file.
`transport=native | solace | rabbitmq | kafka`	Specifies the transport type. If you specify `solace`, `rabbitmq`, or `kafka` transports instead of the default `native` transport, then use the required client configuration files specified in the description of the C++ C_dfESPpubsubSetPubsubLib() API call.
`loglevel=trace | debug | info | warn | error | fatal | off`	Sets the logging level for the adapter. The default level is `warn`.
`logconfigfile= file`	Specifies the log configuration file.
`tokenlocation= location`	Specifies the location of the file in the local file system that contains the OAuth token that is required for authentication by the publish/subscribe server.
`configfilesection=[ section]`	Specifies the name of the section in **/opt/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 ]`.
`restartonerror=true | false`	When `true` , specifies to restart the adapter if a fatal error is reported.
`transportconfigfile= file`	Specifies the publish/subscribe transport configuration file. For `solace` , the default is **./solace.cfg**. For `rabbitmq` , the default is **./rabbitmq.cfg**. For `kafka` , the default is **./kafka.cfg**. **Note:** No transport configuration file is required for native transport.
`opcuanodeiddelimiter= character`	Specifies the character used to delimit fields in the *opcuanodeids* parameter. The default value is ','.
`opcuaapplicationuri=id`	Specifies a globally unique identifier for the client instance.
`opcuacert=path`	Specifies the path to the valid certificate file (DER format).
`opcuakey=key`	Specifies the path to the valid key file (DER format).

Publisher Usage

Table 103 *Required Keys*

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

Table 104 *Optional Keys*

Key-Value Pair	Description
`opcuanamespaceuri=` *uri*	Specifies the OPC-UA server namespace URI. The default is the namespace at `index=0` .
`opcuausername=` *username*	Specifies OPC-UA user name. By default, there is none.

`opcuapassword=` *password*	Specifies the OPC-UA password. By default, there is none.
	Note: When required, you can configure this parameter with an encrypted password. The encrypted version of the password can be generated using OpenSSL, which must be installed on your system. When 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 '`*opcpwd*`' | openssl enc -e -aes-256-cbc -md md5\`
	`-a -salt -pass pass:'espOPCUAconnectorUsedByUser=`*opcuid*`'`
	Then copy the encrypted password into your `opcuapassword` parameter and enable the `opcuapasswordencrypted` parameter.

`opcuasecuritymode= none | sign | signandencrypt`	Specifies the OPC-UA message security. The default value is **`signandencrypt`**.
`opcuasecuritypolicy = basic128rsa15 | basic256 | basic256sha256 | aes128sha256rsaoaep`	Specifies an OPC-UA security policy. By default, there is no security policy. **Note:** In order to use this key, you must configure `opcuacert` and `opcuakey`.

Key-Value Pair	Description
	Note: This key is ignored when you connect to a non-SSL OPC-UA endpoint.
`opcuanodeids= list`	Specifies a comma–separated list of Node IDs to map to window schema fields, in the form `<identifier type>_<identifier>`. The default assumes that window field names are in Node ID form.
`gdconfig= file`	Specifies the guaranteed delivery configuration file.
`transport=native | solace | rabbitmq | kafka`	Specifies the transport type. If you specify `solace`, `rabbitmq`, or `kafka` transports instead of the default `native` transport, then use the required client configuration files specified in the description of the C++ `C_dfESPpubsubSetPubsubLib()` API call.
`loglevel=trace | debug | info | warn | error | fatal | off`	Sets the logging level for the adapter. This is the same range of logging levels that you can set in the `C_dfESPpubsubInit()` publish/subscribe API call and in the engine `initialize()` call. The default level is `warn`.
`logconfigfile= file`	Specifies the log configuration file.
`tokenlocation= location`	Specifies the location of the file in the local file system that contains the OAuth token that is required for authentication by the publish/subscribe server.
`configfilesection=[ section]`	Specifies the name of the section in **`/opt/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 ]`.
`restartonerror=true | false`	When `true` , specifies to restart the adapter if a fatal error is reported.
`transportconfigfile= file`	Specifies the publish/subscribe transport configuration file. For `solace` , the default is **`./solace.cfg`**. For `rabbitmq` , the default is **`./rabbitmq.cfg`**. For `kafka` , the default is **`./kafka.cfg`**. **Note:** No transport configuration file is required for native transport.
`opcuanodeiddelimiter= character`	Specifies the character used to delimit fields in the *opcuanodeids* parameter. The default value is ','.

Key-Value Pair	Description
`blocksize=` *size*	Sets the block size. The default value is 1.
`transactional=true | false`	When `true` , event blocks are transactional. By default, event blocks are normal.
`publishwithupsert=true | false`	When `true` , build events with opcode = Upsert instead of Insert.
`publishinterval=` *seconds*	Specifies the interval (in seconds) at which all current Node ID values are published into the Source window regardless of whether the values have changed. By default, values are published when they change.
`maxevents=` *number*	Specifies the maximum number of events to publish.
`quiesceproject=true | false`	When `true` , quiesces the project after all events are injected into the Source window.
`opcuaapplicationuri=`*id*	Specifies a globally unique identifier for the client instance.
`opcuacert=`*path*	Specifies the path to the valid certificate file (DER format).
`opcuakey=`*key*	Specifies the path to the valid key file (DER format).

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 OPC-DA Publisher Adapter

OPC Data Access (OPC DA) is a machine-to-machine communication protocol for industrial automation. It is a precursor to OPC Unified Architecture (OPC UA) and it is used for similar purposes as that protocol in industries that use OPC UA.

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

The adapter does the following tasks:

- monitors a list of configured tag IDs
- receives changed values asynchronously

- builds a unique ESP event for each received value
- publishes that event into an ESP source window with a schema conforming to the type of tag values to be received

The supported window schemas are as follows:

- index*:int64,sensorname:string,sensorvalue:int32,timestamp:stamp
- index*:int64,sensorname:string,sensorvalue:int64,timestamp:stamp
- index*:int64,sensorname:string,sensorvalue:double,timestamp:stamp
- index*:int64,sensorname:string,sensorvalue:date,timestamp:stamp
- index*:int64,sensorname:string,sensorvalue:stamp,timestamp:stamp
- index*:int64,sensorname:string,sensorvalue:string,timestamp:stamp
- index*:int64,sensorname:string,sensorvalue:rstring,timestamp:stamp
- index*:int64,sensorname:string,sensorvalue:array(i32),timestamp:stamp
- index*:int64,sensorname:string,sensorvalue:array(i64),timestamp:stamp
- index*:int64,sensorname:string,sensorvalue:array(dbl),timestamp:stamp

The types listed for the `sensorvalue` field support the following Java representations of OPCDA types:

Table 105 *Supported Data Types for Sensor Values*

ESP type	Java Types
int32	Integer, Short, Byte, Boolean
int64	Long, Integer, Short, Byte, Boolean
double	Double, Float, Integer, Short, Byte
date	Date, Instant
stamp	Date, Instant
string	String, Character
rstring	String, Character
array(i32)	Integer[], Short[], Byte[], Boolean[]
array(i64)	Long[], Integer[], Short[], Byte[], Boolean[]
array(dbl)	Double[], Float[], Integer[], Short[], Byte[]

The tag ID list used by a running instance of the adapter must contain only tags that match the `sensorvalue` field type in the schema of the Source window to which the adapter is publishing.

To read tags of a different type, you must run an additional instance of the adapter and publish into a different Source window.

The tag ID list can be contained in a file on the local file system. The full path to that file is configured using the adapter optional `tagsfile` configuration parameter. The file format is one tag ID per line.

Alternatively, the tag ID list can be built at run-time by answering **`Y`** to the `Browse available tags ... ?` prompt issued by the adapter at start-up. This interface only shows browsed tags that match the `sensorvalue` field type of the connected window. Any tags added to the list at this time are also added to the file-based list, if a file is configured.

Syntax:

dfesp_opcda_adapter

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

Table 106 *Required Keys*

Key	Description
`type=pub`	Specifies a publisher adapter.
`url=`*`pubsubURL`*	Specifies the standard URL in the form `"dfESP://`*`host:port/project/continuousquery/window`*`"`.
`opcdahostport=`*`host:port`*	Specifies the OPC-DA server host name and port in the form "*host:port*".
`opcdaservername=`*`servername`*	Specifies the OPC-DA server name.
`opcdausername=`*`username`*	Specifies the OPC-DA user name.
`opcdaserverntdomain=`*`domainname`*	Specifies the OPC-DA server Windows NT domain name.
`opcdapassword=`*`password`*	Specifies the OPC-DA password.
`opcdaserverclsid=`*`clsid`*	Specifies the OPC-DA server COM CLSID.

Table 107 *Optional Keys*

Key	Description
`pwdencrypted=true | false`	When **`true`**, the OPC-DA password is encrypted. The default value is **`false`**.
`blocksize=`*`size`*	Specifies the number of events per event block. The default value is 1.

Key	Description
`tokenlocation=`*`location`*	Specifies the location of the file on the local file system that contains the OAuth token required for authentication by the publish/subscribe server.
`samplinginterval=`*`number`*	Specifies the OPC-DA value sampling interval in milliseconds. The default value is 10.
`enablejinteroplogger=true | false`	When true, enable the j-interop built-in log handler. The default value is false.
`publishwithupsert=true | false`	When **`true`**, build events with opcode = Upsert instead of Insert.
`transportconfigfile=`*`file`*	Specifies the publish/subscribe transport configuration file. For Solace, the default is **`./solace.cfg`**. For Tervela, the default is **`./client.config`**. For RabbitMQ, the default is **`./rabbitmq.cfg`**. For Kafka, the default is **`./kafka.cfg`**.
`maxevents=`*`number`*	Specifies the maximum number of events to publish.
`configfilesection=[`*`section`*`]`	Specifies the name of the section in **`/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config`** (Linux) or **`%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.config`** (Windows) to parse for configuration parameters. Specify the value as `[`*`section`*`]`.
`loglevel=trace | debug | info | warn | error | fatal | off`	Specifies 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`**.
`tagsfile=`*`pathtofile`*	Specifies the full path to a file containing tags to read. The format is one tag ID per line. Tags added while browsing are also added to this file. The default value is no tags file.
`pubsublib=`*`transport`*	Specifies the transport type, valid values are **`"native"`**, **`"rabbitmq"`**, **`"kafka"`**, **`"solace"`**, and **`"tervela"`**. The default value is **`"native"`**.
`gdconfigfile=`*`file`*	Specifies the Guaranteed Delivery configuration file.

Key	Description
`transactional=true | false`	When **`true`**, event blocks are transactional. The default value is **`false`**.
`quiesceproject=true | false`	When **`true`**, quiesce the project after all events are injected into the Source window.

Note: If a value string includes a comma, then enclose the entire value string in double quotation marks.

Note: The following characters in a value string must be escaped with a reverse slash (\): double quotation mark, space, parentheses.

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 window that is associated with the connector must use the following schema:

```
ID*:int32,elementindex :string,element :string,attribute :string,value :variable:,
timestamp:stamp ,status:string
```

Use the following schema values.

Schema Value	Description
elementindex	Value of the connector *afelement* configuration parameter. Specify either an AF element name or an AF element template name.
element	Name of the AF element associated with the *value*.
attribute	Name of the AF attribute owned by the AF *element*.
value	Value of the AF *attribute*.
timestamp	Timestamp associated with the *value*.
status	Status associated with the *value*.

Note: The type of the *value* field is determined by the type of the AF attribute as defined in the AF hierarchy.

The following mapping of event stream processor data type to AF attributes is supported:

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

Event Stream Processor Data Type	AF Attribute
	TypeCode::UInt32
ESP_INT64	TypeCode::Int64 TypeCode::UInt64
ESP_UTF8STR	TypeCode::String
ESP_TIMESTAMP	TypeCode::DateTime

If an attribute has `TypeCode::Object`, the connector 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
ESP_DOUBLE	PIPointType::Float16 PIPointType::Float32
ESP_TIMESTAMP	PIPointType::Timestamp
ESP_UTF8STR	PIPointType::String

Note: It is strongly recommended to install the PI SDK Utility along with PI AF Client to facilitate troubleshooting and confirm connector functionality.

Use the following parameters with PI connectors:

Table 108 *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`.

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

Table 109 *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 110 *Optional Parameters for Subscriber PI Connectors*

Parameter	Description
rmretdel	Removes all delete events from event blocks received by the subscriber that were introduced by a window retention policy.
pisystem	Specifies the PI system. The default is the PI system that is configured in the PI AF client.
afdatabase	Specifies the AF database. The default is the AF database that is configured in the PI AF client.
afrootelement	Specifies the root element in the AF hierarchy from which to search for parameter afelement . The default is the top-level element in the AF database.
afattribute	Specifies a specific attribute in the element. The default is all attributes in the element.
configfilesection	Specifies the name of the section in **/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config** (Linux) or **%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default**

Parameter	Description
	\connectors.config (Windows) to parse for configuration parameters. Specify the value as `[configfilesection ]`.

Table 111 *Optional Parameters for Publisher PI Connectors*

Parameter	Description
`blocksize`	Specifies the number of events to include in a published event block. The default value is 1.
`transactional`	Sets the event block type to transactional. The default event block type is normal.
`pisystem`	Specifies the PI system. The default is the PI system that is configured in the PI AF client.
`afdatabase`	Specifies the AF database. The default is the AF database that is configured in the PI AF client.
`afrootelement`	Specifies the root element in the AF hierarchy from which to search for the parameter `afelement`. The default is the top-level element in the AF database.
`afattribute`	Specifies a specific attribute in the element. The default is all attributes in the element.
`archivetimestamp`	Specifies that all archived values from the specified timestamp onwards are to be published when connecting to the PI system. The default is to publish only new values.
`configfilesection`	Specifies the name of the section in **/opt/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 ]`.
`publishwithupsert`	Builds events with `opcode=Upsert` instead of `Insert`.
`maxevents`	Specifies the maximum number of events to publish.
`allvaluestostrings`	Specifies to convert all received values to a string. Requires that the `value` field in the Source window have `type = ESP_UTF8STR` .

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, ...*

Subscriber Usage

Table 112 *Required Keys*

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

Table 113 *Optional Keys*

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

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

Publisher Usage

Table 114 *Required Keys*

Key-Value Pair	Description
`type=pub`	Specifies a publisher adapter.
`url=` *pubsubURL*	Specifies the standard URL in the form **`dfESP://`** ***`host :port/ project/continuousquery /window`***.
`afelement=` *element*	Specifies the AF element or element template name. Wildcards are supported.

Table 115 *Optional Keys*

Key-Value Pair	Description
`iselementtemplate=true | false`	When `true` , specifies that the value of `afelement` is the name of an element template.
`pisystem=` *name*	Specifies the name of the PI system.
`afdatabase=` *name*	Specifies the name of the Asset Framework database.
`afrootelement=` *element*	Specifies a root element in the Asset Framework hierarchy from which to search for `afelement`.
`afattribute=` *attribute*	Specifies an attribute in *afelement* and ignores other attributes there.
`gdconfig=` *file*	Specifies the guaranteed delivery configuration file.
`transport=native | solace | tervela | rabbitmq | kafka`	Specifies the transport type. If you specify `solace`, `tervela` , `rabbitmq`, or `kafka` transports instead of the default `native` transport, then use the required client configuration files specified in the description of the C++ C_dfESPpubsubSetPubsubLib() API call.
`loglevel=trace | debug | info | warn | error | fatal | off`	Sets the logging level for the adapter. This is the same range of logging levels that you can set in the `C_dfESPpubsubInit()` publish/subscribe API call and in the engine `initialize()` call. The default level is `warn`.
`logconfigfile=` *file*	Specifies the log configuration file.

Key-Value Pair	Description
`tokenlocation=` *location*	Specifies the location of the file in the local file system that contains the OAuth token that is required for authentication by the publish/subscribe server.
`configfilesection=[` *section*`]`	Specifies the name of the section in **/opt/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*`]`.
`restartonerror=true` | `false`	When `true`, specifies to restart the adapter if a fatal error is reported.
`transportconfigfile=` *file*	Specifies the publish/subscribe transport configuration file. For `solace`, the default is **./solace.cfg**. For `tervela`, the default is **./client.config**. For `rabbitmq`, the default is **./rabbitmq.cfg**. For `kafka`, the default is **./kafka.cfg**. **Note:** No transport configuration file is required for native transport.
`archivetimestamp=` *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`, event blocks are transactional. By default, event blocks are normal.
`publishwithupsert=true` | `false`	When `true`, build events with opcode = Upsert instead of Insert.
`maxevents=` *number*	Specifies the maximum number of events to publish.
`quiesceproject=true` | `false`	When `true`, quiesces the project after all events are injected into the Source window.
`allvaluestostrings=true` | `false`	When `true`, specifies to convert all received values to a string. Requires that the `value` field in the Source window have `type = ESP_UTF8STR`.

Publisher Failover

For information about implementing hot failover for publisher adapters, see "Publisher Adapter Failover with Kafka" in *SAS Event Stream Processing: Implementing Failover*.

Using the PI Web Connector and Adapter

Overview

The PI Web API enables you to retrieve and manipulate time series, asset, and event frame data. The API also supports indexing and searching metadata within the Asset Framework (AF) and PI Data Archive components of the PI System.

Using the PI Web Connector

Overview

The PI Web connector supports publish and subscribe operations for a PI AF server. The AF data reference can be defined as a PI Point for these elements. The model must implement a window of a fixed schema in order to carry values that are associated with AF attributes owned by AF elements in the AF hierarchy.

The window that is associated with the connector must use the following fixed schema:

```
ID*:int32,elementindex:string,element:string,attribute:string,
value:variable,timestamp:stamp,status:string
```

Use the following schema values.

Schema Value	Description
elementindex	Value of the connector *afelement* configuration parameter. Specify either an AF element name or an AF element template name.
element	Name of the AF element associated with the *value*.
attribute	Name of the AF attribute owned by the AF *element*.
value	Value of the AF *attribute*.
timestamp	Timestamp associated with the *value*.

Schema Value	Description
status	Status associated with the *value*.

Note: The type of the *value* field is determined by the type of the AF attribute as defined in the AF hierarchy. This type is exposed by the PI Web API as one of a set of supported Attribute Data Types.

The following mapping of event stream processor data type to Attribute Data Types is supported:

Event Stream Processor Data Type	Attribute Data Type
ESP_DOUBLE	Single Double
ESP_INT32	Byte Int16 Int32 Boolean
ESP_INT64	Int64 Boolean
ESP_UTF8STR	String Guid
ESP_TIMESTAMP	DateTime

When required, you can configure the `piaccesstoken` parameter with an encrypted token. The encrypted version of the token 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 invoke OpenSSL to display your encrypted token:

```
echo 'piaccesstoken' | openssl enc -e -aes-256-cbc -md md5 -a -salt -pass
pass:'espPIWEBconnectorUsedByUser=pisystem'
```

Then copy the encrypted token into your `piaccesstoken` parameter and configure `piaccesstokenencrypted=true`.

Refreshing Access Tokens That Use OAuth 2.0

You can configure the PiWeb connector and adapter to fetch and refresh access tokens from an OAuth 2.0 platform. Enable this functionality by specifying an `oauthgranttype`.

There are two methods to request access tokens:

- using a refresh token

- using client credentials

To get an access token from a refresh token, the OAuth 2.0 platform must support the Refresh Token grant type. If the OAuth provider responds to an access token request with a refresh token, then the value of the refresh token is updated.

To get an access token from a client secret and client ID, the OAuth 2.0 platform must support the Client Credentials grant type. Token refresh intervals are not handled automatically, and must be specified with the `oauthtokenrefreshrate` parameter.

Publisher Usage

Table 116 *Required Parameters for Publisher PI Web Connectors*

Parameter	Description
`type`	Specifies a publisher connector. Must be "`pub`".
`pihost`	Specifies the system name or IP address of the system that hosts the PI System.
`pisystem`	Specifies the name of the PI System.
`afdatabase`	Specifies the name of the Asset Framework (AF) database.
`afelement`	Specifies the AF element or element template name. Wildcards are supported.

Table 117 *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.
`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 behavior is to publish only new values.

Parameter	Description
	Specify a value as a time string that conforms to the format described at the *OSI PI Web API Reference*.
`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 ]`.
`publishwithupsert`	Builds events with `opcode=Upsert` instead of `Insert`.
`maxevents`	Specifies the maximum number of events to publish.
`includecurrentvalue`	Specifies to publish the current value of all configured attributes before publishing new values.
`iselementtemplate`	Specifies that the `afelement` parameter is an element template name. By default, the `afelement` parameter specifies an element name.
`allvaluestostrings`	Specifies to convert all received values to a string. Requires that the `value` field in the Source window have `type = ESP_UTF8STR`.
`piaccesstoken`	Specifies the access token required to access the AF database.
`piaccesstokenencrypted`	When `true`, specifies that `piaccesstoken` is encrypted.
`picertificate`	Specifies the full path to a **`.pem`** file used for encrypted connections to the PI server.
`picertificatepassphrase`	Specifies the passphrase for the certificate configured in `picertificate`.
`piauthmethod`	Specifies the authentication method for the token configured in `piaccesstoken`. Valid values are **`basic`** and **`bearer`**. The default value is **`bearer`**.
`oauthgranttype`	Specifies whether OAuth token refresh is enabled. A value of **`credentials`** requests access tokens using the Client Credentials grant type. A value of **`token`** requests access tokens using the Refresh Token grant type.
`oauthtokenendpoint`	This is required when `oauthgranttype` is specified. Specifies the OAuth 2.0 token endpoint URL.
`oauthtokenrefreshrate`	Specifies the number of seconds between token refreshes. The default value is 900.

Parameter	Description
`oauthclientid`	This is required when `oauthgranttype` is specified. Specifies the ID for the OAuth client.
`oauthclientsecret`	This is required when `oauthgranttype` is specified. Specifies the client secret for the OAuth client.
`ignoreserverdown`	Specifies to periodically retry the connection to the PI Web server when it goes down. The retry uses an exponential backoff algorithm. Failed requests are not retried. The default value is **`false`**.
`searchfullheirarchy`	Specifies whether to search the full Asset Framework hierarchy when finding elements and attributes. The default value is **`true`**.
`maxattributesperwebsocket`	Specifies the maximum number of attributes to read per websocket. The default value is 90.
`websocketidletimeout`	Specifies a timeout value in seconds that causes a websocket to be closed after the specified idle time. Requires that `ignoreserverdown` also be configured, which causes the websocket to be reconnected.

Subscriber Usage

Table 118 *Required Parameters for Subscriber PI Web Connectors*

Parameter	Description
`type`	Specifies a subscriber connector. The value must be `sub`.
`pihost`	Specifies the system name or IP address of the system that hosts the PI System.
`pisystem`	Specifies the name of the PI System.
`afdatabase`	Specifies the name of the Asset Framework (AF) database.
`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 119 *Optional Parameters for Subscriber PI Connectors*

Parameter	Description
`rmretdel`	Removes all deleted events from event blocks received by the subscriber that were introduced by a window retention policy.
`afrootelement`	Specifies the root element in the AF hierarchy from which to search for the attribute and element contained in subscribed events. The default is the top-level element in the AF database.
`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 ]`.
`cachetargetattributes`	Specifies to save found element-attribute pairs to a cache, instead of looking them up for every subscribed event. Setting this parameter might improve performance, but at the cost of increased memory usage.
`piaccesstoken`	Specifies the access token required to access the AF database.
`piaccesstokenencrypted`	When `true`, specifies that `piaccesstoken` is encrypted.
`picertificate`	Specifies the full path to a **`.pem`** file used for encrypted connections to the PI server.
`picertificatepassphrase`	Specifies the *passphrase* for the certificate configured in `picertificate`.
`piauthmethod`	Specifies the authentication method for the token configured in `piaccesstoken`. Valid values are **`basic`** and **`bearer`**. The default value is **`bearer`**.
`oauthtokenendpoint`	This is required when `oauthgranttype` is specified. Specifies the OAuth 2.0 token endpoint URL.
`oauthtokenrefreshrate`	Specifies the number of seconds between token refreshes. The default value is 900.
`oauthclientid`	This is required when `oauthgranttype` is specified. Specifies the ID for the OAuth client.
`oauthgranttype`	Specifies whether OAuth token refresh is enabled. A value of **`credentials`** requests access tokens using the Client Credentials grant type. A value of **`token`** requests access tokens using the Refresh Token grant type.

Parameter	Description
`oauthclientsecret`	This is required when `oauthgranttype` is specified. Specifies the client secret for the OAuth client.
`ignoreserverdown`	Specifies to periodically retry the connection to the PI Web server when it goes down. The retry uses an exponential backoff algorithm. Failed requests are not retried. The default value is **`false`**.
`searchfullheirarchy`	Specifies whether to search the full Asset Framework hierarchy when finding elements and attributes. The default value is **`true`**.
`nocachecontrolheader`	When **`true`**, specifies not to include the cache-control header in HTTP requests. The default value is **`false`**, which specifies to include the cache-control header with `value="no-cache"`.
`osisoftmessageformat`	Send HTTP requests to the Osisoft Message Format (OMF) endpoint instead of the Streamsets endpoint. The default value is **`false`**.
`omfutc`	Write timestamps in OMF data messages in UTC instead of local time. The default value is **`false`**.
`searchattributesunbatched`	Do not batch searches for new element/attributes found in a subscribed event block. The default value is **`false`**.
`stickycookiename`	Specifies the name of an HTTP cookie to copy from the status response to HTTP POST requests. The cookie is included in subsequent HTTP POST requests.

Using the PI Web Adapter

Overview

The PI Web adapter supports publish and subscribe operations against a PI Asset Framework (PI) server. It uses the PI Web API to issue REST requests and to open WebSocket connections to the PI server.

dfesp_piweb_adapter -C *key1* =*val1*,*key2* =*val2*,*key3* =*val3*, ...

Subscriber Usage

Table 120 *Required Keys*

Key-Value Pair	Description
`type=sub`	Specifies a subscriber adapter.
`url=` *pubsubURL*	Specifies the standard URL in the form **`dfESP://`** ***`host :port/ project/continuousquery /window`***. Append `?snapshot=true |false` for subscribers. Append`?collapse=true | false` or `?rmretdel=true | false` or both for subscribers if needed.
`pihost=` *name*	Specifies the system name or IP address of the system that hosts the PI System.
`pisystem=` *name*	Specifies the name of the PI System.
`afdatabase=` *element*	Specifies the name of the Asset Framework (AF) database.

Table 121 *Optional Keys*

Key-Value Pair	Description
`afrootelement=` *element*	Specifies a root element in the AF hierarchy from which to search for the attribute and element contained in subscribed events. By default, the entire hierarchy is searched.
`gdconfig=` *file*	Specifies the guaranteed delivery configuration file.
`transport=native | solace | rabbitmq | kafka`	Specifies the transport type. If you specify `solace`, `rabbitmq`, or `kafka` transports instead of the default `native` transport, then use the required client configuration files specified in the description of the C++ C_dfESPpubsubSetPubsubLib() API call.
`loglevel=trace | debug | info | warn | error | fatal | off`	Sets the logging level for the adapter. This is the same range of logging levels that you can set in the `C_dfESPpubsubInit()` publish/subscribe API call and in the engine `initialize()` call. The default level is `warn`.
`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 **`/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 ]`.
`restartonerror=true | false`	When `true`, specifies to restart the adapter if a fatal error is reported.
`transportconfigfile= file`	Specifies the publish/subscribe transport configuration file. For `solace`, the default is **`./solace.cfg`**. For `rabbitmq`, the default is **`./rabbitmq.cfg`**. For `kafka`, the default is **`./kafka.cfg`**. **Note:** No transport configuration file is required for native transport.
`piaccesstoken= string`	Specifies the access token required to access the AF database.
`piaccesstokenencrypted=true | false`	When **`true`**, specifies that `piaccesstoken` is encrypted.
`piauthmethod=basic | bearer`	Specifies the authentication method for the token configured in `piaccesstoken`. The default value is **`bearer`**.
`ignoreserverdown=true | false`	Specifies to periodically retry the connection to the PI Web server when it goes down. The retry uses an exponential backoff algorithm. Failed requests are not retried. The default value is **`false`**.
`searchfullheirarchy=true | false`	Specifies whether to search the full Asset Framework hierarchy when finding elements and attributes. The default value is **`true`**.
`nocachecontrolheader= true | false`	When **`true`**, specifies not to include the cache-control header in HTTP requests. The default value is **`false`**, which specifies to include the cache-control header with `value="no-cache"`.
`osisoftmessageformat= true | false`	Send HTTP requests to the Osisoft Message Format (OMF) endpoint instead of the Streamsets endpoint. The default value is **`false`**.
`omfutc= true | false`	Write timestamps in OMF data messages in UTC instead of local time. The default value is **`false`**.

Key-Value Pair	Description
`searchattributesunbatched= true | false`	Do not batch searches for new element/attributes found in a subscribed event block. The default value is **`false`**.
`stickycookiename=`*`name`*	Specifies the name of an HTTP cookie to copy from the status response to HTTP POST requests. The cookie is included in subsequent HTTP POST requests.

Publisher Usage

Table 122 *Required Keys*

Key-Value Pair	Description
`type=pub`	Specifies a publisher adapter.
`url=` *`pubsubURL`*	Specifies the standard URL in the form **`dfESP://`** ***`host :port/ project/continuousquery /window`***.
`pihost=` *`name`*	Specifies the system name or the IP address of the system hosting the PI System.
`pisystem=` *`name`*	Specifies the name of the PI System.
`afdatabase=`*`name`*	Specifies the name of the Asset Framework (AF) database.
`afelement=` *`element`*	Specifies the AF element or element template name. Wildcards are supported.

Table 123 *Optional Keys*

Key-Value Pair	Description
`iselementtemplate=true | false`	When `true`, specifies that the value of `afelement` is the name of an element template.
`afrootelement=` *`element`*	Specifies a root element in the Asset Framework hierarchy from which to search for `afelement`.
`afattribute=` *`attribute`*	Specifies an attribute in *afelement* and ignores other attributes there.
`gdconfig=` *`file`*	Specifies the guaranteed delivery configuration file.

Key-Value Pair	Description
`transport=native | solace | rabbitmq | kafka`	Specifies the transport type. If you specify `solace`, `rabbitmq`, or `kafka` transports instead of the default `native` transport, then use the required client configuration files specified in the description of the C++ C_dfESPpubsubSetPubsubLib() API call.
`loglevel=trace | debug | info | warn | error | fatal | off`	Sets the logging level for the adapter. This is the same range of logging levels that you can set in the `C_dfESPpubsubInit()` publish/subscribe API call and in the engine `initialize()` call. The default level is `warn`.
`logconfigfile=` *file*	Specifies the log configuration file.
`tokenlocation=` *location*	Specifies the location of the file in the local file system that contains the OAuth token that is required for authentication by the publish/subscribe server.
`configfilesection=[`*section*`]`	Specifies the name of the section in **`/opt/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*`]`.
`restartonerror=true | false`	When `true`, specifies to restart the adapter if a fatal error is reported.
`transportconfigfile=` *file*	Specifies the publish/subscribe transport configuration file. For `solace`, the default is **`./solace.cfg`**. For `rabbitmq`, the default is **`./rabbitmq.cfg`**. For `kafka`, the default is **`./kafka.cfg`**. **Note:** No transport configuration file is required for native transport.
`archivetimestamp=` *timestamp*	Specifies that all archived values from the specified *timestamp* onwards are to be published when connecting to the PI system. The default behavior is to publish only new values. Specify a value as a time string that conforms to the format described at the *OSI PI Web API Reference*.
`transactional=true | false`	When `true`, event blocks are transactional. By default, event blocks are normal.
`publishwithupsert=true | false`	When `true`, build events with opcode = Upsert instead of Insert.

Key-Value Pair	Description
`maxevents=` *number*	Specifies the maximum number of events to publish.
`quiesceproject=true | false`	When **`true`**, quiesces the project after all events are injected into the Source window.
`allvaluestostrings=true | false`	When **`true`**, specifies to convert all received values to a string. Requires that the `value` field in the Source window have `type = ESP_UTF8STR`.
`includecurrentvalue=true | false`	When **`true`**, publish the current value of all configured attributes before publishing new values.
`piaccesstoken=`*`string`*	Specifies the access token required to access the AF database.
`piaccesstokenencrypted=true | false`	When **`true`**, specifies that `piaccesstoken` is encrypted.
`picertificate=`*`path`*	Specifies the full path to a **`.pem`** file used for encrypted connections to the PI server.
`picertificatepassphrase=`*`passphrase`*	Specifies the *passphrase* for the certificate configured in `picertificate`.
`piauthmethod=basic | bearer`	Specifies the authentication method for the token configured in `piaccesstoken`. The default value is **`bearer`**.
`oauthgranttype = 'credentials' | 'token'`	Specifies whether OAuth token refresh is enabled. A value of **`credentials`** requests access tokens using the Client Credentials grant type. A value of **`token`** requests access tokens using the Refresh Token grant type.
`oauthtokenendpoint =` *`url`*	This is required when `oauthgranttype` is specified. Specifies the OAuth 2.0 token endpoint URL.
`oauthtokenrefreshrate =` *`seconds`*	Specifies the number of seconds between token refreshes. The default value is 900.
`oauthclientid =` *`OAuthclientID`*	This is required when `oauthgranttype` is specified. Specifies the ID for the OAuth client.
`oauthclientsecret =` *`secret`*	This is required when `oauthgranttype` is specified. Specifies the client secret for the OAuth client.
`ignoreserverdown=true | false`	Specifies to periodically retry the connection to the PI Web server when it goes down. The retry uses an exponential backoff algorithm. Failed requests are not retried. The default value is **`false`**.

Key-Value Pair	Description
`searchfullheirarchy=true | false`	`searchfullheirarchy` Specifies whether to search the full Asset Framework hierarchy when finding elements and attributes. The default value is **`true`**.
`maxattributesperwebsocket=`*`number`*	Specifies the maximum number of attributes to read per websocket. The default value is 90.
`websocketidletimeout=`*`seconds`*	Specifies a timeout value in seconds that causes a websocket to be closed after the specified idle time. Requires that `ignoreserverdown` also be configured, which causes the websocket to be reconnected.

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 124 *Required Parameters for the Project Publish Connector*

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

Table 125 *Optional Parameters for the Project Publish Connector*

Parameter	Description
`maxevents`	Specifies the maximum number of events to publish.
`configfilesection`	Specifies the name of the section in **`/opt/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 ]`.

Using the Pylon Publisher Connector and Adapter

Using the Pylon Publisher Connector

Overview

IMPORTANT This functionality is no longer supported or included in SAS Viya 3.5 revision 24w44. No updates to this functionality are available for earlier releases.

The Pylon connector communicates with a Basler GigE or a USB-attached camera to continuously publish captured frames into a Source window.

Full access to the complete set of camera configuration parameters is available only in the pylon Camera Software Suite, which you can download from https://www.baslerweb.com/en/support/downloads/software-downloads/. This utility can save a camera's configuration to a file, and the path to that file can be configured on the Pylon connector to enable you to run the camera with any valid camera configuration.

The frame data received by the Pylon connector is uncompressed and copied into a Source window field of type ESP_BINARY. You can override the camera's default image format with the `camerapixelformat` parameter. When this parameter is configured, a console message is logged at start-up that shows whether the desired format was successfully configured on the camera.

Note: Processing of captured frames by a video encoding window requires that the camera's pixel format be set to **`BayerRG8`**.

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.

If the optional `cameraid` configuration parameter is configured, then the Source window schema must also contain an extra string field at the end. For example:

```
id*:int64,frame:blob,cameraid:string
```

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

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 Pylon 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).

Before You Use the GigE Camera

The GigE camera must have a known, fixed IP address, and the attached Ethernet network cable must provide power using Power-over-Ethernet.

If the IP address for the GigE camera is unknown or if the camera does not have an IP configuration, then you must run the Pylon IP Configurator. This configurator is available with the pylon Camera Software Suite. In order for the GigE camera to be discovered, it must be connected to the same subnet as the machine running the software suite. You can then copy the camera's IP address into the connector configuration in order to run the Pylon connector.

If no GigE camera IP address has been configured for the Pylon connector, then it connects to the first camera that it finds on the local subnet. Running the Pylon connector without a configured IP address for the camera might be useful for testing, but it is not recommended for production deployments.

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

Before You Use the USB Attached Camera

For USB cameras that are attached to a Linux computer, user level access to the USB device might not be enabled, or kernel drivers might not be present. In this case, you might need to update the file **`/etc/udev/rules.d`** with a new rule similar to this one:

```
#Enable user access to all basler cameras
SUBSYSTEM=="usb", ATTRS{idVendor}=="2676", MODE:="0666", TAG+="uaccess", TAG+="udev-acl"
```

Parameters

Table 126 *Required Parameters for Pylon Connectors*

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

Table 127 *Optional Parameters for Pylon Connectors*

Parameter	Description
`cameraipaddress`	Specifies the camera IP address. The default value is the address of the first camera found on the local subnet. **Note:** This parameter is not supported for USB cameras.
`maxnumframes`	Specifies the maximum number of frames to publish. The default value is no maximum.
`maxframerate`	Specifies the maximum number of frames per second to publish. The default value is the rate at which frames are received from the camera.
`camerafeaturesfile`	Specifies a Pylon Features Stream (`*.pfs`) configuration file to load. The default is to use the current camera configuration unmodified.
`camerawidth`	Specifies the Area-Of-Interest width. The default value is the value in the current camera configuration.
`cameraheight`	Specifies the Area-Of-Interest height. The default value is the value in the current camera configuration.
`camerapixelformat`	Specifies the image pixel format. The default value is the format in the current camera configuration.
`camerapacketsize`	Specifies the Ethernet packet size. The default value is the value in the current camera configuration. **Note:** This parameter is not supported for USB cameras.
`transactional`	Sets the event block type to transactional. The default event block type is normal.
`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`*` ]`.

Parameter	Description
`publishwithupsert`	Builds events with `opcode=Upsert` instead of `opcode=Insert`.
`cameraxoffset`	Specifies the Area-Of-Interest horizontal offset. The default value is the value in the current camera configuration.
`camerayoffset`	Specifies the Area-Of-Interest vertical offset. The default value is the value in the current camera configuration.
`maxevents`	Specifies the maximum number of events to publish.
`convertyuyvtojpg`	Converts images that are in UYV422_YUVY_Packed format into JPG format. This parameter is not supported on Microsoft Windows.
`cameraid`	Specifies an arbitrary string that is copied into the corresponding string field in the Source window. This value can be used by the model to identify the source camera.

Using the Pylon Publisher Adapter

IMPORTANT This functionality is no longer supported or included in SAS Viya 3.5 revision 24w44. No updates to this functionality are available for earlier releases.

The Pylon publisher adapter provides the same functionality as the Pylon publisher connector, with the addition of some adapter-only optional parameters. To run the adapter with a GigE camera, run the following command:

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 single quotation marks (for example, *`key =\'str,ing \'`*).

To run the adapter with a USB camera, run the following command:

dfesp_pylonusb_adapter -C *key1 =val1,key2 =val2,key3 =val3, ...*

Table 128 *Required Keys*

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

Table 129 *Optional Keys*

Key-Value Pair	Description
`gdconfig=` *file*	Specifies the guaranteed delivery configuration file.
`transport=native | solace | tervela | rabbitmq | kafka`	Specifies the transport type. If you specify `solace`, `tervela`, `rabbitmq`, or `kafka` transports instead of the default `native` transport, then use the required client configuration files specified in the description of the C++ C_dfESPpubsubSetPubsubLib() API call.
`loglevel=trace | debug | info | warn | error | fatal | off`	Sets the logging level for the adapter. This is the same range of logging levels that you can set in the `C_dfESPpubsubInit()` publish/subscribe API call and in the engine `initialize()` call. The default level is `warn`.
`logconfigfile=` *file*	Specifies the log configuration file.
`tokenlocation=` *location*	Specifies the location of the file in the local file system that contains the OAuth token that is required for authentication by the publish/subscribe server.
`configfilesection=[` *section*`]`	Specifies the name of the section in **`/opt/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*`]`.
`transactional=true | false`	When `true`, event blocks are transactional. By default, event blocks are normal.
`publishwithupsert=true | false`	When `true`, build events with opcode = Upsert instead of Insert.
`transportconfigfile=` *file*	Specifies the publish/subscribe transport configuration file. For `solace`, the default is **`./solace.cfg`**. For `tervela`, the default is **`./client.config`**. For `rabbitmq`, the default is **`./rabbitmq.cfg`**. For `kafka` , the default is **`./kafka.cfg`**. **Note:** No transport configuration file is required for native transport.
`cameraipaddress=` *ipaddress*	Specifies the camera IP address. The default value is the address of the first camera found on the local subnet. **Note:** This key is not supported for USB cameras.

Key-Value Pair	Description
`maxnumframes= number`	Specifies the maximum number of frames to publish. The default value is no maximum.
`maxframerate= number`	Specifies the maximum number of frames per second to publish. The default value is the rate at which frames are received from the camera.
`camerafeaturesfile= configfile`	Specifies a Pylon Features Stream (*.pfs) configuration file to load. The default is to use the current camera configuration unmodified.
`camerawidth= width`	Specifies the Area-Of-Interest width. The default value is the value in the current camera configuration.
`cameraheight= height`	Specifies the Area-Of-Interest height. The default value is the value in the current camera configuration.
`camerapixelformat= format`	Specifies the image pixel format. The default value is the format in the current camera configuration.
`camerapacketsize= size`	Specifies the Ethernet packet size. The default value is the value in the current camera configuration. **Note:** This key is not supported for USB cameras.
`cameraxoffset= horizontaloffset`	Specifies the Area-Of-Interest horizontal offset. The default value is the value in the current camera configuration.
`camerayoffset= verticaloffset`	Specifies the Area-Of-Interest vertical offset. The default value is the value in the current camera configuration.
`maxevents= number`	Specifies the maximum number of events to publish.
`quiesceproject=true | false`	When `true`, quiesces the project after all events are injected into the Source window.
`restartonerror=true | false`	When `true`, specifies to restart the adapter when a fatal error is reported.
`convertyuyvtojpg=true | false`	When `true`, converts images that are in UYV422_YUVY_Packed format into JPG format. This parameter is not supported on Microsoft Windows.
`cameraid=id`	Specifies an arbitrary string that is copied into the corresponding string field in the Source window. This value can be used by the model to identify the source camera.

Using the QuasarDB Connector and Adapter

Overview

QuasarDB is a distributed database for time series data that combines in-memory capabilities with long-term storage. Use cases include the following.

- market data store for back testing and analysis
- trades store for compliance
- time deviation database for compliance
- sensor data repository for predictive maintenance
- monitoring database for large deployments

The publisher grabs rows from a table in the QuasarDB cluster through a user-configured select statement. It then injects an event per row into the associated Source window. Any non-key Source window fields that are not present in the result set from the cluster are left with a null value. If the Source window has an int64 field named `index` defined as a key field, then the publisher simply writes this field with an incrementing value that starts at 0. If any other Source window key fields are not present in the source table, then a fatal error is logged and the connector stops.

By default, the publisher adds an `IN RANGE(epoch, now)` clause to the configured select statement, where `now` is the current system time. The connector then repeatedly executes the `select` statement, with each iteration using the prior iteration's end time as the start time (instead of `epoch`). The default interval is 1 second; you can configure it to an alternative value. You can also configure an alternate start time for the initial iteration (instead of epoch) through the optional `inrangestartabs` or `inrangestartrel` parameters.

The publisher supports the following type mappings:

Table 130 *Publisher Type Mappings*

QuasarDB Type	ESP Type
qdb_query_result_double	ESP_DOUBLE
qdb_query_result_blob	ESP_BINARY
qdb_query_result_int64	ESP_INT64
qdb_query_result_timestamp	ESP_TIMESTAMP or ESP_DATETIME

QuasarDB Type	ESP Type
qdb_query_result_count	ESP_INT64
qdb_query_result_string	ESP_UTF8STR
qdb_query_result_none	anything; field value is null

The subscriber executes Insert, Update, and Delete operations on a user-configured destination table in the cluster. If the table does not exist, then the subscriber creates it by executing a `CREATE TABLE` query using the subscribed window schema. If the table already exists, then the subscribed window field names and types must match the destination table's column definitions.

For insert events that are generated by the subscribed window, a single batch insert is executed for all insert events that are contained in a generated event block. For update and delete events in the event block, one explicit update or delete query statement is sent to the table per event, with a WHERE clause to identify the key fields.

For debugging purposes, you can log these query statements to the console by enabling debug level logging. You can enable debug-level logging when running the model that contains the connector or you can run the adapter with debug-level logging.

By default, the timestamp that is associated with batch inserts is the current system time when the batch is built. Alternatively, if the subscribed window contains a timestamp field named `$timestamp`, then that value is used instead.

The subscriber supports the following type mappings:

Table 131 *Subscriber Type Mappings*

ESP Type	QuasarDB Type
ESP_DOUBLE	qdb_ts_column_double
ESP_BINARY	qdb_ts_column_blob
ESP_INT64	qdb_ts_column_int64
ESP_TIMESTAMP	qdb_ts_column_timestamp
ESP_DATETIME	qdb_ts_column_timestamp
ESP_INT32	qdb_ts_column_int64
ESP_UTF8STR	qdb_ts_column_string

Using the QuasarDB Connector

Note: To use the QuasarDB connector, you must install version 3.9.*x* or higher of the QuasarDB C API libraries on the platform where you run the connector.

Table 132 *Required Parameters for Subscriber Connectors*

Parameter	Description
`qdbhostport`	Specifies the QuasarDB cluster host and port, in the form *`host:port`*.
`desttablename`	Specifies the target table name.
`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.
`type`	Specifies to subscribe. Must be **`sub`**.

Table 133 *Optional Parameters for Subscriber Connectors*

Parameter	Description
`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`*` ]`.
`commitrows`	Specifies a minimum number of output rows to buffer.
`commitsecs`	Specifies the maximum number of seconds to hold on to an incomplete commit buffer.
`rmretdel`	Specifies to remove all Delete events from event blocks received by a subscriber that were introduced by a window retention policy.
`qdbclusterpublickey`	Specifies the cluster public key.

Parameter	Description
qdbuser	Specifies the user name part of the user credentials.
qdbuserprivatekey	Specifies the private key part of the user credentials.
qdbclusterpublickeyfile	Specifies the complete path to a file containing the cluster public key.
qdbusercredentialsfile	Specifies the complete path to a file containing the user credentials.

Table 134 *Required Parameters for Publisher Connectors*

Parameter	Description
qdbhostport	The QuasarDB cluster host and port, in the form *host*:*port*.
selectstatement	Specifies the SQL statement to be executed on the source database. By default, the connector adds an `IN RANGE ()` clause to this statement. Enable the `norange` parameter to disable this functionality. By default, this statement is executed repeatedly at 1-second intervals with the `IN RANGE` parameters adjusted to obtain a continuous set of changes to the table. Configure the `repeatinterval` parameter to modify this interval.
type	Specifies to publish. Must be **pub**.

Table 135 *Optional Parameters for Publisher 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 **/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 ]`.
maxevents	Specifies the maximum number of events to publish.
publishwithupsert	Builds events with `opcode=Upsert` instead of Insert.

Parameter	Description
`transactional`	Sets the event block type to transactional. The default event block type is normal.
`repeatinterval`	Specifies the number of seconds between publisher repetitions. The default value is 1.
`qdbclusterpublickey`	Specifies the cluster public key.
`qdbuser`	Specifies the user name part of the user credentials.
`qdbuserprivatekey`	Specifies the private key part of the user credentials.
`qdbclusterpublickeyfile`	Specifies the complete path to a file containing the cluster public key.
`qdbusercredentialsfile`	Specifies the complete path to a file containing the user credentials.
`norange`	Specifies whether the configured `selectstatement` is executed without an `IN RANGE ()` clause added to it.
`inrangestartabs`	Specifies an absolute timestamp in QuasarDB syntax to use as the first value in the `IN RANGE()` clause added to the configured `selectstatement`. The default value is **`epoch`**. This applies only to the first execution of `selectstatement`.
`inrangestartrel`	Specifies a negative offset in QuasarDB syntax relative to `now` to use as the second value in the `IN RANGE()` clause added to the configured `selectstatement`. The default value is **`now`**. This applies only to the first execution of the `selectstatement`. When `deduplicate` is configured, this applies to every execution of `selectstatement`.
`deduplicate`	Specifies to remove all rows from a result set that are duplicates of rows received in the prior result set. Invalid when `inrangestartrel` is not configured or `norange` is configured.

Using the QuasarDB Adapter

Syntax

dfesp_quasar_adapter -C *key1=val1,key2=val2,key3=val3,...*

Subscriber Usage

Table 136 *Required Key-Value Pairs*

Key-Value Pair	Description
`type=sub`	Specifies a subscriber adapter.
`url=`*`pubsubURL`*	Specifies the standard URL in the form `dfESP://`*`host:port/project/continuousquery/window`*. Append `?snapshot=true |false`for subscribers. When `?snapshot=true`, the subscriber receives a collection of Insert events that are contained in the window at that point in time. The subscriber then receives a stream of events produced from the time of the snapshot onward. Those subsequent events can be Inserts, Updates, or Deletes. Append `?rmretdel=true | false` for subscribers if needed.
`qdbhostport=`*`host:port`*	Specifies the QuasarDB cluster host and port, in the form "*`host:port`*".
`desttablename=`*`table`*	Specifies the subscriber target database table.

Table 137 *Optional Key-Value Pairs*

Key-Value Pair	Description
`gdconfig=`*`file`*	Specifies the guaranteed delivery configuration file.
`transport=native | solace | rabbitmq | kafka`	Specifies the transport type. If you specify `solace`, `rabbitmq`, or `kafka` transports instead of the default native transport, then use the required client configuration files specified in the description of the C++ `C_dfESPpubsubSetPubsubLib()` API call.

Key-Value Pair	Description
`loglevel=trace | debug | info | warn | error | fatal | off`	Sets the logging level for the adapter. This is the same range of logging levels that you can set in the `C_dfESPpubsubInit()` publish/subscribe API call and in the engine `initialize()` call. The default level is **warn**.
`logconfigfile=`*`logconfigfile`*	Specifies the log configuration file.
`tokenlocation=`*`location`*	Specifies the location of the file in the local file system that contains the OAuth token required for authentication by the publish/subscribe server.
`configfilesection=[`*`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`*` ]`.
`restartonerror= true | false`	When **true**, restarts the adapter when a fatal error is reported.
`transportconfigfile=`*`file`*	Specifies the publish/subscribe transport configuration file. For solace, the default is **./solace.cfg**. For rabbitmq, the default is **./rabbitmq.cfg**. For kafka, the default is **./kafka.cfg**. **Note:** No transport configuration file is required for native transport.
`commitrows=`*`minrows`*	Specifies the minimum number of output rows to buffer.
`commitsecs=`*`maxsecs`*	Specifies the maximum number of seconds to hold onto an incomplete commit buffer.
`qdbclusterpublickey=`*`string`*	Specifies the cluster public key.
`qdbuser=`*`string`*	Specifies the user name part of the user credentials.
`qdbuserprivatekey=`*`string`*	Specifies the private key part of the user credentials.
`qdbclusterpublickeyfile=`*`file`*	Specifies the complete path to a file that contains the cluster public key.
`qdbusercredentialsfile=`*`file`*	Specifies the complete path to a file that contains the user credentials.

Publisher Usage

Table 138 *Required Key-Value Pairs*

Key-Value Pair	Description
`type=pub`	Specifies a publisher.
`url=`*`pubsubURL`*	Specifies the standard URL in the form `dfESP://`*`host:port/project/continuousquery/window`*.
`qdbhostport=`*`host:port`*	Specifies the QuasarDB cluster host and port, in the form "*`host:port`*".
`selectstatement=`*`statement`*	Specifies the SQL statement to be executed on the source database. By default, the connector adds an `IN RANGE ()` clause to this statement. Enable the **`norange`** parameter to disable this functionality. By default, this statement is executed repeatedly at 1-second intervals with the IN RANGE parameters adjusted to obtain a continuous set of changes to the table. Configure the `repeatinterval` parameter to modify this interval.

Table 139 *Optional Key-Value Pairs*

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

Key-Value Pair	Description
`qdbuserprivatekey=`*`string`*	Specifies the private key part of the user credentials.
`qdbclusterpublickeyfile=`*`file`*	Specifies the complete path to a file that contains the cluster public key.
`qdbusercredentialsfile=`*`file`*	Specifies the complete path to a file that contains the user credentials.
`norange=true | false`	Specifies whether the configured `selectstatement` is executed without an `IN RANGE ()` clause added to it.
`inrangestartabs=`*`stamp`*	Specifies an absolute timestamp in QuasarDB syntax to use as the first value in the `IN RANGE()` clause added to the configured `selectstatement`. The default value is **`epoch`**. This applies only to the first execution of `selectstatement`.
`inrangestartrel=`*`offset`*	Specifies a negative offset in QuasarDB syntax relative to **`now`** to use as the second value in the IN RANGE() clause added to the configured `selectstatement`. The default value is **`now`**. This applies only to the first execution of the `selectstatement`. When `deduplicate` is configured, this applies to every execution of `selectstatement`.
`deduplicate=true | false`	Specifies to remove all rows from a result set that are duplicates of rows received in the prior result set. Invalid when `inrangestartrel` is not configured or `norange` is configured.

Using the RabbitMQ 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 RabbitMQ Connector

The RabbitMQ connector communicates with a RabbitMQ 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 RabbitMQ subscriber connector receives event blocks and publishes them to a RabbitMQ routing key. A RabbitMQ publisher connector reads event blocks from a dynamically created RabbitMQ queue and injects them into an event stream processing Source window.

Event blocks as transmitted through the RabbitMQ 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 RabbitMQ connector supports hot failover operation. This mode requires that you install the presence-exchange plug-in on the RabbitMQ 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 RabbitMQ 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 RabbitMQ server. The exchange takes place through the RabbitMQ 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 RabbitMQ 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 RabbitMQ server. The size of the message buffer is a required parameter for subscriber connectors.

You can configure a subscriber RabbitMQ connector to send a custom snapshot of window contents to any subscriber client. The client must have established a new connection to the RabbitMQ server. This enables late subscribers to catch up upon connecting. This functionality also requires that you install the presence-exchange plug-in on the RabbitMQ 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 RabbitMQ 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, be sure to include the path to those TLS libraries in your run-time environment.

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

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

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

The following holds when consuming from those queues:

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

When the publisher connector creates a durable receive queue with auto-delete disabled, it consumes from that queue with `noack = false` but does not explicitly acknowledge messages. This enables a rebooted 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 RabbitMQ 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 -md md5 -a -salt\
 -pass pass:'espRMQconnectorUsedByUser=rmquserid'
```

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

Table 140 *Required Parameters for Subscriber RabbitMQ Connectors*

Parameter	Description
`type`	Specifies to subscribe. Must be “`sub`”.
`rmquserid`	Specifies the user name required to authenticate the connector’s session with the RabbitMQ server.
`rmqpassword`	Specifies the password associated with `rmquserid`.
`rmqhost`	Specifies the RabbitMQ server host name.
`rmqport`	Specifies the RabbitMQ server port.
`rmqexchange`	Specifies the RabbitMQ exchange created by the connector, if nonexistent.
`rmqtopic`	Specifies the RabbitMQ routing key to which messages are published.
`rmqtype`	Specifies `binary` , `csv`, `json` , `protobuf`, or the name of a string field in the subscribed window schema.

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

Table 141 *Required Parameters for Publisher RabbitMQ Connectors*

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

Table 142 Optional Parameters for Subscriber RabbitMQ Connectors

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

Parameter	Description
`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.

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

`rmqheaders`	A comma–separated list of key value optional headers in messages sent to RabbitMQ. The default value is no headers.
`rmqssl`	Enables TLS encryption on the connection to the RabbitMQ server.
`rmqsslcacert`	When `rmqssl` is enabled, specifies the full path of the TLS CA certificate .pem file. **IMPORTANT** You cannot use escaped characters in the *path*.
`rmqsslkey`	When `rmqssl` is enabled, specifies the full path of the TLS key .pem file. **IMPORTANT** You cannot use escaped characters in the *path*.
`rmqsslcert`	When `rmqssl` is enabled, specifies the full path of the TLS certificate .pem file. **IMPORTANT** You cannot use escaped characters in the *path*.
`doubleprecision`	Specifies the number of fractional digits in the ASCII representation of a double. The default value is 6.

Parameter	Description
rmqheartbeat	Specifies the RabbitMQ heartbeat interval in seconds. The default value is 0, meaning that heartbeats are disabled.
usedeliverytagmsgid	When reading a message from RabbitMQ, copy the delivery tag that is provided by the RabbitMQ server into the resulting event block. This delivery tag is to be used when failover is enabled.
quorumqueues	When **true**, specifies to declare all queues with parameter x-queue-type = quorum. The default is that the parameter x-queue-type is not defined.
nometaqueue	When **true**, specifies to not declare the meta queue that receives requests from SAS Event Stream Processing clients that use RabbitMQ transport. The default is to declare the meta queue and consume requests from it.

Table 143 *Optional Parameters for Publisher RabbitMQ Connectors*

Parameter	Description
transactional	When rmqtype=CSV , sets the event block type to transactional . The default event block type is normal.
blocksize	When rmqtype=CSV , specifies the number of events to include in a published event block. The default value is 1.
dateformat	Specifies the format of DATE and STAMP fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (DATE) or microseconds (STAMP) since epoch. The dateformat parameter accepts any time format that is supported by the UNIX strftime function.
buspersistence	Controls both auto-delete and durable .
buspersistencequeue	Specifies the queue name used by a persistent publisher.
ignorecsvparseerrors	Specifies that when a field in an input CSV event cannot be parsed, the event is dropped, an error is logged, and publishing continues.
ignoreparseerrors	Specifies that when a field in an input CSV, JSON, or Protobuf event cannot be parsed, the event is dropped, an error is logged, and publishing continues.
protofile	Specifies the **.proto** file that contains the Google Protocol Buffers message definition used to convert event blocks to protobuf messages. When you specify this parameter, you must also specify the protomsg parameter.

Parameter	Description
	This parameter is ignored when `rmqtype` is not set to **`protobuf`** .
`protomsg`	Specifies the name of a Google Protocol Buffers message in the **`.proto`** file that you specified with the `protofile` parameter. Event blocks are converted into this message. This parameter is ignored when `rmqtype` is not set to **`protobuf`** .
`configfilesection`	Specifies the name of the section in **`/opt/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 ]`.
`csvfielddelimiter`	Specifies the character delimiter for field data in input CSV events. The default delimiter is the , character.
`noautogenfield`	Specifies that input events are missing the key field that is autogenerated by the source window. **Note:** This parameter applies only when the format of input data is CSV.
`ackwindow`	Specifies the time period (in seconds) to leave messages that are received from RabbitMQ unacknowledged. Applies only when `buspersistence = true`, when, by default, messages are never acknowledged. When configured, messages are acknowledged this number of seconds after having been received. Must be configured if `acktimer` is configured.
`acktimer`	Specifies the time interval (in seconds) for how often to check whether to send acknowledgments that are triggered by the `ackwindow` parameter. Must be configured if `ackwindow` is configured.
`publishwithupsert`	Builds events with `opcode=Upsert` instead of `Insert`.
`rmqpasswordencrypted`	Specifies that `rmqpassword` is encrypted.
`addcsvopcode`	Prepends an opcode and comma to CSV input events. The opcode is Insert unless `publishwithupsert` is enabled.
`addcsvflags`	Specifies the event type to insert into input CSV events (with a comma). Valid values are `normal` and `partialupdate` .
`rmqvhost`	Specifies the RabbitMQ vhost. The default is `/`.

Parameter	Description
useclientmsgid	If the Source window has been restored from a persist to disk, ignores received binary event blocks that contain a message ID less than the greatest message ID in the restored window.
rmqssl	Enables TLS encryption on the connection to the RabbitMQ server.
rmqsslcacert	When rmqssl is enabled, specifies the full path of the TLS CA certificate .pem file.
rmqsslkey	When rmqssl is enabled, specifies the full path of the TLS key .pem file.
rmqsslcert	When rmqssl is enabled, specifies the full path of the TLS certificate .pem file.
maxevents	Specifies the maximum number of events to publish.
stripnullsfromcsv	Strip all nulls from input CSV events.
rmqheartbeat	Specifies the RabbitMQ heartbeat interval in seconds. The default value is 0, meaning that heartbeats are disabled.
usedeliverytagmsgid	When reading a message from RabbitMQ, copy the delivery tag that is provided by the RabbitMQ server into the resulting event block. This delivery tag is to be used when failover is enabled.
buspersistencenonexclusive	When the buspersistence parameter is configured, specify **true** to consume from the queue in non-exclusive mode. The default value is **false**, to consume in exclusive mode.
quorumqueues	When **true**, specifies to declare all queues with parameter x-queue-type = quorum. The default is that the parameter x-queue-type is not defined.
nometaqueue	When **true**, specifies to not declare the meta queue that receives requests from SAS Event Stream Processing clients that use RabbitMQ transport. The default is to declare the meta queue and consume requests from it.
deadletterexchange	Specifies to declare the message queue with the configured *value* of the x-dead-letter-exchange queue parameter. By default, x-dead-letter-exchange is not configured. For more information, see the RabbitMQ documentation.
deadletterroutingkey	Specifies to declare the message queue with the configured *value* of the x-dead-letter-routing-key queue parameter. By default, x-dead-letter-routing-key is not configured.

Parameter	Description
	For more information, see the RabbitMQ documentation.
`prefetchcount`	Specifies the maximum number of unacknowledged messages to read. By default, there is no maximum.

Using the RabbitMQ Adapter

Overview

The RabbitMQ adapter supports publish and subscribe operations on a RabbitMQ 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*, ...

Subscriber Usage

Table 144 *Required Keys*

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

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

Table 145 *Optional Keys*

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

Key-Value Pairs	Description
`gdconfig= file`	Specifies the guaranteed delivery configuration file.
`transport=native | solace | tervela | rabbitmq | kafka`	Specifies the transport type. If you specify `solace`, `tervela` , `rabbitmq`, or `kafka` transports instead of the default `native` transport, then use the required client configuration files specified in the description of the C++ C_dfESPpubsubSetPubsubLib() API call.
`loglevel=trace | debug | info | warn | error | fatal | off`	Sets the logging level for the adapter. This is the same range of logging levels that you can set in the `C_dfESPpubsubInit()` publish/subscribe API call and in the engine `initialize()` call. The default level is `warn`.
`logconfigfile= file`	Specifies the log configuration file.
`tokenlocation= location`	Specifies the location of the file in the local file system that contains the OAuth token that is required for authentication by the publish/subscribe server.
`configfilesection=[ section]`	Specifies the name of the section in **`/opt/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 ]`.
`rmqpasswordencrypted=true | false`	When `true` , `rmqpassword` is encrypted.
`rmqvhost= vhost`	Specifies the RabbitMQ virtual host. The default is `/`.
`restartonerror=true | false`	When `true` , specifies to restart the adapter if a fatal error is reported.
`transportconfigfile= file`	Specifies the publish/subscribe transport configuration file. For `solace` , the default is **`./solace.cfg`**. For `tervela` , the default is **`./client.config`**. For `rabbitmq` , the default is **`./rabbitmq.cfg`**. For `kafka` , the default is **`./kafka.cfg`**. **Note:** No transport configuration file is required for native transport.
`csvincludeschema=never | once | pereventblock`	When `rmqtype=CSV` , specifies when to pass the window's serialized schema to subscriber callback. The default value is `never`.

Key-Value Pairs	Description
`csvmsgperevent=true | false`	When `true` , for CSV, specifies to send one message per event. The default is one message per transactional event block or else one message per event.
`csvmsgpereventblock=true | false`	When `true` , for CSV, specifies to send one message per event block. The default is one message per transactional event block or else one message per event.

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

`rmqheaders=` *list*	Specifies a comma-separated list of *key*: *value* optional headers in messages sent to RabbitMQ. The default value is no headers.
`rmqheartbeat=` *value*	Specifies the RabbitMQ heartbeat interval in seconds. The default value is 0, which means that heartbeats are disabled.
`usedeliverytagmsgid=true | false`	When reading a message from RabbitMQ, copy the delivery tag that is provided by the RabbitMQ server into the resulting event block. This delivery tag is to be used when failover is enabled.
`quorumqueues=true | false`	Specifies to declare all queues with parameter `x-queue-type = quorum`. The default is that the parameter `x-queue-type` is not defined.
`nometaqueue=true | false`	Specifies to not declare the `meta` queue that receives requests from SAS Event Stream Processing clients that use RabbitMQ transport. The default is to declare the `meta` queue and consume requests from it.

Publisher Usage

Table 146 *Required Keys*

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

Table 147 *Optional Keys*

Key-Value Pair	Description
`rmqssl=true | false`	When `true` , enables TLS encryption on the connection to the RabbitMQ server.
`rmqsslcacert=` *`path`*	When `rmqssl` is `true`, specifies the full path of the TLS CA certificate .pem file.
`buspersistence=true | false`	When `true` , use durable queues and persistent messages.
`dateformat=` *`format`*	Specifies the format of `DATE` and `STAMP` fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (`DATE`) or microseconds (`STAMP`) since epoch. The `dateformat`

Key-Value Pair	Description
	parameter accepts any time format that is supported by the UNIX `strftime` function.
`protofile=` *file*	Specifies the **`.proto`** file to be used for Google protocol buffer support. This key is ignored when `rmqtype` is not set to **`protobuf`** .
`protomsg=` *message*	Specifies the message itself in the **`.proto`** file that is specified by the protofile parameter. This key is ignored when `rmqtype` is not set to **`protobuf`** .
`gdconfig=` *file*	Specifies the guaranteed delivery configuration file.
`transport=native | solace | tervela | rabbitmq | kafka`	Specifies the transport type. If you specify `solace`, `tervela` , `rabbitmq`, or `kafka` transports instead of the default `native` transport, then use the required client configuration files specified in the description of the C++ C_dfESPpubsubSetPubsubLib() API call.
`loglevel=trace | debug | info | warn | error | fatal | off`	Sets the logging level for the adapter. This is the same range of logging levels that you can set in the `C_dfESPpubsubInit()` publish/subscribe API call and in the engine `initialize()` call. The default level is `warn`.
`logconfigfile=` *file*	Specifies the log configuration file.
`tokenlocation=` *location*	Specifies the location of the file in the local file system that contains the OAuth token that is required for authentication by the publish/subscribe server.
`configfilesection=[` *section*`]`	Specifies the name of the section in **`/opt/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*`]`.
`rmqpasswordencrypted=true | false`	When `true` , `rmqpassword` is encrypted.
`rmqvhost=` *vhost*	Specifies the RabbitMQ virtual host. The default is `/`.
`restartonerror=true | false`	When `true` , specifies to restart the adapter if a fatal error is reported.
`transportconfigfile=` *file*	Specifies the publish/subscribe transport configuration file. For `solace` , the default is **`./solace.cfg`**. For `tervela` , the default is **`./client.config`**.

Key-Value Pair	Description
	For `rabbitmq` , the default is **`./rabbitmq.cfg`**. For `kafka` , the default is **`./kafka.cfg`**. **Note:** No transport configuration file is required for native transport.
`blocksize=` *size*	Sets the block size. The default value is 1.
`transactional=true | false`	When `true` , event blocks are transactional. By default, event blocks are normal.
`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.
`ignoreparseerrors=true | false`	Specifies that when a field in an input CSV, JSON, or Protobuf 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 RabbitMQ when `buspersistence` is `true` .
`acktimer=` *seconds*	When `buspersistence` is `true`, specifies the time interval for how often to check whether to send acknowledgments that are triggered by the `ackwindow` parameter.
`csvfielddelimiter=` *delimeter*	Specifies the character delimiter for field data in input CSV events. The default delimiter is the `,` character.
`noautogenfield=true | false`	When `true` , specifies that input events are missing the key field that is autogenerated by the Source window. **Note:** This parameter applies only when the format of input data is CSV.
`publishwithupsert=true | false`	When `true` , build events with opcode = Upsert instead of Insert.
`addcsvopcode= true | false`	When `true` , prepends an opcode and comma to write CSV events. The opcode is Insert unless `publishwithupsert` is `true` .
`addcsvflags=none | normal | partialupdate`	Specifies the event type to Insert into input CSV events (with comma).
`maxevents=` *number*	Specifies the maximum number of events to publish.

Key-Value Pair	Description
`quiesceproject=true | false`	When `true` , quiesces the project after all events are injected into the Source window.
`stripnullsfromcsv=true | false`	When `true` , strip all nulls from input CSV events.
`rmqheartbeat= value`	Specifies the RabbitMQ heartbeat interval in seconds. The default value is 0, which means that heartbeats are disabled.
`usedeliverytagmsgid=true | false`	When reading a message from RabbitMQ, copy the delivery tag that is provided by the RabbitMQ server into the resulting event block. This delivery tag is to be used when failover is enabled.
`buspersistencenonexclusive=true | false`	When the `buspersistence` parameter is configured, specify **`true`** to consume from the queue in non-exclusive mode. The default value is **`false`**, to consume in exclusive mode.
`quorumqueues=true | false`	Specifies to declare all queues with parameter `x-queue-type = quorum`. The default is that the parameter `x-queue-type` is not defined.
`nometaqueue=true | false`	Specifies to not declare the `meta` queue that receives requests from SAS Event Stream Processing clients that use RabbitMQ transport. The default is to declare the `meta` queue and consume requests from it.
`deadletterexchange=value`	Specifies to declare the message queue with the configured *value* of the `x-dead-letter-exchange` queue parameter. By default, `x-dead-letter-exchange` is not configured. For more information, see the RabbitMQ documentation.
`deadletterroutingkey=value`	Specifies to declare the message queue with the configured *value* of the `x-dead-letter-routing-key` queue parameter. By default, `x-dead-letter-routing-key` is not configured. For more information, see the RabbitMQ documentation.
`prefetchcount=number`	Specifies the maximum number of unacknowledged messages to read. By default, there is no maximum.

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 `*`"`. Conversely, 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 *`esprespurl`* 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 ,...*

IMPORTANT The default values of Boolean parameters for this adapter are **`false`**. You cannot manually reset a Boolean parameter from **`true`** to **`false`** with `-C`. To reset the value, simply remove the parameter from the command line.

Table 148 *Required Keys*

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

Table 149 *Optional Keys*

Key-Value Pair	Description
`configfilesection= name`	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.
`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.
`espresurl= url`	Specifies the publish/subscribe standard URL to which responses should be published.
`gdconfigfile= file`	Specifies the guaranteed delivery configuration file.
`ignorefailedhttpconnects=true|false`	When true, drop the subscribed event and continue when an HTTP connect fails. The default value is false.
`opaquejson= field`	Specifies a subscribed window field from which to extract the complete JSON request. This is in contrast to building the request from all window fields. The field must have type ESP_UTF8STR.
`pubsublib= native | solace | tervela | rabbitmq | kafka`	Specifies the transport type. When you specify the `solace`, `tervela`, `rabbitmq`, or `kafka` transports instead of the default `native` transport, use the required client configuration files specified at "Using the Java Publish/

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

Key-Value Pair	Description
`bufsize=` *mb*	Specifies the size (in megabytes) of the buffered input-output stream that is associated with the socket connection to the ESP server.
`maxqueuesize=`*size*	Specifies the maximum *size* of the ESP server event block queue, the adapter event block queue, and the adapter Rest request queue. Specify an integer that represents the number of elements in the queue. The default value is unbounded.
`respopaquejson=` *field*	When `esprespurl` is configured, specifies the string field in the target Source window into which the JSON response should be copied. Otherwise, the JSON response is parsed.
`respuseinsert = true | false`	When `esprespurl` is configured, specifies to create events with opcode Insert instead of Upsert.
`intelligentdecisioningformat=true|false`	When **`true`**, send POST data in a JSON format that is expected by SAS Intelligent Decisioning for SAS Micro Analytic Service. The default value is **`false`**. **Note:** This key requires the use of SAS Intelligent Decisioning 5.4 or later.

When `intelligentdecisioningformat=false`, POST data looks something like this.

```
POST 1
{
    "DateTime": 1500323064998000,
    "Double": 1.1415,
    "String": "test string 2",
    "Int32": 22,
    "Int64": 222
}
POST 2
{
    "DateTime": 1500323064998000,
    "Double": 2.1415,
    "String": "test string 3",
    "Int32": 33,
    "Int64": 333
}
POST 3
{
    "DateTime": 1499423054998000,
    "Double": 0.1415,
    "String": "test string 1",
    "Int32": 11,
    "Int64": 111
}
```

When `intelligentdecisioningformat=true`, POST data looks something like this.

```
POST 1
{
    "inputs": [{
        "name:": "DateTime",
        "value": 1500323064998000
    }, {
        "name:": "Double",
        "value": 1.1415
    }, {
        "name:": "String",
        "value": "test string 2"
    }, {
        "name:": "Int32",
        "value": 22
    }, {
        "name:": "Int64",
        "value": 222
    }]
}
POST 2
{
    "inputs": [{
        "name:": "DateTime",
        "value": 1499423054998000
    }, {
        "name:": "Double",
        "value": 0.1415
    }, {
        "name:": "String",
        "value": "test string 1"
    }, {
        "name:": "Int32",
        "value": 11
    }, {
        "name:": "Int64",
        "value": 111
    }]
}
POST 3
{
    "inputs": [{
        "name:": "DateTime",
        "value": 1500323064998000
    }, {
        "name:": "Double",
        "value": 2.1415
    }, {
        "name:": "String",
        "value": "test string 3"
    }, {
        "name:": "Int32",
        "value": 33
    }, {
        "name:": "Int64",
        "value": 333
    }]
}
```

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 -md md5 -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 ,...*

> **IMPORTANT** The default values of Boolean parameters for this adapter are **`false`**. You cannot manually reset a Boolean parameter from **`true`** to **`false`** with `-C`. To reset the value, simply remove the parameter from the command line.

Table 150 *Required Keys*

Key-Value Pair	Description
`wssurl= url`	Specifies the SAS Workspace Server connection URL in the form "*host :port*" .
`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* ? snapshot=true | false<?collapse=true | false><? rmretdel=true | false>". **Note:** Multiple URLs, separated by commas, are permitted.
`type=sub`	Specifies a subscriber adapter.

Key-Value Pair	Description
`username=` *name*	Specifies the SAS Workspace Server user name.
`password=` *password*	Specifies the user password to access the SAS Workspace Server.

Table 151 *Optional Keys*

Key-Value Pair	Description
`obstoanalyze=` *number*	Specifies the number of observations to analyze to define the output SAS data set structure. Default value is 4096.
`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.
`gdconfigfile=` *file*	Specifies the guaranteed delivery configuration file.
`pubsublib= native | solace | tervela | rabbitmq | kafka`	Specifies the transport type. When you specify the `solace`, `tervela`, `rabbitmq`, or `kafka` transports instead of the default `native` transport, use the required client configuration files specified at "Using the Java Publish/Subscribe API" in *SAS Event Stream Processing: Publish/Subscribe API Reference* .
`loglevel=severe | warning | info`	Specifies the application logging level. The default value is `warning` .
`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*	Specifies the output data set schema in the following form: `" rowname1:sastype1 :sasformat1 :label1,...,rownameN:sastypeN :sasformatN :labelN"`

Key-Value Pair	Description
	If you omit *sasformatX*, then no SAS format is applied to the row. If you omit *labelX*, then no label is applied to the row. 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. Labels do not need to be wrapped in quotation marks. See the example here: `-s "ID:num,OUTPUT_DTTM:num:DATETIME20.:Output Datetime,VIN:char(17)"` **Note:** If you had previously specified the `-a` option, this option is ignored.
`restartonerror=true | false`	When true, restarts the adapter if a fatal error is reported. The default value is false.
`unencryptedpassword=true | false`	When true, the configured password is not encrypted. The default value is false.
`bufsize=` *mb*	Specifies the size (in megabytes) of the buffered input-output stream that is associated with the socket connection to the ESP server.
`transportconfigfile=` *file*	Specifies the full path to the publish/subscribe transport configuration file. The default file depends on the transport type specified with through the `pubsublib` parameter: For `solace` , the default file is **`solace.cfg`**. For `tervela` , the default file is **`client.config`**. For `rabbitmq` , the default file is **`rabbitmq.cfg`**. For `kafka` , the default file is **`kafka.cfg`**. **Note:** No transport configuration file is required for native transport.

Publisher usage:

dfesp_dataset_adapter -C *key1* =*val1*, *key2* =*val2*, *key3* =*val3* , ...

Table 152 *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* ". **Note:** 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 153 *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. The default value is false, and event blocks are normal.
`gdconfigfile= file`	Specifies the guaranteed delivery configuration file.
`pubsublib= native | solace | tervela | rabbitmq | kafka`	Specifies the transport type. When you specify the `solace`, `tervela`, `rabbitmq`, or `kafka` transports instead of the default `native` transport, use the required client configuration files specified at "Using the Java Publish/Subscribe API" in *SAS Event Stream Processing: Publish/Subscribe API Reference* .

Key-Value Pair	Description
`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"`. **Note:** For *dataset*, specify the *name* 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. The default value is false.
`publishwithupsert=true | false`	When true, build events with `opcode=Upsert`. The default value is false, and events are built with `opcode=Insert`. **`javaadapters.config`** parameter name: `publishwithupsert`
`restartonerror=true | false`	When true, restarts the adapter when a fatal error is reported. The default value is false. **`javaadapters.config`** parameter name: `restartonerror`
`unencryptedpassword=true | false`	When true, the configured password is not encrypted. The default value is false.
`maxevents=` *`number`*	Specifies the maximum number of events to publish.
`bufsize=` *`mb`*	Specifies the size (in megabytes) of the buffered input-output stream that is associated with the socket connection to the ESP server.
`transportconfigfile=` *`file`*	Specifies the full path to the publish/subscribe transport configuration file. The default file depends on the transport type specified with through the `pubsublib` parameter: For `solace` , the default file is **`solace.cfg`**. For `tervela` , the default file is **`client.config`**. For `rabbitmq` , the default file is **`rabbitmq.cfg`**. For `kafka` , the default file is **`kafka.cfg`**. **Note:** No transport configuration file is required for native transport.

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

IMPORTANT This functionality is no longer supported or included in SAS Viya 3.5 revision 24w44. No updates to this functionality are available for earlier releases.

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 154 *Required Parameters for the SMTP Connector*

Parameter	Description
`type`	Specifies to subscribe. Must be "`sub`".
`smtpserver`	Specifies the SMTP server host name or IP address.
`sourceaddress`	Specifies the email address to be used in the "from" field of the email.
`destaddress`	Specifies the email address to which to send the email message.

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

Table 155 *Optional Parameters for SMTP Connectors*

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

Using the SMTP Subscriber Adapter

IMPORTANT This functionality is no longer supported or included in SAS Viya 3.5 revision 24w44. No updates to this functionality are available for earlier releases.

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, ...*

Table 156 *Required Keys*

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

Table 157 *Optional Keys*

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

Key-Value Pair	Description
`logconfigfile= file`	Specifies the log configuration file.
`tokenlocation= location`	Specifies the location of the file in the local file system that contains the OAuth token that is required for authentication by the publish/subscribe server.
`configfilesection=[ section]`	Specifies the name of the section in **`/opt/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 ]`.
`emailperevent=true | false`	When `true` , send an email message for every event instead of one per event block.
`restartonerror=true | false`	When `true` , specifies to restart the adapter if a fatal error is reported.
`transportconfigfile= file`	Specifies the publish/subscribe transport configuration file. For `solace` , the default is **`./solace.cfg`**. For `tervela` , the default is **`./client.config`**. For `rabbitmq` , the default is **`./rabbitmq.cfg`**. For `kafka` , the default is **`./kafka.cfg`**. **Note:** No transport configuration file is required for native transport.
`dateformat= format`	Specifies the format of `DATE` and `STAMP` fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (`DATE`) or microseconds (`STAMP`) since epoch. The `dateformat` parameter accepts any time format that is supported by the UNIX `strftime` function.
`doubleprecision= number`	Specifies the number of fractional digits in the ASCII representation of a double. The default value is 6.
`connectionpereventblock=true | false`	When true, reconnect and disconnect to and from the SMTP server for every event block. When false, maintain a persistent connection to the SMTP server. The default value is false.

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

> **IMPORTANT** This functionality is no longer supported or included in SAS Viya 3.5 revision 24w44. No updates to this functionality are available for earlier releases.

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

Note: To use this connector:

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

Protocol support is currently limited to the following:

- HTTP packets sent to port 80 over TCP. To capture HTTP packets that you send to other ports (for example, 8080), configure the list of ports in 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` or `payload:blob`

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 158 *Required Parameters for the Sniffer Connector*

Parameter	Description
`type`	Specifies to publish. Must be "`pub`".
`interface`	Specifies the name of the network interface on the local machine from which to capture packets.
`protocol`	Specifies the port number associated with the protocol type of packets to be captured. You can specify this as a comma-separated list of port numbers.

`packetfields`	Specifies the packet fields to be extracted from a captured packet and included in the published event. Use SAS Event Stream Processing schema syntax. This value does not include the `index:int64` `frame_time:stamp`, or `ts:stamp` fields.
	Separate nested fields with an underscore character. Match field names to standard names as displayed by the Wireshark open-source packet analyzer. You must remove spaces and hyphens to maintain field name integrity.
	An example for Radius is as follows:
	`radius_AcctStatusType:int32,radius_EventTimestamp:stamp,radius_FramedIPAddress:string,radius_UserName:string .`
	An example for HTTP is as follows:
	`ip_Source:string,ip_Destination:string,http_Host:string,http_Referer:string,http_UserAgent:string,http_GET_Request URI:string .`
	An example for TLS is as follows:
	`ssltls_Handshake_ClientHello_GMTUnixTime:date,ssltls_Handshake_ClientHello_SessionID:string,ssltls_Handshake_ClientHello_CypherSuites:string,ssltls_Handshake_ClientHello_CompressionMethods:string,ssltls_Handshake_ClientHello_Extensions_ServerName:string .`
	If `protocol` is not set to 80, 1813, or 443 and `httpports` is not configured, the only supported values are as follows:
	■ the IP source and destination address
	■ the TCP or UDP source and destination ports
	■ `payload:string` or `payload:blob`

Table 159 *Optional Parameters for the Sniffer Connector*

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

Using the Sniffer Publisher Adapter

IMPORTANT This functionality is no longer supported or included in SAS Viya 3.5 revision 24w44. No updates to this functionality are available for earlier releases.

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, ...*

Table 160 *Required Keys*

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

Table 161 *Optional Keys*

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

Key-Value Pair	Description
	`C_dfESPpubsubInit()` publish/subscribe API call and in the engine `initialize()` call. The default level is `warn`.
`logconfigfile=` *file*	Specifies the log configuration file.
`tokenlocation=` *location*	Specifies the location of the file in the local file system that contains the OAuth token that is required for authentication by the publish/subscribe server.
`configfilesection=[` *section*`]`	Specifies the name of the section in **`/opt/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*`]`.
`blocksize=` *size*	Sets the block size. The default value is 1.
`transactional=true | false`	When `true`, event blocks are transactional. By default, event blocks are normal.
`addtimestamp= true | false`	When true, appends an `ESP_TIMESTAMP` field to each published event.
`vendorid=` *field*	Specifies the vendor ID field to match when capturing the attribute-specific field in a vendor-specific attribute in a RADIUS Accounting-Request packet.
`vendortype=` *field*	Specifies the vendor type field to match when capturing the attribute-specific field in a vendor-specific attribute in a RADIUS Accounting-Request packet.
`indexfieldname=` *name*	Specifies the name to use instead of `index` for the `index:int65` field in the Source window schema.
`publishwithupsert=true | false`	When `true`, build events with opcode = Upsert instead of Insert.
`pcapfilter=` *expression*	Specifies a filter expression as defined in the pcap documentation. The value is passed to the pcap driver in order to filter packets received by the adapter.
`httpports=` *list*	Specifies a comma-separated list of destination ports. All sniffed packets that contain a specified port are parsed for HTTP GET parameters. The default value is 80.

Key-Value Pair	Description
`ignorenonpayloadpackets=true | false`	When `true` , specifies to ignore packets that have no payload, as calculated by subtracting the TCP or UDP header size from the packet size.
`restartonerror=true | false`	When `true` , specifies to restart the adapter if a fatal error is reported.
`transportconfigfile=` *file*	Specifies the publish/subscribe transport configuration file. For `solace` , the default is **`./solace.cfg`**. For `tervela` , the default is **`./client.config`**. For `rabbitmq` , the default is **`./rabbitmq.cfg`**. For `kafka` , the default is **`./kafka.cfg`**. **Note:** No transport configuration file is required for native transport.
`maxevents=` *number*	Specifies the maximum number of events to publish.
`quiesceproject=true | false`	When `true` , quiesces the project after all events are injected into the Source window.

For information about implementing hot failover for publisher adapters, see "Publisher Adapter Failover with Kafka" 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 `buspersistenceque` queue configured on the publisher connector (for "/I" topics), or the queue configured on the client subscriber (for "/O" topics). Add the corresponding topic to each configured queue.
- When `buspersistence` is enabled or hot failover is enabled on subscriber connectors, you must enable "Allow Guaranteed Endpoint Create", "Allow Guaranteed Message Send", and "Allow Guaranteed Message Receive" in your client profile.

When the connector starts, it subscribes to topic *`urlhostport`* `/M` (where *urlhostport* is a connector configuration parameter). This enables the connector to receive metadata requests from clients that publish or subscribe to a window in an 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 -md md5 -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 162 *Required Parameters for Subscriber Solace Systems Connectors*

Parameter	Description
`type`	Specifies to subscribe. Must be "`sub`".

Parameter	Description
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	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 163 *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.

Parameter	Description
urlhostport	Specifies the *host*: *port* field in the metadata topic subscribed to on start-up to field metadata requests.

Table 164 *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.

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

Table 165 *Optional Parameters for Publisher Solace Systems Connectors*

Parameter	Description
buspersistence	Creates the Guaranteed message flow to bind to the topic endpoint provisioned on the appliance that the published Guaranteed messages are delivered and spooled to. By default this flow is disabled, because it is not required to receive messages published using Direct Messaging.
buspersistencequeue	Specifies the name of the queue to which the Guaranteed message flow binds.
protofile	Specifies the `.proto` file that contains the Google Protocol Buffers message definition used to convert event blocks to `protobuf` messages. When you specify this parameter, you must also specify the `protomsg` parameter.
protomsg	Specifies the name of a Google Protocol Buffers message in the `.proto` file that you specified with the `protofile` parameter. Event blocks are converted into this message.
configfilesection	Specifies the name of the section in **/opt/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.
publishwithupsert	Specifies to build with `opcode = Upsert` instead of `opcode = Insert .`
dateformat	Specifies the format of `DATE` and `STAMP` fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (`DATE`) or microseconds (`STAMP`) since epoch. The `dateformat` parameter accepts any time format that is supported by the UNIX `strftime` function.
solpasswordencrypted	Specifies that `solpassword` is encrypted.

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

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, ...*

Subscriber Usage

Table 166 *Required Keys*

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

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

Table 167 *Optional Keys*

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

Key-Value Pair	Description
	\SASEventStreamProcessingEngine\default \connectors.config (Windows) to parse for configuration parameters. Specify the value as `[configfilesection ]`.
`dateformat= format`	Specifies the format of `DATE` and `STAMP` fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (`DATE`) or microseconds (`STAMP`) since epoch. The `dateformat` parameter accepts any time format that is supported by the UNIX `strftime` function.
`solpasswordencrypted=true | false`	When true, the Solace password is encrypted.
`doubleprecision=number`	Specifies the number of fractional digits in the ASCII representation of a double. The default value is 6.
`restartonerror=true | false`	When `true` , specifies to restart the adapter if a fatal error is reported.
`transportconfigfile= file`	Specifies the publish/subscribe transport configuration file. For `solace` , the default is **./solace.cfg**. For `tervela` , the default is **./client.config**. For `rabbitmq` , the default is **./rabbitmq.cfg**. For `kafka` , the default is **./kafka.cfg**. **Note:** No transport configuration file is required for native transport.

Publisher Usage

Table 168 *Required Keys*

Key-Value Pair	Description
`type=pub`	Specifies a publisher adapter.
`url= pubsubURL`	Specifies the standard URL in the form **dfESP://** ***host :port/ project/continuousquery /window***.
`soluserid= username`	Specifies the Solace user name.
`solpassword= password`	Specifies the Solace password.
`solhostport= hostport`	Specifies the Solace *host:port* .

Key-Value Pair	Description
`solvpn= vpn`	Specifies the Solace VPN name.
`soltopic= topic`	Specifies the Solace topic.
`urlhostport= hostport`	Specifies the *host:port* field in the metadata topic subscribed to by the connector.

Table 169 *Optional Keys*

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

Key-Value Pair	Description
	\connectors.config (Windows) to parse for configuration parameters. Specify the value as [*configfilesection*].
dateformat= *format*	Specifies the format of DATE and STAMP fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (DATE) or microseconds (STAMP) since epoch. The dateformat parameter accepts any time format that is supported by the UNIX strftime function.
solpasswordencrypted=true | false	When true, the Solace password is encrypted.
restartonerror=true | false	When true , specifies to restart the adapter if a fatal error is reported.
transportconfigfile= *file*	Specifies the publish/subscribe transport configuration file. For solace , the default is **./solace.cfg**. For tervela , the default is **./client.config**. For rabbitmq , the default is **./rabbitmq.cfg**. For kafka , the default is **./kafka.cfg**. **Note:** No transport configuration file is required for native transport.
buspersistencequeue= *queue*	Specifies the queue name used by Solace Guaranteed Messaging publisher.
publishwithupsert=true | false	When true , build events with opcode = Upsert instead of Insert.
blocksize= *size*	Sets the block size. The default value is 1.
transactional=true | false	When true , event blocks are transactional. By default, event blocks are normal.
getmsgfromdestattr=true | false	When true , extracts the payload from the destination attribute instead of the message body.
maxevents= *number*	Specifies the maximum number of events to publish.
quiesceproject=true | false	When true , quiesces the project after all events are injected into the Source window.

Publisher Failover

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

IMPORTANT This functionality is no longer supported or included in SAS Viya 3.5 revision 24w44. No updates to this functionality are available for earlier releases.

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

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

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

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

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

If required, you can configure the `tdatauserpwd` parameter with an encrypted password. The encrypted version of the password can be generated by using OpenSSL, which must be installed on your system. If you have installed the 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 'tdatapwd' | openssl enc -e -aes-256-cbc -md md5\
 -a -salt -pass pass:'espTDATAconnectorUsedByUser=tdatauid'
```

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

Table 170 *Required Parameters for Teradata Connectors*

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

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

Table 171 *Optional Parameters for Teradata Connectors*

Parameter	Description
rmretdel	Removes all delete events from event blocks received by the subscriber that were introduced by a window retention policy.
tdatabatchperiod	Specifies the batch period in seconds. Required when using the `update` operator, otherwise ignored.
stage1tablename	Specifies the first staging table. Required when using the `load` operator, otherwise ignored.
stage2tablename	Specifies the second staging table. Required when using the `load` operator, otherwise ignored.
connectstring	Specifies the connect string used to access the target and staging tables. See "Using the Database Connector" for Teradata `connectstring` requirements.
tdatatracelevel	Specifies the trace level for Teradata messages written to the trace file in the current working directory. The trace file is named *operator* 1.txt. Default is 1 (TD_OFF). Other valid values are: 2

Parameter	Description
	(TD_OPER), 3 (TD_OPER_CLI), 4 (TD_OPER_OPCOMMON), 5 (TD_OPER_SPECIAL), 6 (TD_OPER_ALL), 7 (TD_GENERAL), and 8 (TD_ROW).
`configfilesection`	Specifies the name of the section in **`/opt/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`*` ]`.
`tdatauserpwdencrypted`	Specifies that `tdatauserpwd` is encrypted.

Using the Teradata Subscriber Adapter

IMPORTANT This functionality is no longer supported or included in SAS Viya 3.5 revision 24w44. No updates to this functionality are available for earlier releases.

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*, ...

Table 172 *Required Keys*

Key-Value Pair	Description
`url=` *pubsubURL*	Specifies the standard URL in the form **`dfESP://`*`host`*`:`*`port`*`/`*`project`*`/`*`continuousquery`*`/`*`window`*. Append `?snapshot=true` |`false`. Append`?collapse=true` | `false` or `?rmretdel=true` | `false` or both if needed.
`tdatadriver=stream` | `update` | `load`	Specifies the Teradata operator. For more information about these operators, see the documentation provided with the Teradata Tools and Utilities.

Key-Value Pair	Description
`tdatausername= username`	Specifies the Teradata user name.
`tdatauserpwd= password`	Specifies the Teradata user password.
`desttablename= name`	Specifies the name of the target table on the Teradata server.
`tdatatdpid= name`	Specifies the name of the target Teradata server.
`tdatamaxsessions= number`	Specifies the maximum number of sessions on the Teradata server. A Teradata session is a logical connection between an application and Teradata Database. It allows an application to send requests to and receive responses from Teradata Database. A session is established when Teradata Database accepts the user name and password of a user.
`tdataminsessions= number`	Specifies the minimum number of sessions on the Teradata server.
`tdatainsertonly=true | false`	When `true` , specifies that the subscribed window contains insert-only data.

Table 173 *Optional Keys*

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

Key-Value Pair	Description
configfilesection=[*section*]	Specifies the name of the section in **/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config** (Linux) or **%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.config** (Windows) to parse for configuration parameters. Specify the value as [*configfilesection*].
tdatabatchperiod= *seconds*	Specifies the batch period in seconds. This parameter is required when tdatadriver=update .
stage1tablename= *name*	Specifies the name of the first staging table on the Teradata server. This parameter is required when tdatadriver=load.
stage2tablename= *name*	Specifies the name of the second staging table on the Teradata server. This parameter is required when tdatadriver=load.
connectstring= *string*	Specifies the connect string to be used by the ODBC driver to access the staging and target tables on the Teradata server. The string takes the form "DSN=*dsn*[;uid= *uid*] [;pwd=*pwd*]". This parameter is required when tdatadriver=load.
tdatatracelevel= *level*	Specifies the trace level for Teradata messages written to the trace file in the current working directory. The trace file is named *operator*1.txt. Default value is 1 (TD_OFF). Other valid values are: 2 (TD_OPER), 3 (TD_OPER_CLI), 4 (TD_OPER_OPCOMMON), 5 (TD_OPER_SPECIAL), 6 (TD_OPER_ALL), 7 (TD_GENERAL), and 8 (TD_ROW).
tdatauserpwdencrypted=true | false	When true , tdatauserpwd is encrypted.
restartonerror=true | false	When true , specifies to restart the adapter if a fatal error is reported.
transportconfigfile= *file*	Specifies the publish/subscribe transport configuration file. For solace , the default is **./solace.cfg**. For tervela , the default is **./client.config**. For rabbitmq , the default is **./rabbitmq.cfg**. For kafka , the default is **./kafka.cfg**.

Key-Value Pair	Description
	Note: No transport configuration file is required for native transport.

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

IMPORTANT This functionality is no longer supported or included in SAS Viya 3.5 revision 24w44. No updates to this functionality are available for earlier releases.

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 `tvauserid` and `tvapassword` configuration parameters
- the inbound and outbound topic strings and associated schema
- publish or subscribe entitlement rights associated with a client user name

When the connector starts, it publishes a message to topic SAS.META.*tvaclientname* (where *tvaclientname* is a connector configuration parameter). This message contains the following information:

- The mapping of the 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 -md md5\
-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 174 *Required Parameters for Subscriber Tervela Connectors*

Parameter	Description
`type`	Specifies to subscribe. Must be "`sub`".
`tvauserid`	Specifies a user name defined in the Tervela TPM. Publish-topic entitlement rights must be associated with this user name.
`tvapassword`	Specifies the password associated with `tvauserid`.
`tvaprimarytmx`	Specifies the host name or IP address of the primary TMX.
`tvatopic`	Specifies the topic name for the topic to which to subscribed. This topic must be configured on the TPM for the GD service and `tvauserid` must be assigned the Guaranteed Delivery subscribe rights for this Topic in the TPM.
`tvaclientname`	Specifies the client name associated with the Tervela Guaranteed Delivery context. If hot failover is enabled, this name must match the `tvaclientname` of other subscriber connectors in the failover group. Otherwise, the name must be unique among all instances of Tervela connectors.
`tvamaxoutstand`	Specifies the maximum number of unacknowledged messages that can be published to the Tervela fabric (effectively the size of the publication cache). Should be twice the expected transmit rate.
`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

Parameter	Description
	flushed, and buffered messages are sent to the fabric as required to maintain message ID sequence.
`urlhostport`	Specifies the *`host/ port`* string sent in the metadata message published by the connector on topic SAS.META. *tvaclientname* when it starts.
`snapshot`	Specifies whether to send snapshot data. When `true`, the subscriber receives a collection of Insert events that are contained in the window at that point in time. The subscriber then receives a stream of events produced from the time of the snapshot onward. Those subsequent events can be Inserts, Updates, or Deletes.

Table 175 *Required Parameters for Publisher Tervela Connectors*

Parameter	Description
`type`	Specifies to publish. Must be "`pub`".
`tvauserid`	Specifies a user name defined in the Tervela TPM. Subscribe-topic entitlement rights must be associated with this user name.
`tvapassword`	Specifies the password associated with `tvauserid`.
`tvaprimarytmx`	Specifies the host name or IP address of the primary TMX.
`tvatopic`	Specifies the topic name for the topic to which to publish. This topic must be configured on the TPM for the GD service.
`tvaclientname`	Specifies the client name associated with the Tervela Guaranteed Delivery context. Must be unique among all instances of Tervela connectors.
`tvasubname`	Specifies the name assigned to the Guaranteed Delivery subscription being created. The combination of this name and `tvaclientname` are used by the fabric to replay the last subscription state. If a subscription state is found, it is used to resume the subscription from its previous state. If not, the subscription is started new, starting with a replay of messages received in the past eight hours.
`urlhostport`	Specifies the *`host: port`* string sent in the metadata message published by the connector on topic SAS.META. *tvaclientname* when it starts.

Table 176 *Optional Parameters for Subscriber Tervela Connectors*

Parameter	Description
`collapse`	Enables conversion of UPDATE_BLOCK events to make subscriber output publishable.
`hotfailover`	Enables hot failover mode.
`tvasecondarytmx`	Specifies the host name or IP address of the secondary TMX. Required if logging in to a fault-tolerant pair.
`tvalogfile`	Causes the connector to log to the specified file instead of to syslog (on Linux or Solaris) or Tervela.log (on Windows).
`tvapubbwlimit`	Specifies the maximum bandwidth, in Mbps, of data published to the fabric. The default is 100 Mbps.
`tvapubrate`	Specifies the rate at which data messages are published to the fabric, in Kbps. The default is 30,000 messages per second.
`tvapubmsgexp`	Specifies the maximum amount of time, in seconds, that published messages are kept in the cache in the Tervela API. This cache is used as part of the channel egress reliability window (if retransmission is required). The default value is 1 second.
`rmretdel`	Specifies to remove all delete events from event blocks received by a subscriber that were introduced by a window retention policy.
`configfilesection`	Specifies the name of the section in **`/opt/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`*` ]`.
`protofile`	Specifies the **`.proto`** file that contains the Google Protocol Buffers message definition. This definition is used to convert event blocks to protobuf messages. When you specify this parameter, you must also specify the `protomsg` parameter.
`protomsg`	Specifies the name of a Google Protocol Buffers message in the **`.proto`** file that you specified with the `protofile` parameter. Event blocks are converted into this message.
`json`	Enables transport of event blocks encoded as JSON messages.
`dateformat`	Specifies the format of `DATE` and `STAMP` fields in CSV events. The default behavior is that these fields are interpreted as an integer

Parameter	Description
	number of seconds (DATE) or microseconds (STAMP) since epoch. The dateformat parameter accepts any time format that is supported by the UNIX strftime function.
tvapasswordencrypted	Specifies that tvapassword is encrypted.

Table 177 *Optional Parameters for Publisher Tervela Connectors*

Parameter	Description
tvasecondarytmx	Specifies the host name or IP address of the secondary TMX. Required when logging in to a fault-tolerant pair.
tvalogfile	Causes the connector to log to the specified file instead of to syslog (on Linux or Solaris) or Tervela.log (on Windows)
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*].
protofile	Specifies the **.proto** file that contains the Google Protocol Buffers message definition. This definition is used to convert event blocks to protobuf messages. When you specify this parameter, you must also specify the protomsg parameter.
protomsg	Specifies the name of a Google Protocol Buffers message in the **.proto** file that you specified with the protofile parameter. Event blocks are converted into this message.
json	Enables transport of event blocks encoded as JSON messages.
publishwithupsert	Specifies to build events with opcode = Upsert instead of opcode = Insert.
dateformat	Specifies the format of DATE and STAMP fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (DATE) or microseconds (STAMP) since epoch. The dateformat parameter accepts any time format that is supported by the UNIX strftime function.
tvapasswordencrypted	Specifies that tvapassword is encrypted.
maxevents	Specifies the maximum number of events to publish.

Using the Tervela Data Fabric Adapter

Overview

IMPORTANT This functionality is no longer supported or included in SAS Viya 3.5 revision 24w44. No updates to this functionality are available for earlier releases.

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, ...*

Subscriber Usage

Table 178 *Required Keys*

Key-Value Pair	Description
`type=sub`	Specifies a subscriber adapter.
`url=` *pubsubURL*	Specifies the standard URL in the form **`dfESP://`** ***`host :port/ project/continuousquery /window`***. Append `?snapshot=true |false`. Append`?collapse=true | false` or `?rmretdel=true | false` or both if needed.
`tvauserid=` *username*	Specifies the Tervela user name.
`tvapassword=` *password*	Specifies the Tervela password.
`tvaprimarytmx=` *tmx*	Specifies the Tervela primary TMX. The TMX-500 Message Switch is the primary appliance used to communicate across the Tervela data fabric.
`tvatopic=` *topic*	Specifies the Tervela topic.
`tvaclientname=` *name*	Specifies the Tervela client name.
`urlhostport=` *string*	Specifies the *host:port* string sent in connector metadata message

Key-Value Pair	Description
`tvamaxoutstand= number`	Specifies the Tervela maximum number of unacknowledged messages.
`numbufferedmsgs= number`	Specifies the maximum number of messages buffered by a standby subscriber connector.

Table 179 *Optional Keys*

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

Key-Value Pair	Description
	configuration parameters. Specify the value as `[configfilesection ]`.
`dateformat= format`	Specifies the format of `DATE` and `STAMP` fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (`DATE`) or microseconds (`STAMP`) since epoch. The `dateformat` parameter accepts any time format that is supported by the UNIX `strftime` function.
`tvapasswordencrypted=true | false`	When `true` , `tvapassword` is encrypted
`restartonerror=true | false`	When `true` , specifies to restart the adapter if a fatal error is reported.
`transportconfigfile= file`	Specifies the publish/subscribe transport configuration file. For `solace` , the default is **`./solace.cfg`**. For `tervela` , the default is **`./client.config`**. For `rabbitmq` , the default is **`./rabbitmq.cfg`**. For `kafka` , the default is **`./kafka.cfg`**. **Note:** No transport configuration file is required for native transport.
`tvapubbwlimit= bandwidth`	Specifies the Tervela maximum bandwidth of published data (Mbps). The default value is 100.
`tvapubrate= rate`	Specifies the Tervela publish rate (Kbps). The default value is 30.
`tvapubmsgexp= seconds`	Specifies the Tervela maximum time to cache published messages (seconds); the default value is 1.

Publisher Usage

Table 180 *Required Keys*

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

Key-Value Pair	Description
`tvauserid=` *`username`*	Specifies the Tervela user name.
`tvapassword=` *`password`*	Specifies the Tervela password.
`tvaprimarytmx=` *`tmx`*	Specifies the Tervela primary TMX. The TMX-500 Message Switch is the primary appliance used to communicate across the Tervela data fabric.
`tvatopic=` *`topic`*	Specifies the Tervela topic.
`tvaclientname=` *`name`*	Specifies the Tervela client name.
`urlhostport=` *`string`*	Specifies the *host:port* string sent in connector metadata message
`tvasubname=` *`name`*	Specifies the Tervela name of the Guaranteed Delivery subscription

Table 181 *Optional Keys*

Key-Value Pair	Description
`tvasecondarytmx=` *`tmx`*	Specifies the Tervela secondary TMX.
`tvalogfile=` *`file`*	Specifies the Tervela log file. The default is "`syslog`".
`protofile=` *`file`*	Specifies the **`.proto`** file to be used for Google protocol buffer support.
`protomsg=` *`message`*	Specifies the message itself in the **`.proto`** file that is specified by the protofile parameter.
`json=true | false`	When `true` , transport JSON messages instead of event blocks.
`gdconfig=` *`file`*	Specifies the guaranteed delivery configuration file.
`transport=native | solace | tervela | rabbitmq | kafka`	Specifies the transport type. If you specify `solace`, `tervela` , `rabbitmq`, or `kafka` transports instead of the default `native` transport, then use the required client configuration files specified in the description of the C++ C_dfESPpubsubSetPubsubLib() API call.
`loglevel=trace | debug | info | warn | error | fatal | off`	Sets the logging level for the adapter. This is the same range of logging levels that you can set in the

Key-Value Pair	Description
	`C_dfESPpubsubInit()` publish/subscribe API call and in the engine `initialize()` call. The default level is `warn`.
`logconfigfile=` *file*	Specifies the log configuration file.
`tokenlocation=` *location*	Specifies the location of the file in the local file system that contains the OAuth token that is required for authentication by the publish/subscribe server.
`configfilesection=[` *section*`]`	Specifies the name of the section in **`/opt/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*`]`.
`dateformat=` *format*	Specifies the format of `DATE` and `STAMP` fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (`DATE`) or microseconds (`STAMP`) since epoch. The `dateformat` parameter accepts any time format that is supported by the UNIX `strftime` function.
`tvapasswordencrypted=true | false`	When `true`, `tvapassword` is encrypted
`restartonerror=true | false`	When `true`, specifies to restart the adapter if a fatal error is reported.
`transportconfigfile=` *file*	Specifies the publish/subscribe transport configuration file. For `solace`, the default is **`./solace.cfg`**. For `tervela`, the default is **`./client.config`**. For `rabbitmq`, the default is **`./rabbitmq.cfg`**. For `kafka`, the default is **`./kafka.cfg`**. **Note:** No transport configuration file is required for native transport.
`publishwithupsert=true | false`	When `true`, build events with opcode = Upsert instead of Insert.
`maxevents=` *number*	Specifies the maximum number of events to publish.
`quiesceproject=true | false`	When `true`, quiesces the project after all events are injected into the Source window.

Publisher Failover

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

IMPORTANT This functionality is no longer supported or included in SAS Viya 3.5 revision 24w44. No updates to this functionality are available for earlier releases.

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 182 *Required Parameters for Subscriber Tibco RV Connectors*

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

Table 183 *Required Parameters for Publisher Tibco RV Connectors*

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

Table 184 Optional Parameters for Subscriber Tibco RV Connectors

Parameter	Description
collapse	Enables conversion of UPDATE_BLOCK events to make subscriber output publishable.
tibrvservice	Specifies the Rendezvous service used by the Tibco RV transport created by the connector. The default service name is "rendezvous".
tibrvnetwork	Specifies the network interface used by the Tibco RV transport created by the connector. The default network depends on the type of daemon used by the connector.
tibrvdaemon	Specifies the Rendezvous daemon used by the connector. The default is the default socket created by the local daemon.
rmretdel	Specifies to remove all delete events from event blocks received by a subscriber that were introduced by a window retention policy.
dateformat	Specifies the format of DATE and STAMP fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (DATE) or microseconds (STAMP) since epoch. The dateformat parameter accepts any time format that is supported by the UNIX strftime function.
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*].
protofile	Specifies the **.proto** file that contains the Google Protocol Buffers message definition. This definition is used to convert event blocks to protobuf messages. When you specify this parameter, you must also specify the protomsg parameter. This parameter is ignored when tibrvtype is not set to **protobuf**.
protomsg	Specifies the name of a Google Protocol Buffers message in the **.proto** file that you specified with the protofile parameter. Event blocks are converted into this message. This parameter is ignored when tibrvtype is not set to **protobuf**.
csvmsgperevent	For CSV, specifies to send one message per event. The default is one message per transactional event block or else one message per event.

Parameter	Description
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 185 *Optional Parameters for Publisher Tibco RV Connectors*

Parameter	Description
blocksize	Specifies the number of events to include in a published event block. The default value is 1.
transactional	Sets the event block type to transactional. The default event block type is normal.
dateformat	Specifies the format of DATE and STAMP fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (DATE) or microseconds (STAMP) since epoch. The dateformat parameter accepts any time format that is supported by the UNIX strftime function.
tibrvservice	Specifies the Rendezvous service used by the Tibco RV transport created by the connector. The default service name is rendezvous.
tibrvnetwork	Specifies the network interface used by the Tibco RV transport created by the connector. The default network depends on the type of daemon used by the connector.
tibrvdaemon	Specifies the Rendezvous daemon used by the connector. The default is the default socket created by the local daemon.
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*].
ignorecsvparseerrors	Specifies that when a field in an input CSV event cannot be parsed, the event is dropped, an error is logged, and publishing continues.
protofile	Specifies the **.proto** file that contains the Google Protocol Buffers message definition. This definition is used to convert

Parameter	Description
	event blocks to protobuf messages. When you specify this parameter, you must also specify the `protomsg` parameter. This parameter is ignored when `tibrvtype` is not set to **`protobuf`**.
`protomsg`	Specifies the name of a Google Protocol Buffers message in the **`.proto`** file that you specified with the `protofile` parameter. Event blocks are converted into this message. This parameter is ignored when `tibrvtype` is not set to **`protobuf`**.
`csvfielddelimiter`	Specifies the character delimiter for field data in input CSV events. The default delimiter is the `,` character.
`noautogenfield`	Specifies that input events are missing the key field that is autogenerated by the source window. **Note:** This parameter applies only when the format of input data is CSV.
`publishwithupsert`	Specifies to build events with `opcode=Upsert` instead of `opcode=Insert` .
`addcsvopcode`	Prepends an opcode and comma to input CSV events. The opcode is Insert unless `publishwithupsert` is enabled.
`addcsvflags`	Specifies the event type to insert into input CSV events (with a comma). Valid values are `normal` and `partialupdate` .
`maxevents`	Specifies the maximum number of events to publish.
`stripnullsfromcsv`	Strip all nulls from input CSV events.

Using the Tibco Rendezvous (RV) Adapter

Overview

IMPORTANT This functionality is no longer supported or included in SAS Viya 3.5 revision 24w44. No updates to this functionality are available for earlier releases.

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*, ...

Subscriber Usage

Table 186 *Required Keys*

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

Table 187 *Optional Keys*

Key-Value Pair	Description
`tibrvservice=` *service*	Specifies the Tibco RV service. Rendezvous daemon processes communicate using UDP or PGM services.
`tibrvnetwork=` *network*	Specifies the Tibco RV network.
`tibrvdaemon=` *daemon*	Specifies the Tibco RV daemon.
`dateformat=` *format*	Specifies the format of `DATE` and `STAMP` fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (`DATE`) or microseconds (`STAMP`) since epoch. The `dateformat` parameter accepts any time format that is supported by the UNIX `strftime` function.
`protofile=` *file*	Specifies the **`.proto`** file to be used for Google protocol buffer support. This key is ignored when `tibrvtype` is not set to **`protobuf`**.
`protomsg=` *message*	Specifies the message itself in the **`.proto`** file that is specified by the protofile parameter.

Key-Value Pair	Description
	This key is ignored when `tibrvtype` is not set to **`protobuf`**.
`gdconfig=` *`file`*	Specifies the guaranteed delivery configuration file.
`transport=native | solace | tervela | rabbitmq | kafka`	Specifies the transport type. If you specify `solace`, `tervela` , `rabbitmq`, or `kafka` transports instead of the default `native` transport, then use the required client configuration files specified in the description of the C++ C_dfESPpubsubSetPubsubLib() API call.
`loglevel=trace | debug | info | warn | error | fatal | off`	Sets the logging level for the adapter. This is the same range of logging levels that you can set in the `C_dfESPpubsubInit()` publish/subscribe API call and in the engine `initialize()` call. The default level is `warn`.
`logconfigfile=` *`file`*	Specifies the log configuration file.
`tokenlocation=` *`location`*	Specifies the location of the file in the local file system that contains the OAuth token that is required for authentication by the publish/subscribe server.
`configfilesection=[` *`section`*`]`	Specifies the name of the section in **`/opt/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`*`]`.
`restartonerror=true | false`	When `true` , specifies to restart the adapter if a fatal error is reported.
`transportconfigfile=` *`file`*	Specifies the publish/subscribe transport configuration file. For `solace` , the default is **`./solace.cfg`**. For `tervela` , the default is **`./client.config`**. For `rabbitmq` , the default is **`./rabbitmq.cfg`**. For `kafka` , the default is **`./kafka.cfg`**. **Note:** No transport configuration file is required for native transport.
`csvmsgperevent=true | false`	When `true` , send one message per event. By default, one message is sent per transactional event block.
`csvmsgpereventblock=true | false`	When `true` , send one message per event block. By default, one message is sent per transactional event block.

Key-Value Pair	Description
`doubleprecision=` *number*	Specifies the number of fractional digits in the ASCII representation of a double. The default value is 6.

Publisher Usage

Table 188 *Required Keys*

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

Table 189 *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 **`protobuf`**.

Key-Value Pair	Description
protomsg= *message*	Specifies the message itself in the **.proto** file that is specified by the protofile parameter. This key is ignored when tibrvtype is not set to **protobuf**.
gdconfig= *file*	Specifies the guaranteed delivery configuration file.
transport=native | solace | tervela | rabbitmq | kafka	Specifies the transport type. If you specify solace, tervela , rabbitmq, or kafka transports instead of the default native transport, then use the required client configuration files specified in the description of the C++ C_dfESPpubsubSetPubsubLib() API call.
loglevel=trace | debug | info | warn | error | fatal | off	Sets the logging level for the adapter. This is the same range of logging levels that you can set in the C_dfESPpubsubInit() publish/subscribe API call and in the engine initialize() call. The default level is warn.
logconfigfile= *file*	Specifies the log configuration file.
tokenlocation= *location*	Specifies the location of the file in the local file system that contains the OAuth token that is required for authentication by the publish/subscribe server.
configfilesection=[*section*]	Specifies the name of the section in **/opt/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*].
restartonerror=true | false	When true , specifies to restart the adapter if a fatal error is reported.
transportconfigfile= *file*	Specifies the publish/subscribe transport configuration file. For solace , the default is **./solace.cfg**. For tervela , the default is **./client.config**. For rabbitmq , the default is **./rabbitmq.cfg**. For kafka , the default is **./kafka.cfg**. **Note:** No transport configuration file is required for native transport.
blocksize= *size*	Sets the block size. The default value is 1.
transactional=true | false	When true , event blocks are transactional. By default, event blocks are normal.

Key-Value Pair	Description
`ignorecsvparseerrors=true | false`	Specifies that when a field in an input CSV event cannot be parsed, the event is dropped, an error is logged, and publishing continues.
`csvfielddelimiter=` *`delimeter`*	Specifies the character delimiter for field data in input CSV events. The default delimiter is the `,` character.
`noautogenfield=true | false`	When `true` , specifies that input events are missing the key field that is autogenerated by the Source window. **Note:** This parameter applies only when the format of input data is CSV.
`publishwithupsert=true | false`	When `true` , build events with opcode = Upsert instead of Insert.
`addcsvopcode= true | false`	When `true` , prepends an opcode and comma to write CSV events. The opcode is Insert unless `publishwithupsert` is `true` .
`addcsvflags=none | normal | partialupdate`	Specifies the event type to Insert into input CSV events (with comma).
`maxevents=` *`number`*	Specifies the maximum number of events to publish.
`quiesceproject=true | false`	When `true` , quiesces the project after all events are injected into the Source window.
`stripnullsfromcsv=true | false`	When `true` , strip all nulls from input CSV events.

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 190 *Required Parameters for Timer Connectors*

Parameter	Description
`type`	Specifies to publish or subscribe.
`basetime`	Specifies the start time in the format defined by the `timeformat` parameter.
`interval`	Specifies the interval length in units defined by the `unit` parameter.
`unit`	Specifies the unit of the `interval` parameter. Units include `millisecond | second | minute | hour | day | week | month | year`. **Note:** When the value of `unit` is `millisecond`, the `interval` length cannot be less than 100.

Table 191 *Optional Parameters for Timer Connectors*

Parameter	Description
`label`	Specifies the string to be written to the source window "label" field. The default value is the connector name.
`timeformat`	Specifies the format of the `basetime` parameter. The default value is "`%Y-%m-%d %H:%M:%S`".
`transactional`	Sets the event block type to `transactional`. The default value is `normal`.
`configfilesection`	Specifies the name of the section in **`/opt/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`*` ]`.
`publishwithupsert`	Builds events with `opcode = Upsert` instead of `opcode = Insert .`
`maxevents`	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, ...*

Table 192 *Required Keys*

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

Key-Value Pair	Description
`unit=millisecond | second | minute | hour | day | week | month | year`	Specifies the units of the `interval` parameter. **Note:** When `unit=millisecond`, the `interval` length cannot be less than 100.

Table 193 *Optional Keys*

Key-Value Pair	Description
`gdconfig=` *file*	Specifies the guaranteed delivery configuration file.
`transport=native | solace | tervela | rabbitmq | kafka`	Specifies the transport type. If you specify `solace`, `tervela` , `rabbitmq`, or `kafka` transports instead of the default `native` transport, then use the required client configuration files specified in the description of the C++ C_dfESPpubsubSetPubsubLib() API call.
`loglevel=trace | debug | info | warn | error | fatal | off`	Sets the logging level for the adapter. This is the same range of logging levels that you can set in the `C_dfESPpubsubInit()` publish/subscribe API call and in the engine `initialize()` call. The default level is `warn`.
`logconfigfile=` *file*	Specifies the log configuration file.
`tokenlocation=` *location*	Specifies the location of the file in the local file system that contains the OAuth token that is required for authentication by the publish/subscribe server.
`configfilesection=[` *section*`]`	Specifies the name of the section in **`/opt/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*`]`.
`transactional=true | false`	When `true` , event blocks are transactional. By default, event blocks are normal.
`publishwithupsert=true | false`	When `true` , build events with opcode = Upsert instead of Insert.
`restartonerror=true | false`	When `true` , specifies to restart the adapter if a fatal error is reported.
`transportconfigfile=` *file*	Specifies the publish/subscribe transport configuration file. For `solace` , the default is **`./solace.cfg`**. For `tervela` , the default is **`./client.config`**.

Key-Value Pair	Description
	For `rabbitmq` , the default is **`./rabbitmq.cfg`**. For `kafka` , the default is **`./kafka.cfg`**. **Note:** No transport configuration file is required for native transport.
`label=` *`string`*	Specifies the string to be written to the Source window `label` field. The default value is the connector name.
`timeformat=format`	Specifies the format of the `basetime` parameter. The default value is `%Y-%m-%d %H:%M:%S`.
`maxevents=` *`number`*	Specifies the maximum number of events to publish.
`quiesceproject=true | false`	When `true` , quiesces the project after all events are injected into the Source window.

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 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 that are available online. Download the ZIP file and examine the following models in the **`xml`** directory:

`url_connector_weather/config.xml`

`url_connector_news/config.xml`

IMPORTANT You must add the `contentType` attribute to requests whenever you use a file URL to pull data with the URL connector.

```
<requests>
<request name="results_xml" contentType="text/xml">
<url><![CDATAfile://file.xml]></url>
<headers>
<header name="content-type">application/xml</header>
</headers>
</request>
</requests>
```

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 194 *Required Parameters for Publisher URL Connectors*

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

Table 195 *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.

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. It also serves as the window key field. If the optional `cameraid` configuration parameter is configured, then the Source window schema must also contain an extra string field at the end (for example: `id*:int64,frame:blob,cameraid:string`).

Table 196 *Required Parameters for the UVC Publisher Connector*

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

Table 197 *Optional Parameters for the UVC Publisher Connector*

Parameter	Description
`frame_rate`	Specifies the frames per second that the camera streams. Must be a double. The default value is `15`.
`format_in`	Specifies the image format of captured photos. The default is `mjpg`". `yuyv`, an uncompressed image format, is also supported.
`format_out`	Specifies the image format that the connector publishes. The default is `mjpg`. `yuyv`, an uncompressed image format, is supported only when `format_in` is `yuyv`.
`width`	Specifies the width of the photo. Not all resolutions are supported. Consult the camera menu or use the `v4l2-ctl` command-line utility for a list of supported resolutions.
`height`	Specifies the height of the photo. Not all resolutions are supported. Consult the camera menu or use the `v4l2-ctl` command-line utility for a list of supported resolutions.
`brightness`	Specifies the brightness of the photo. Consult the camera menu or use the `v4l2-ctl` command-line utility for a list of supported brightness values.
`gain`	Specifies the gain of the photo.

Parameter	Description
	Consult the camera menu or use the `v4l2-ctl` command-line utility for a list of supported gain values.
`saturation`	Specifies the saturation of the photo. Consult the camera menu or use the `v4l2-ctl` command-line utility for a list of supported saturation values.
`contrast`	Specifies the contrast of the photo. Consult the camera menu or use the `v4l2-ctl` command-line utility for a list of supported contrast values.
`device`	Specifies the device name the camera is using on the Linux operating system. The device name is typically **`/dev/videoX`**.
`predelay`	Specifies a delay time, in seconds, on starting the connector. In some environments, the camera might take time to reset and initialize. If the camera fails to start and `predelay` is *n*, the connector waits *n* seconds to start. If the connector fails to detect the camera, the ESP server exits with a fatal error.
`maxevents`	Specifies the maximum number of events to publish.
`cameraid`	Specifies an arbitrary string that is copied into the corresponding string field in the Source window. This value can be used by the model to identify the source camera.

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_uvc_adapter -C *key1 =val1,key2 =val2,key3 =val3,...*

Table 198 *Required Keys*

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

Table 199 *Optional Keys*

Key-Value Pair	Description
`transport=native | solace | tervela | rabbitmq | kafka`	Specifies the transport type. If you specify `solace`, `tervela` , `rabbitmq`, or `kafka` transports instead of the default `native` transport, use the required client configuration files specified in the description of the C++ C_dfESPpubsubSetPubsubLib() API call.
`loglevel=trace | debug | info | warn | error | fatal | off`	Sets the logging level for the adapter. This is the same range of logging levels that you can set in the `C_dfESPpubsubInit()` publish/subscribe API call and in the engine `initialize()` call. The default level is `warn`.
`logconfigfile=` *file*	Specifies the log configuration file.
`tokenlocation=` *location*	Specifies the location of the file in the local filesystem that contains the OAuth token required for authentication by the publish/subscribe server.
`configfilesection=[` *section*`]`	Specifies the name of the section in **`/opt/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*`]`.
`blocksize=` *size*	Sets the block size. The default value is 1.
`transactional=true | false`	When `true`, event blocks are transactional. By default, event blocks are normal.
`publishwithupsert=true | false`	When `true`, build events with opcode = Upsert instead of Insert.
`restartonerror=true | false`	When `true` , specifies to restart the adapter if a fatal error is reported.
`frame_rate=` *number*	Specifies the number of frames per second. The default is `15`.

Key-Value Pair	Description
`device=` *name*	Specifies the device name. The default is "`/dev/video0`".
`format_in=` *format*	Specifies the image format of captured photos. The default is `mjpg`. `yuyv`, an uncompressed image format, is also supported.
`format_out=` *format*	Specifies the photo format output from the adapter. The default is `mjpg`. `yuyv` is an uncompressed image format. It is supported only when `format_in` is `yuyv`
`width=` *width*	Specifies the width for the frame resolution. The default is `640` .
`height=` *height*	Specifies the height for the frame resolution. The default is `480` .
`brightness=` *level*	Specifies the brightness level. The default value is 0. Consult the camera menu or use the **`v4l2-ctl`** command-line utility for a list of supported brightness values.
`gain=` *gain*	Specifies the gain. The default value is 0. Consult the camera menu or use the **`v4l2-ctl`** command-line utility for a list of supported gain values.
`saturation=` *saturation*	Specifies the saturation. The default value is 0. Consult the camera menu or use the **`v4l2-ctl`** command-line utility for a list of supported saturation values.
`contrast=` *contrast*	Specifies the contrast. The default value is 0. Consult the camera menu or use the **`v4l2-ctl`** command-line utility for a list of supported contrast values.
`predelay=` *seconds*	Specifies the number of seconds to delay before a retry if unable to start camera. The default is `0`.
`transportconfigfile=` *file*	Specifies the publish/subscribe transport configuration file. For `solace` , the default is **`./solace.cfg`**. For `tervela` , the default is **`./client.config`**. For `rabbitmq` , the default is **`./rabbitmq.cfg`**. For `kafka` , the default is **`./kafka.cfg`**. **Note:** No transport configuration file is required for native transport.

Key-Value Pair	Description
`maxevents= number`	Specifies the maximum number of events to publish.
`quiesceproject=true | false`	When `true` , quiesces the project after all events are injected into the Source window.
`cameraid=id`	Specifies an arbitrary string that is copied into the corresponding string field in the Source window. This value can be used by the model to identify the source camera.

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 200 *Required Parameters for WebSocket Connectors*

Parameter	Description
`type`	Specifies to publish. Must be "`pub`".
`url`	Specifies the URL for the WebSocket connection.
`configUrl`	Specifies the URL for the connector configuration file. This configuration file contains information about the transformation steps required to publish events.

Parameter	Description
	For example configuration files, see the WebSocket connector examples available online. Download the ZIP file and example the following models in the **xml** directory: **ws_connector_json/config.xml** or **ws_connector_xml/config.xml**
contentType	Specifies XML or JSON as the type of content received over the WebSocket connection.

Table 201 Optional Parameters for WebSocket Connectors

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

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.

If required, configure the value of the `mqpassword` parameter with an encrypted password. You can generate the encrypted version of the password by running the `dfesp_pwd_encrypt` utility that is found in the **/opt/sas/viya/home/SASEventStreamProcessingEngine/bin** directory of the ESP server container. You must run the utility within the container. The command syntax is as follows:

```
dfesp_pwd_encrypt mqpassword espMQconnectorUsedByUser=mquserid
```

Next, use the encrypted password as the value of the `mqpassword` parameter and enable the `mqpasswordencrypted` parameter. Alternatively, use SAS Event Stream Processing Studio to encrypt this value.

Use the following parameters for MQ connectors.

Table 202 Required Parameters for Subscriber MQ Connectors

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

Table 203 Required Parameters for Publisher MQ Connectors

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

Table 204 Optional Parameters for Subscriber MQ Connectors

Parameter	Description
`mqtopic`	Specifies the MQ topic name. Required if `mqqueue` is not configured.
`mqqueue`	Specifies the MQ queue name. Required if `mqtopic` is not configured.
`collapse`	Enables conversion of UPDATE_BLOCK events to make subscriber output publishable.
`queuemanager`	Specifies the MQ queue manager.
`dateformat`	Specifies the format of `DATE` and `STAMP` fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (`DATE`) or microseconds (`STAMP`) since epoch. The `dateformat` parameter accepts any time format that is supported by the UNIX `strftime` function.

Parameter	Description
rmretdel	Specifies to remove all delete events from event blocks received by a subscriber that were introduced by a window retention policy.
configfilesection	Specifies the name of the section in **/opt/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*].
protofile	Specifies the **.proto** file that contains the Google Protocol Buffers message definition. This definition is used to convert event blocks to protobuf messages. When you specify this parameter, you must also specify the protomsg parameter. This parameter is ignored when mqtype is not set to **protobuf** .
protomsg	Specifies the name of a Google Protocol Buffers message in the **.proto** file that you specified with the protofile parameter. Event blocks are converted into this message. This parameter is ignored when mqtype is not set to **protobuf** .
usecorrelid	Copies the value of the **correlid** field in the event to the MQ message correlation ID.
csvmsgperevent	For CSV, specifies to send one message per event. The default is one message per transactional event block or else one message per event.
csvmsgpereventblock	For CSV, specifies to send one message per event block. The default is one message per transactional event block or else one message per event.
doubleprecision	Specifies the number of fractional digits in the ASCII representation of a double. The default value is 6.
mquserid=*id*	Specifies the user ID to use for authenticating against the queue manager.
mqpassword=*password*	Specifies the password to use for authenticating against the queue manager.
mqpasswordencrypted=true | false	When **true**, mqpassword is encrypted.

Table 205 *Optional Parameters for Publisher MQ Connectors*

Parameter	Description
`mqtopic`	Specifies the MQ topic name. Required if `mqqueue` is not configured.
`mqqueue`	Specifies the MQ queue name. Required if `mqtopic` is not configured.
`mqsubname`	Specifies the MQ subscription name. Required if `mqqueue` is not configured.
`blocksize`	Specifies the number of events to include in a published event block. The default value is 1.
`transactional`	Sets the event block type to transactional. The default event block type is normal.
`dateformat`	Specifies the format of `DATE` and `STAMP` fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (`DATE`) or microseconds (`STAMP`) since epoch. The `dateformat` parameter accepts any time format that is supported by the UNIX `strftime` function.
`queuemanager`	Specifies the MQ queue manager.
`configfilesection`	Specifies the name of the section in **`/opt/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`*`]`.
`ignorecsvparseerrors`	Specifies that when a field in an input CSV event cannot be parsed, the event is dropped, an error is logged, and publishing continues.
`protofile`	Specifies the **`.proto`** file that contains the Google Protocol Buffers message definition. This definition is used to convert event blocks to protobuf messages. When you specify this parameter, you must also specify the `protomsg` parameter. This parameter is ignored when `mqtype` is not set to **`protobuf`**.
`protomsg`	Specifies the name of a Google Protocol Buffers message in the **`.proto`** file that you specified with the `protofile` parameter. Event blocks are converted into this message. This parameter is ignored when `mqtype` is not set to **`protobuf`**.

Parameter	Description
`csvfielddelimiter`	Specifies the character delimiter for field data in input CSV events. The default delimiter is the `,` character.
`noautogenfield`	Specifies that input events are missing the key field that is autogenerated by the source window. **Note:** This parameter applies only when the format of input data is CSV.
`publishwithupsert`	Builds events with `opcode=Upsert` instead of `Insert`.
`addcsvopcode`	Prepends an opcode and comma to input CSV events. The opcode is Insert unless `publishwithupsert` is enabled.
`addcsvflags`	Specifies the event type to insert into input CSV events (with a comma). Valid values are **`normal`** and **`partialupdate`**.
`usecorrelid`	Copies the value of the MQ message correlation ID into the **`correlid`** field in every SAS Event Stream Processing event.
`ignoremqmdformat`	Specifies to ignore the value of the Message Descriptor Format parameter, and assume the message format is compatible with the `mqtype` parameter setting.
`maxevents`	Specifies the maximum number of events to publish.
`stripnullsfromcsv`	Strip all nulls from input CSV events.
`mquserid=`*`id`*	Specifies the user ID to use for authenticating against the queue manager.
`mqpassword=`*`password`*	Specifies the password to use for authenticating against the queue manager.
`mqpasswordencrypted=true | false`	When **`true`**, `mqpassword` is encrypted.

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,...*

If required, configure the value of the `mqpassword` parameter with an encrypted password. You can generate the encrypted version of the password by running the `dfesp_pwd_encrypt` utility that is found in the **/opt/sas/viya/home/SASEventStreamProcessingEngine/bin** directory of the ESP server container. You must run the utility within the container. The command syntax is as follows:

```
dfesp_pwd_encrypt mqpassword espMQconnectorUsedByUser=mquserid
```

Next, use the encrypted password as the value of the `mqpassword` parameter and enable the `mqpasswordencrypted` parameter. Alternatively, use SAS Event Stream Processing Studio to encrypt this value.

Subscriber Usage

Table 206 *Required Keys*

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

Table 207 *Optional Keys*

Key-Value Pair	Description
`mqtopic=` *name*	Specifies the MQ topic name.
`queuemanager=` *name*	Specifies the MQ queue manager name.
`dateformat=` *format*	Specifies the format of `DATE` and `STAMP` fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (`DATE`) or microseconds (`STAMP`) since epoch. The `dateformat` parameter accepts any time format that is supported by the UNIX `strftime` function.

Key-Value Pair	Description
protofile= *file*	Specifies the **.proto** file to be used for Google protocol buffer support. This key is ignored when mqtype is not set to **protobuf** .
protomsg= *message*	Specifies the message itself in the **.proto** file that is specified by the protofile parameter. This key is ignored when mqtype is not set to **protobuf** .
gdconfig= *file*	Specifies the guaranteed delivery configuration file.
transport=native | solace | tervela | rabbitmq | kafka	Specifies the transport type. If you specify solace, tervela , , orrabbitmq kafka transports instead of the default native transport, use the required client configuration files specified in the description of the C++ C_dfESPpubsubSetPubsubLib() API call.
loglevel=trace | debug | info | warn | error | fatal | off	Sets the logging level for the adapter. This is the same range of logging levels that you can set in the C_dfESPpubsubInit() publish/subscribe API call and in the engine initialize() call. The default level is warn.
logconfigfile= *file*	Specifies the log configuration file.
tokenlocation= *location*	Specifies the location of the file in the local filesystem that contains the OAuth token required for authentication by the publish/subscribe server.
configfilesection=[*section*]	Specifies the name of the section in **/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.config** (Linux) or **%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.config** (Windows) to parse for configuration parameters. Specify the value as [*configfilesection*].
usecorrelid=true | false	When true, copies the MQ correlation ID to or from the **correlid** string field in the SAS Event Stream Processing event. For non-binary formats only.
restartonerror=true | false	When true , specifies to restart the adapter if a fatal error is reported.
mqqueue= *name*	Specifies the MQ queue name.
transportconfigfile= *file*	Specifies the publish/subscribe transport configuration file. For solace , the default is **./solace.cfg**. For tervela , the default is **./client.config**.

Key-Value Pair	Description
	For `rabbitmq` , the default is **`./rabbitmq.cfg`**. For `kafka` , the default is **`./kafka.cfg`**. **Note:** No transport configuration file is required for native transport.
`csvmsgperevent= true | false`	When true, 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.
`mquserid=`*`id`*	Specifies the user ID to use for authenticating against the queue manager.
`mqpassword=`*`password`*	Specifies the password to use for authenticating against the queue manager.
`mqpasswordencrypted=true | false`	When **`true`**, `mqpassword` is encrypted.

Publisher Usage

Table 208 *Required Keys*

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

Table 209 *Optional Keys*

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

Key-Value Pair	Description
`usecorrelid=true | false`	When true, copies the MQ correlation ID to or from the **`correlid`** string field in the SAS Event Stream Processing event. For non-binary formats only.
`restartonerror=true | false`	When `true` , specifies to restart the adapter if a fatal error is reported.
`mqqueue=` *name*	Specifies the MQ queue name.
`transportconfigfile=` *file*	Specifies the publish/subscribe transport configuration file. For `solace` , the default is **`./solace.cfg`**. For `tervela` , the default is **`./client.config`**. For `rabbitmq` , the default is **`./rabbitmq.cfg`**. For `kafka` , the default is **`./kafka.cfg`**. **Note:** No transport configuration file is required for native transport.
`mqsubname=` *name*	Specifies the MQ subscription name. Required if `mqqueue` is not configured.
`mqsubqueue=` *name*	Specifies the MQ queue name. **Note:** For backward compatibility, use `-Q` only.
`blocksize=` *size*	Sets the block size. The default value is 1.
`transactional=true | false`	When `true` , event blocks are transactional. By default, event blocks are normal.
`ignorecsvparseerrors=true | false`	Specifies that when a field in an input CSV event cannot be parsed, the event is dropped, an error is logged, and publishing continues.
`csvfielddelimiter=` *delimiter*	Specifies the character delimiter for field data in input CSV events. The default delimiter is the `,` character.
`noautogenfield=true | false`	When `true` , specifies that input events are missing the key field that is autogenerated by the Source window. **Note:** This parameter applies only when the format of input data is CSV.
`publishwithupsert=true | false`	When `true` , build events with opcode = Upsert instead of Insert.

Key-Value Pair	Description
`addcsvopcode=true | false`	When `true` , prepends an opcode and comma to write CSV events. The opcode is Insert unless `publishwithupsert` is `true` .
`addcsvflags=none | normal | partialupdate`	Specifies the event type to Insert into input CSV events (with comma).
`ignoremqmdformat=true | false`	Specifies to ignore the value of the Message Descriptor Format parameter, and assume that the message format is compatible with the `mqtype` parameter setting.
`maxevents=` *number*	Specifies the maximum number of events to publish.
`quiesceproject=true | false`	When `true` , quiesces the project after all events are injected into the Source window.
`stripnullsfromcsv=true | false`	When `true` , strip all nulls from input CSV events.
`mquserid=`*id*	Specifies the user ID to use for authenticating against the queue manager.
`mqpassword=`*password*	Specifies the password to use for authenticating against the queue manager.
`mqpasswordencrypted=true | false`	When **`true`**, `mqpassword` is encrypted.

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`
- `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.

For more information about the `dfESPconnector` class, open **`$DFESP_HOME/doc/html/index.html`** (UNIX deployments) or **`%DFESP_HOME%\doc\html\index.html`** (Windows deployments) in a web browser. This provides access to the complete class and method documentation.

A connector can implement a subscriber that receives events generated by a window, or a publisher that injects events into a window. However, a single instance of a connector cannot publish and subscribe simultaneously.

A subscriber connector receives events by using a callback method defined in the connector class that is invoked in a thread owned by the engine. A publisher connector typically creates a dedicated thread to read events from the source. It then injects those events into a Source window, leaving the main connector thread for subsequent calls made into the connector.

A connector must define these static data structures:

Static Data Structure	Description
`dfESPconnectorInfo`	Specifies the connector name, publish/subscribe type, initialization function pointer, and configuration data pointers.
`subRequiredConfig`	Specifies an array of `dfESPconnectorParmInfo_t` entries listing required configuration parameters for a subscriber.
`sizeofSubRequiredConfig`	Specifies the number of entries in `subRequiredConfig`.
`pubRequiredConfig`	Specifies an array of `dfESPconnectorParmInfo_t` entries listing required configuration parameters for a publisher.

Static Data Structure	Description
`sizeofPubRequiredConfig`	Specifies the number of entries in `pubRequiredConfig`.
`subOptionalConfig`	Specifies an array of `dfESPconnectorParmInfo_t` entries listing optional configuration parameters for a subscriber.
`sizeofSubOptionalConfig`	Specifies the number of entries in `subOptionalConfig`.
`pubOptionalConfig`	Specifies an array of `dfESPconnectorParmInfo_t` entries listing optional configuration parameters for a publisher.
`sizeofPubOptionalConfig`	Specifies the number of entries in `pubOptionalConfig`.

A connector must define these static methods:

Static Method	Description
`dfESPconnector *initialize(dfESPengine *`*`engine`*` , dfESPpsLib_t `*`psLib`*`)`	Returns an instance of the connector.
`dfESPconnectorInfo *getConnectorInfo()`	Returns the `dfESPconnectorInfo` structure.

You can invoke these static methods before you create an instance of the connector.

A connector must define these virtual methods:

Virtual Method	Description
`start()`	Starts the connector. Must call base class method `checkConfig()` to validate connector configuration before starting. Must also call base class method `start()`. Must set `variable _started = true` upon success.
`stop()`	Stops the connector. Must call base class method `stop()`. Must leave the connector in a state whereby `start()` can be subsequently called to restart the connector.

Virtual Method	Description
`callbackFunction()`	Specifies the method invoked by the engine to pass event blocks generated by the window to which it is connected.
`errorCallbackFunction()`	Specifies the method invoked by the engine to report errors detected by the engine. Must call user callback function `_errorCallback`, if nonzero.
`setupCallbackFunction()`	Specifies the method invoked by the engine to set up any connector that requires the Source window schema.

A connector must set its running state for use by the connector orchestrator. It does this by calling the `dfESPconnector::setState()` method. The two relevant states are **`state_RUNNING`** and **`state_FINISHED`**. All connectors must set **`state_RUNNING`** when they start. Only connectors that actually finish transferring data need to set **`state_FINISHED`**. Typically, setting state in this way is relevant only for publisher connectors that publish a finite number of event blocks.

Finally, a derived connector can implement up to ten user-defined methods that can be called from an application. Because connectors are plug-ins loaded at run time, a user application cannot directly invoke class methods. It is not linked against the connector.

The base connector class defines virtual methods `userFunction_01` through `userFunction_10`, and a derived connector then implements those methods as needed. For example:

```
void * myConnector::userFunction_01(void *myData) {
```

An application would invoke the method as follows:

```
myRC = myConnector->userFunction_01((void *)myData);
```

Integrating a Custom Connector

All connectors are managed by a global connector manager. The default connectors shipped with 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.