SAS® Event Stream Processing 5.2: Connectors and Adapters

Using Connectors and Adapters

Overview

To stream data into or out of an event stream processing engine window, you can use a connector or an adapter. Connectors and adapters use the publish/subscribe API to interface with a variety of communication fabrics, drivers, and clients. Connectors are C++ classes that are instantiated in the same process space as the event stream processing engine. You can use connectors from within XML models or C++ models. Adapters are stand-alone executable files, usually built from connector classes. Thus, some adapters are executable versions of their corresponding connector.

Configuration Directory

After you deploy SAS Event Stream Processing, configuration files are located in a specific location:

Table 1 *Default Location of Configuration Files*

Operating System	Location
Linux	`/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default`
Windows	`%ProgramData%\SAS\Viya\etc\SASEventStreamProcessingEngine\default`

On Windows, if the environment variable ProgramData is defined, it is placed at the beginning of the configuration directory's path, before `\SAS\`.

You can use two environment variables to set up alternative configurations.

Table 2 Environment Variables to Set Up Alternative Configurations

Environment Variable	Description
DFESP_CONFIG	Replaces the default location. You must copy files from the default location to the full path specified in order to use alternative logs, metering, and MAS stores. The ESP servers run by a user with this variable set would abide by configuration values specified in the alternative location.
DFESP_CONFIG_TAIL	Appends to the default location. You must copy files from the default location to the specified subdirectory in order to use alternative logs, metering, and MAS stores.

Note: To edit configuration files, you must be a member of the `sas` group.

Overview to Connectors

What Do Connectors Do?

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.

Connector Examples

C++ Examples

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

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

- `db_connector_publisher`
- `db_connector_subscriber`
- `json_connector_publisher`

- `json_connector_subscriber`
- `socket_connector_publisher`
- `socket_connector_subscriber`
- `xml_connector_publisher`
- `xml_connector_subscriber`

XML Examples

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

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

Obtaining Connectors

To obtain a new instance of a connector in a C++ application, call the `dfESPwindow::getConnector()` method. Pass the connector type as the first parameter, and pass a connector instance name and an "active" Boolean value as optional second and third parameters:

```
dfESPconnector *inputConn =
static_cast<dfESPconnector *>(input->getConnector("fs","inputConn", true));
```

The packaged connector types are as follows:

- `adapter`
- `bacnet` (Linux only)
- `db`
- `fs`
- `kafka`
- `modbus`
- `mq`
- `mqtt`
- `nurego`
- `opcua`
- `pylon`
- `project`
- `rmq`
- `sniffer`
- `smtp`

- `sol`
- `tdata`
- `tdlistener`
- `tibrv`
- `timer`
- `tva`
- `url`
- `uvc`
- `websocket`
- `pi` (Windows only)

After a connector instance is obtained, any of its base class public methods can be called. This includes `setParameter()`, which can be called multiple times to set required and optional parameters. Parameters must be set before the connector is started.

The `type` parameter is required and is common to all connectors. It must be set to `pub` or `sub`.

Additional connector configuration parameters are required depending on the connector type, and are described later in this section.

Activating Optional Plug-ins and 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, SAS Event Stream Processing loads all connectors found in the **`plugins`** directory, except those specified in the file **`connectors.excluded`** in the configuration directory.

By default, this file specifies connectors that 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.

Edit **`connectors.excluded`** as needed. You can list any of the valid connector types in this file.

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 **`connectors.config`** 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.

Orchestrating Connectors

By default, all connectors start automatically when their associated project starts and they run concurrently. Once connector orchestration is enabled, only connectors with connector orchestration defined are started. Connector orchestration enables you to define the order in which connectors within a project execute, depending on the state of the connector. You can thereby 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.

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

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

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

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

Consider the following configuration that consists of four 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>}
```

And then 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 C++ code to define this orchestration 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);
```

The corresponding XML code 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>
```

Overview to Adapters

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

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

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

Adapters must have full access to all 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.

Adapters are written in C++, Java, or Python.

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

Language	Adapter
Java Note: The Java adapters are built using Java version 8. You must use Java SE Runtime Environment 8 on any platform running a Java adapter.	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 Twitter Publisher
Python	BoardReader Twitter GNIP

Similar to connectors, adapters can obtain their configuration parameters from a file. C++ adapters use the configuration file in the configuration directory. Java adapters use **javaadapters.config**.

Note: All 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.

You can specify a section label in the configuration file using the **-C** argument on the command line when you run the adapter. For more information, see "Setting Configuration Parameters in a File".

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

All 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.

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

Using the Adapter Connector

Recall that connectors are C++ classes that are instantiated in the same process space as the ESP server. You can use the ESP server to orchestrate the order in which connectors run in a project. (For more information about connector orchestration, see "Orchestrating Connectors".)

Unlike connectors, adapters are stand-alone processes that you manage directly with command–line tools. Adapters cannot be orchestrated with the ESP server or ESP client. The inability to use the ESP server to manage the order in which adapters run makes them inconvenient to deploy in scale.

Many adapters have a corresponding connector, but some do not. 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. You can also use the Adapter Connector to start and end adapter processes. This enables you to manage the lifecycle of an adapter with the ESP server, regardless of any need for orchestration.

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

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

Table 3 *Required Parameters for the Adapter Connector*

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

Table 4 *Optional Parameters for the Adapter Connector*

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

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

Using the Bacnet Connector and Adapter

Using the Bacnet Connector

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

Note: Support for the Bacnet connector is available only on Linux platforms.

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

The format for the JSON configuration file is as follows:

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

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

Multiple Bacnet connector instances cannot exist within a single process. To run multiple Bacnet connector instances as individual processes, you must run multiple Bacnet adapters. You can run these adapters manually as separate processes or as multiple Adapter Connectors configured in the event stream processing model.

The following fields are required 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 5 Required Parameters for the Publisher Bacnet Connector

Parameter	Description
`bacnetbbmdaddress`	Specifies the IP address of the BBMD
`bacnetbbmdport`	Specifies the port of the BBMD

Parameter	Description
`bacnetconfigfile`	Specifies the JSON configuration file containing Bacnet device and object

Table 6 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]`.
`ignoretimeouts`	Logs a warning and continues if an attempt to read a property from a Bacnet device results in a timeout. The default is to log an error and stop.
`publishwithupsert`	Builds events with `opcode=Upsert` instead of `Insert`.
`maxevents`	Specifies the maximum number of events to publish.
`transactional`	Sets the event block type to transactional. The default event block type is normal.

The Bacnet to SAS Event Stream Processing data type mappings are as follows:

Bacnet type	SAS Event Stream Processing data types
Boolean	ESP_INT32 ESP_INT64 ESP_DOUBLE
Unsigned Int	ESP_INT32 ESP_INT64 ESP_DOUBLE
Signed Int	ESP_INT32 ESP_INT64 ESP_DOUBLE
Real	ESP_DOUBLE
Double	ESP_DOUBLE
Character String	ESP_UTF8STR

Bacnet type	SAS Event Stream Processing data types
Date	ESP_DATETIME ESP_TIMESTAMP
Enumerated	ESP_INT32 ESP_INT64 ESP_DOUBLE
Octet String	ESP_BINARY

Using the Bacnet 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.

Publisher usage:

dfesp_bacnet_adapter -B *bacnetbbmdaddress* <-b *blocksize*> <-C *configfilesection*> -c *bacnetconfigfile* <-E *tokenlocation*> <-e> <-g *gdconfig*> -h *url* <-I *bacnetipport*> <-i> <-j trace | debug | info | warn | error | fatal | off> <-l native | solace | tervela | rabbitmq | kafka> <-m *maxevents*> -P *bacnetbbmdport* <-Q> <-R> <-V> <-y *logconfigfile*> <-Z *transportconfigfile*>

Parameter	Description
`-B bacnetbbmdaddress`	Specifies the IP address of the BBMD.
`-b blocksize`	Sets the block size. The default value is `1`.
`-C [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 connection parameters.
`-c bacnetconfigfile`	Specifies the JSON configuration file containing Bacnet device and object information.
`-E tokenlocation`	Specifies the location of the file in the local file system that contains the OAuth token required for authentication by the publish/subscribe server.
`-e`	Specifies that events are transactional.
`-g gdconfig`	Specifies the guaranteed delivery configuration file.
`-h url`	Specifies the dfESP publish and subscribe standard URL in the form **`dfESP://host:port/project/continuousquery/window`**.

Parameter	Description
`-j 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.
`-I bacnetipport`	Specifies the local port used by the adapter. The default port number is `47808`.
`-i`	Logs a warning and continues if an attempt to read a property from a Bacnet device results in a timeout. The default is to log an error and stop.
`-l 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.
`-m maxevents`	Specifies the maximum number of events to publish.
`-P bacnetbbmdport`	Specifies the port of the BBMD.
`-Q`	When *maxevents* is configured, quiesces the project after all events are injected into the Source window.
`-R`	Builds events with `opcode=Upsert` instead of Insert.
`-V`	Restarts the adapter if a fatal error is reported.
`-y logconfigfile`	Specifies the log configuration file.
`-Z transportconfigfile`	Specifies the publish/subscribe transport configuration file. The default value depends on the transport type specified with the `-l` parameter: For `-l solace`, the default is **`./solace.cfg`**. For `-l tervela`, the default is **`./client.config`**. For `-l rabbitmq`, the default is **`./rabbitmq.cfg`**. For `-l kafka`, the default 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: Advanced Topics*.

Using the BoardReader Publisher Adapters

The BoardReader application aggregates message feeds. 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.

Here are the parameters that you must configure:

- `customer_project_id`
- `customer_query_id`
- `key`
- `query`

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

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

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

Every received JSON response is converted to an event block using the JSON parsing rules described in "Publish/Subscribe API Support for JSON Messaging" in *SAS Event Stream Processing: Publish/Subscribe API* . The corresponding Source window must contain fields that correspond to every received JSON key unless you specify the adapter parameter `esp-ignoremissingschemafields`. Events are built with the Insert opcode unless you configure the adapter to use Upsert instead. If needed, a JSON filtering function can be enabled using the `esp-matchsubstrings` parameter.

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

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

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

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

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

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

Usage:

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

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

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

Using the SAS Cloud Analytic Services Adapter

The subscriber adapter supports appending event stream processing events to a global table in the SAS Cloud Analytic Services server. The table includes a column named "_opcode" that contains the opcode associated with each SAS Event Stream Processing event.

The publisher adapter supports fetching 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.

You must define the **`DFESP_JAVA_TRUSTSTORE`** and **`SSLCALISTLOC`** environment variables for the client target platform. The SAS Cloud Analytic Services server requires that client connections be secured with TLS encryption, so **`DFESP_JAVA_TRUSTSTORE`** sets the location of your trust store .jks file and **`SSLCALISTLOC`** sets the location of the .pem file containing required certificates.

`caspassword` must be specified in non-encrypted form if the -`p` parameter is used and encryption is not enabled with the -`E` parameter. The encrypted version of the password can be generated using OpenSSL. The OpenSSL executable is included in the SAS Event Stream Processing System Encryption and Authentication Overlay installation. Use the following command on the console to use OpenSSL to display your encrypted password:

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

Use the encrypted password to specify `caspassword` for the -`p` parameter and enable encryption with the -`E` parameter.

Subscriber use:

dfesp_cas_adapter -H *cashostport* -h *url* -k *sub* -t *castable* <-a *commitblocks*> <-b *buffersize*> <-C *configfilesection*> <-E> <-g *gdconfigfile*> <-i *cassessionid*> <-L *pubsublib*> <-l *severe* | *warning* | *info*> <-m *caslib*> <-Z *transportconfigfile*> <-n *casusername*> <-O *tokenlocation*> <-p *caspassword*> <-s *columns*> <-V>

Note: For a subscriber, the specified table in the 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* parameter, the table is promoted to a global table.

Publisher use:

dfesp_cas_adapter -H *cashostport* -h *url* -k *pub* -t *castable* <-b *blocksize*> <-C *configfilesection*> <-c> <-E> <-e> <-g *gdconfigfile*> <-i *cassessionid*> <-L *pubsublib*> <-l *severe* | *warning* | *info*> <-m *caslib*> <-Z *transportconfigfile*> <-n *casusername*> <-O *tokenlocation*> <-p *caspassword*> <-s *columns*> <-u> <-V> <-W *maxevents*>

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.

Parameter	Description
`-a commitblocks`	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.
`-b blocksize`	Specifies the number of events per event block.
`-b buffersize`	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.
`-C [configfilesection]`	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.
`-c`	Specifies to quiesce the project after all events are injected into the Source window. **`javaadapters.config`** parameter name: `quiesceproject`
`-E`	Specifies that *caspassword* is encrypted. **`javaadapters.config`** parameter name: `pwdencrypted`
`-e`	Specifies that event blocks are transactional. The default event block type is normal. **`javaadapters.config`** parameter name: `transactional`
`-g gdconfigfile`	Specifies the guaranteed delivery configuration file for the client.
`-H cashostport`	Specifies the SAS Cloud Analytic Services host name and port.

`-h url`	Specifies the dfESP standard publish and subscribe 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 the following for subscribers if needed:
	`?collapse=true | false`
	`?rmretdel=true | false`

`-i cassessionid`	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.
`-k sub | pub`	Specifies subscribe or publish.

Parameter	Description
`-L native | rabbitmq | solace | tervela | kafka`	Specifies the transport type. When you specify `rabbitmq`, `solace`, `tervela`, 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.
`-l severe | warning | info`	Specifies the application logging level.
`-m caslib`	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.
`-n casusername`	Specifies the user name used to authenticate the SAS Cloud Analytic Services session.
`-O tokenlocation`	Specifies the location of the file in the local file system that contains the OAuth token required for authentication by the publish/subscribe server.
`-p caspassword`	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.
`-s columns`	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 fields names in the event stream processing window, you do not need to use the `-s` parameter. 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).
`-t castable`	Specifies the name of the SAS Cloud Analytic Services table.
`-u`	Builds events with `opcode = Upsert` instead of `Insert`. **`javaadapters.config`** parameter name: `publishwithupsert`
`-V`	Restarts the adapter if a fatal error is reported. **`javaadapters.config`** parameter name: `restartonerror`
`-W maxevents`	Specifies the maximum number of events to publish.

Parameter	Description
`-Z transportconfigfile`	Specifies the publish/subscribe transport configuration file. The default value depends on the transport type specified with the `-L` parameter: Specifies the publish/subscribe transport configuration file. The default value depends on the transport type specified with the `-L` parameter: For `-L solace`, the default is **`./solace.cfg`**. For `-L tervela`, the default is **`./client.config`**. For `-L rabbitmq`, the default is **`./rabbitmq.cfg`**. For `-L kafka`, the default 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 `-s` 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.

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
VARBINARY	BINARY

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

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

Using the Cassandra Adapter

The Cassandra 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 version 3.0.0 or higher of the Datastax Java Driver for Apache Cassandra, which you must download from https://github.com/datastax/java-driver. This driver supports Cassandra version 1.2 through 3.0 and later. It negotiates the version of native protocol to use during a connection. The client target platform must then define the environment variable DFESP_DATASTAX_JARS to specify the location of the Datastax Java driver JAR files.

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

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.

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.

Subscriber usage:

dfesp_cassandra_subscriber -f *node* -k *keyspace* -t *table* -u *url* <-a *loadbalancingpolicy* > <-b *batchsize* > <-c *configfilesection* > <-g *gdconfigfile* > <-l native | rabbitmq | solace | tervela | kafka> <-m *compression* > <-n *username* > <-O *tokenlocation* > <-o *loglevel* > <-p *password* > <-S> <-s *batchsecs* > <-y *retrypolicy* > <-x *port* > <-V> <-Z *transportconfigfile*>

Publisher usage:

dfesp_cassandra_publisher -e *selectstatement* -f *node* -k *keyspace* -t *table* -u *url* <-a *loadbalancingpolicy* > <-b *blocksize* > <-c *configfilesection* > <-g *gdconfigfile* > <-l native | rabbitmq | solace | tervela | kafka> <-m *compression* > <-n *username* > <-O *tokenlocation* > <-o *loglevel* > <-p *password* > <-r> <-S> <-s> <-y *retrypolicy* > <-x *port* > <-Q> <-V> <-W *maxevents*> <-Z *transportconfigfile*>

Parameter	Description
`-a loadbalancingpolicy`	Specify a Cassandra load balancing policy, with optional comma-separated wrapped policies, default is `"TokenAware,DCAwareRoundRobin"`.
`-b batchsize`	Specify the total number of statements per Insert, Update, and Delete set of batches. The default is the event block size.
`-b blocksize`	Specify the number of events per event block.

Parameter	Description
`-c [configfilesection]`	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.
`-e selectstatement`	Specify the CQL statement to use to pull rows from the Cassandra table.
`-f node`	Specify the Cassandra cluster node IP address or host name.
`-g gdconfigfile`	Specifies the guaranteed delivery configuration file for the client.
`-k keyspace`	Specify the Cassandra keyspace.
`-l native | rabbitmq | solace | tervela | kafka`	Specifies the transport type. If you specify `rabbitmq`, `solace`, `tervela`, 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.
`-m compression`	Specifies Cassandra transport compression, valid values are `"NONE"`, `"LZ4"`, `"SNAPPY"`, default is `"NONE"`.
`-n username`	Specifies the user name to log on to the Cassandra host.
`-O tokenlocation`	Specifies the location of the file in the local file system that contains the OAuth token required for authentication by the publish/subscribe server.
`-o severe | warning | info`	Specifies the application logging level.
`-p password`	Specifies the password to use with the specified user name.
`-Q`	Quiesces the project after all events are injected into the Source window. **`javaadapters.config`** parameter name: `quiesceproject`
`-r`	Specifies that event blocks are transactional. The default event block type is normal. **`javaadapters.config`** parameter name: `transactional`
`-S`	Enables SSL on the Cassandra connection, using the default platform JSSE options. **`javaadapters.config`** parameter name: `ssl`
`-s batchsecs`	Specifies the maximum number of seconds to hold an incomplete batch. The default is 0.

Parameter	Description
`-s`	Builds events with `opcode=Upsert` instead of `Insert`. **`javaadapters.config`** parameter name: `publishwithupsert`
`-t table`	Specifies the Cassandra table.

`-u url`	Specifies the dfESP standard publish and subscribe 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 the following for subscribers if needed:
	`?collapse=true | false`
	`?rmretdel=true | false`

`-V`	Restarts the adapter if a fatal error is reported. **`javaadapters.config`** parameter name: `restartonerror`
`-W maxevents`	Specifies the maximum number of events to publish.
`-x port`	Specifies the Cassandra cluster node port. The default value is 9042.
`-y retrypolicy`	Specifies a Cassandra retry policy, with optional comma-separated wrapped policies, default is “`Default`”.
`-Z transportconfigfile`	Specifies the publish/subscribe transport configuration file. The default value depends on the transport type specified with the `-l` parameter: For `-l solace`, the default is **`./solace.cfg`**. For `-l tervela`, the default is **`./client.config`**. For `-l rabbitmq`, the default is **`./rabbitmq.cfg`**. For `-l kafka`, the default 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: Advanced Topics*.

Using the Database Connector and Adapter

Using the Database Connector

Overview to Using the Database Connector

The database connector supports both publish and subscribe operations. 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. For more information about driver-specific parameters, see the Driver Reference for SAS Federation Server in the *SAS Federation Server: Administrator's Guide*.

Some drivers also require additional installation of a client library. These requirements are also listed in the Driver Reference for SAS Federation Server.

Note that if 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 to ensure that the driver correctly uses UTF-16/UCS-2 when calling into the unixODBC Driver Manager.

For Windows installations, 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 in 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.

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

If required, you can configure the `pwd` field in the `connectstring` parameter with an encrypted password. The encrypted version of the password can be generated by using OpenSSL, which must be installed on your system. When you installed the SAS Event Stream Processing System Encryption and Authentication Overlay, you install the included OpenSSL executable. Use the following command on the console to invoke OpenSSL to display your encrypted password:

```
echo "pwd in connectstring" | openssl enc -e -aes-256-cbc -a -salt
-pass pass:"SASespDBconnectorUsedByUser=uid in connectstring"
```

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

Use the following parameters with database connectors.

Note: For the Teradata platform, a separately packaged connector and adapter use the Teradata Parallel Transporter for improved performance.

Table 7 Required Parameters for Subscriber Database Connectors

Parameter	Description
`connectstring`	Specifies the driver type and other driver-specific parameters.
`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 8 Required Parameters for Publisher Database Connectors

Parameter	Description
`connectstring`	Specifies the driver type and other driver-specific parameters.
`type`	Specifies to publish. Must be “`pub`”.

Table 9 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.

Table 10 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]`.
`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”.

Parameter	Description
`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.

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

Subscriber Event Stream Processor to SQL Data Type Mappings

Subscriber Event Stream Processor Data Type	SQL Data Type
ESP_UTF8STR	SQL_CHAR, SQL_VARCHAR, SQL_LONGVARCHAR, SQL_WCHAR, SQL_WVARCHAR, SQL_WLONGVARCHAR, SQL_BINARY, SQL_VARBINARY, SQL_LONGVARBINARY, SQL_GUID
ESP_INT32	SQL_INTEGER, SQL_BIGINT, SQL_DECIMAL, SQL_BIT, SQL_TINYINT, SQL_SMALLINT
ESP_INT64	SQL_BIGINT, SQL_DECIMAL, SQL_BIT, SQL_TINYINT, SQL_SMALLINT
ESP_DOUBLE	SQL_DOUBLE, SQL_FLOAT, SQL_REAL, SQL_NUMERIC, SQL_DECIMAL
ESP_MONEY	SQL_DOUBLE (converted to ESP_DOUBLE), SQL_FLOAT, SQL_REAL, SQL_NUMERIC (converted to SQL_NUMERIC), SQL_DECIMAL (converted to SQL_NUMERIC)
ESP_DATETIME	SQL_TYPE_DATE (sets only year/month/day), SQL_TYPE_TIME (sets only hours/minutes/seconds), SQL_TYPE_TIMESTAMP (sets fractional seconds = 0)
ESP_TIMESTAMP	SQL_TYPE_TIMESTAMP

Publisher SQL to Event Stream Processor Data Type Mappings

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

Using Log Miner Modes

Overview

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

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

For information about Oracle requirements, see the Oracle Administration guide: http://docs.oracle.com/cd/B28359_01/server.111/b28319/logminer.htm. To summarize: the database must be in archive log mode with supplemental logging enabled, `ALTER DATABASE ADD SUPPLEMENTAL LOG DATA`. The user reading the logs must also have `SELECT_CATALOG_ROLE` and `SELECT ANY TRANSACTION` privileges.

Using Oracle Log Miner Mode

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

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

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

The connector runs the repeated query with the following statement:

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

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

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

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

The connector completes these fields as follows:

- `index`: a unique incrementing value
- `ts`: the timestamp when the SQL_REDO was received
- `sql_operation`: INSERT/UPDATE/DELETE
- `sql_timestamp`: the timestamp in the SQL_REDO response
- `schema`: the `logminerschemaowner` configured on the connector
- `tablename`: the `logminertablename` configured on the connector

- sql_`where`: the contents of the WHERE clause in Update and Delete statements, excluding the **`ROWID`** value. Null for Insert.

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

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

Using Greenplum Log Miner Mode

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

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

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

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

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

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

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

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

Using the Database Adapter

See "Using the Database Connector" for functional description and configuration requirements.

Subscriber usage:

dfesp_db_adapter -c *connectstring* -h *url* -k sub -t *tablename* <-C [*configfilesection*]> <-E *tokenlocation* > <-g *gdconfig* > <-j trace | debug | info | warn | error | fatal | off> <-l native | solace | tervela | rabbitmq | kafka> <-N *maxcolbinding* > <-q> <-V> <-X> <-x *commitrows* > <-y *logconfigfile* > <-Z *transportconfigfile*> <-z *commitsecs* >

Publisher usage:

dfesp_db_adapter -c *connectstring* -h *url* -k pub <-B *logminerdbname* > <-b *blocksize* > <-C [*configfilesection*]> <-D *logminerstartdatetime* > <-E *tokenlocation* > <-e> <-G> <-g *gdconfig* > <-j trace | debug | info | warn | error | fatal | off> <-l native | solace | tervela | rabbitmq | kafka> <-m *maxevents*> <-N *maxcolbinding* > <-O> <-Q> <-R> <-S *logminerschemaowner* > <-s *selectstatement* > <-T *logminertablename* > <-V> <-X> <-y *logconfigfile* > <-Z *transportconfigfile*>

Parameter	Description
`-B logminerdbname`	Specifies the `gpperfmon` database that contains the `queries_history` table.

Parameter	Description
`-b blocksize`	Specifies the block size. The default value is 1.
`-C [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]`.
`-c connectstring`	Specifies the driver type and other driver–specific parameters.
`-D logminerstartdatetime`	Specifies the start date time for Oracle or Greenplum log miner mode. Use the following format: `"dd-mmm-yyyy hh:mm:ss"`
`-E tokenlocation`	Specifies the location of the file in the local filesystem that contains the OAuth token required for authentication by the publish/subscribe server
`-e`	Specifies that events are transactional.
`-G`	Enables Greenplum log miner mode.
`-g gdconfig`	Specifies the guaranteed delivery configuration file.
`-h url`	Specifies the dfESP publish and subscribe 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.
`-j 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.
`-k`	Specifies “sub” for subscriber use and “pub” for publisher use
`-l 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.
`-m maxevents`	Specifies the maximum number of events to publish.
`-N maxcolbinding`	Specifies the maximum supported width of string columns. The default value is 4096.
`-O`	Enables Oracle log miner mode.
`-Q`	For the publisher, quiesces the project after all events are injected into the Source window.

Parameter	Description
`-q`	Ignores errors when executing SQL Inserts, Updates, or Deletes.
`-R`	Builds events with `opcode=Upsert` instead of `Insert`.
`-S logminerschemaowner`	Specifies the schema owner for Oracle or Greenplum log miner mode.
`-s selectstatement`	Specifies the publisher source database SQL statement.
`-T logminertablename`	Specifies the table name for Oracle or Greenplum log miner mode.
`-t tablename`	Specifies the subscriber target database table.
`-V`	Restarts the adapter if a fatal error is reported.
`-X`	Specifies that the `pwd` field in `connectstring` is encrypted.
`-x commitrows`	Specifies the minimum number of rows to buffer.
`-y logconfigfile`	Specifies the log configuration file.
`-Z transportconfigfile`	Specifies the publish/subscribe transport configuration file. The default value depends on the transport type specified with the `-l` parameter: For `-l solace`, the default is **`./solace.cfg`**. For `-l tervela`, the default is **`./client.config`**. For `-l rabbitmq`, the default is **`./rabbitmq.cfg`**. For `-l kafka`, the default is **`./kafka.cfg`**. **Note:** No transport configuration file is required for native transport.
`-z commitsecs`	Specifies the maximum number of seconds to hold onto an incomplete commit buffer.

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

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.

Usage:

dfesp_esp_adapter –p *url* -s *url*

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

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: Advanced Topics*.

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`**
- **`json`**
- **`sashdat`** (supports only subscribe operations, and only as a client type socket connector)
- **`syslog`** (supports only publish operations)
- **`xml`**

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

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

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

Table 11 Required Parameters for File and Socket Connectors

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

Table 12 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 13 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/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 `ESP_DATETIME` and `ESP_TIMESTAMP` fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (`ESP_DATETIME`) or microseconds (`ESP_TIMESTAMP`) since epoch. The `dateformat` parameter accepts any time format that is supported by the UNIX `strftime` function.
`doubleprecision`	Specifies the number of fractional digits in the ASCII representation of a double. The default value is 6.
`hdatcashostport`	Specifies the CAS server host and port. Applies only to the **`hdat`** connector.
`hdatcaspassword`	Specifies the CAS server password. Applies only to the **`hdat`** connector.
`hdatcasusername`	Specifies the CAS server user name. Applies only to the **`hdat`** connector.

Parameter	Description
hdatfilename	Specifies the name of the Objective Analysis Package Data (HDAT) file to be written to the Hadoop Distributed File System (HDFS). Include the full path, as shown in the HDFS browser when you browse the name node specified in the `fsname` parameter. Do not include the `.sashdat` extension. Applies only to and is required for the **`hdat`** connector.
hdatlasrhostport	Specifies the SAS LASR Analytic Server host and port. Applies only to the **`hdat`** connector.
hdatlasrkey	Specifies the path to **`tklasrkey.sh`**. By default, this file is located in **`$DFESP_HOME/bin`**. Applies only to the **`hdat`** connector.
hdatmaxdatanodes	Specifies the maximum number of data node connections. The default value is the total number of live data nodes known by the name node. This parameter is ignored when `hdatnumthreads` <=1. Applies only to the **`hdat`** connector.
hdatmaxstringlength	Specifies in bytes the fixed size of string fields in Objective Analysis Package Data (HDAT) files. You must specify a multiple of 8. Strings are padded with spaces. Applies only to and is required for the **`hdat`** connector. To specify unique string lengths per column, configure a comma-separated string of values, using no spaces. Every value must be a multiple of 8, and the number of values must be equal to the number of string fields in the subscribed window schema.
hdatnumthreads	Specifies the size of the thread pool used for multi-threaded writes to data node socket connections. A value of 0 or 1 indicates that writes are single-threaded and use only a name-node connection. Applies only to and is required for the **`hdat`** connector.
hdfsblocksize	Specifies in Mbytes the block size used to write an Objective Analysis Package Data (HDAT) file. Applies only to and is required for the **`hdat`** connector.
hdfsnumreplicas	Specifies the number of Hadoop Distributed File System (HDFS) replicas created with writing an Objective Analysis Package Data (HDAT) file. The default value is 1. Applies only to the **`hdat`** connector.
header	`false | true | full` For a CSV subscriber, specifies to write a header row that shows comma-separated fields. Set the value to `full` to include `opcode, flags,` in the header line.
maxfilesize	Specifies the maximum size in bytes of the subscriber output file. When reached, a new output file is opened. When configured, a timestamp is appended to all output filenames. This parameter does not apply to socket connectors with `fstype` other than **`hdat`**.
periodicity	Specifies the interval in seconds at which the subscriber output file is closed and a new output file opened. When configured, a timestamp is appended to all output filenames for when the file was opened. This parameter does not apply to socket connectors with `fstype` other than **`hdat`**.
rate	When latency mode is enabled, shows this specified rate in generated output files.

Parameter	Description
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.

Table 14 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 input 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 ESP_DATETIME and ESP_TIMESTAMP fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (ESP_DATETIME) or microseconds (ESP_TIMESTAMP) since epoch. The dateformat parameter accepts any time format that is supported by the UNIX strftime function.
growinginputfile	Enables reading from a growing input file by publishers. When enabled, the publisher reads indefinitely from the input file until the connector is stopped or the server drops the connection. This parameter does not apply to socket connectors.
header	Specifies the number of input lines to skip before starting publish operations. The default value is 0. This parameter applies only to csv and syslog publish connectors.
ignorecsvparseerrors	Specifies that when a field in an input CSV event cannot be parsed, the event is dropped, an error is logged, and publishing continues.
maxevents	Specifies the maximum number of events to publish.
noautogenfield	Specifies that input events are missing the key field that is autogenerated by the source window.

Parameter	Description
`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`.
`rate`	Specifies the requested transmit rate in events per second.
`repeatcount`	Specifies the number of times to repeat the publish operation. Applies to publishers that publish a finite number of events. The default value is 0.
`transactional`	Sets the event block type to transactional. The default event block type is normal.

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. If 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.

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`

`opcode` **Tag Value**	**Opcode**
`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.

JSON File and Socket Connector Data Format

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

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

The outermost brackets define an array of event blocks. Each nested pair of brackets defines an array of events that corresponds to an event block. See "Publish/Subscribe API Support for JSON Messaging" in *SAS Event Stream Processing: Publish/Subscribe API* for the semantics to convert this array to and from an event block.

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

Syslog File and Socket Connector Notes

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

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

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

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

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

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

A sample schema is as follows:

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

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 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.lasr.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 run 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.lasr.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.

Common Event Format (CEF) File and Socket Connector

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

The Source window schema must start with the following fields:

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

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

```
index*:int32,datetime:date,hostname:string,version:int32,devicevendor:string,device
product: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"
```

Using the File and Socket Adapter

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)

Subscriber use:

dfesp_fs_adapter -f *fsname* - h *url* -k sub -t binary | csv | xml | json | hdat <-a *aggrsize* > <-B> <-C [*configfilesection*]> <-c *period* > <-d *dateformat* > <-E *tokenlocation* > <-g *gdconfig* > <-H *hdatlasrhostport* > <-j trace | debug | info | warn | error | fatal | off> <-K *hdatalasrkey* > <-l native | solace | tervela | rabbitmq | kafka><-M true | false | full> <-N *latencyblksize* > <-n> <-o *hdat_filename* > <-P *hdatcashostport*> <-q *hdat_max_data_nodes* > <-r *rate* > <-s *maxfilesize* > <-U *hdatcasusername*> <-u *hdfs_blocksize* > <-V> <-v *hdfs_numreplicas* > <-W *hdatcaspassword*> <-w *hdat_numthreads* > <-y *log_configfile* > <-z *hdat_max_stringlength* > <-Z *transportconfigfile*> <-3 *doubleprecision*>

Publisher use:

dfesp_fs_adapter -f *fsname* -h *url* -k pub -t binary | csv | xml | json | syslog | cef < -A> <-b *blocksize* > <-C [*configfilesection*]> < -D *csvfielddelimiter* > <-d *dateformat* > <-E *tokenlocation* > <-e> <-F *eventtype* > <-G *rampupsecs* > <-g *gdconfig* > <-i> <-I> <-J *repeatcount*> <-j trace | debug | info | warn | error | fatal | off> <-l native | solace | tervela | rabbitmq | kafka> <-m *maxevents* > <-n > <-O> <-p > <-Q> <-R> <-r *rate* > <-S> <-V> <-x *header* > <-y *log_configfile* > <-Z *transportconfigfile*>

Parameter	Description
`-A`	Specifies that input events are missing the key field that is autogenerated by the Source window.
`-a aggrsize`	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.
`-B`	Specifies to create an unbuffered stream when writing to a file or socket. By default, streams are buffered.
`-b blocksize`	Sets the block size. The default value is 1.

Parameter	Description
`-C [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]`.
`-c period`	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
`-D csvfielddelimiter`	Specifies the character delimiter for field data in input CSV events. The default delimiter is the `,` character.
`-d dateformat`	Specifies the format of `ESP_DATETIME` and `ESP_TIMESTAMP` fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (`ESP_DATETIME`) or microseconds (`ESP_TIMESTAMP`) since epoch. The `dateformat` parameter accepts any time format that is supported by the UNIX `strftime` function.
`-E tokenlocation`	Specifies the location of the file in the local filesystem that contains the OAuth token required for authentication by the publish/subscribe server.
`-e`	Specifies that events are transactional.
`-F eventtype`	Specifies the event type to Insert into input CSV events (with comma). Valid values are `"normal"` and `"partialupdate"`.
`-f fsname`	Specifies the subscriber output file, publisher input file, or socket “`host:port`". Leave *host* blank to implement a server.
`-G rampupsecs`	Specifies the number of seconds to linearly ramp up from 0 to the transmit rate that you request with `-r`. The default is 0
`-g gdconfig`	Specifies the guaranteed delivery configuration file.
`-H hdatlasrhostport`	Specifies the LASR Analytic Server host and port.
`-h url`	Specifies the dfESP publish and subscribe standard URL in the form `dfESP://host:port/project/continuousquery/window` Append the following for subscribers: `?snapshot=true | false`. Append the following for subscribers if needed: 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. ■ `?collapse=true | false` ■ `?rmretdel=true | false`
`-I`	Specifies that when a field in an input CSV event cannot be parsed, the event is dropped, an error is logged, and publishing continues.
`-i`	Reads from growing file.

Parameter	Description
`-J 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.
`-j 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.
`-K hdatlasrkey`	Specifies the path to tklasrkey.sh. By default, this file is located in **`$DFESP_HOME/bin`**
`-k`	Specifies `sub` for subscriber use and `pub` for publisher use
`-l 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.
`-M true | false | full`	For a CSV subscriber, writes a header row that shows comma-separated field names. Set to "full" to include "opcode, flags," in header line.
`-m maxevents`	Specifies the maximum number of events to publish.
`-N latencyblksize`	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 `-n`.
`-n`	Specifies latency mode.
`-O`	Prepends an opcode and comma to read CSV events. The opcode is Insert unless `-R` is enabled.
`-o hdat_filename`	Specifies the filename to write to Hadoop Distributed File System (HDFS).
`-P hdatcashostport`	Specifies the CAS server host and port.
`-p`	Buffers all event blocks before publishing them.
`-Q`	For the publisher, quiesces the project after all events are injected into the Source window.
`-q hdat_max_datanodes`	Specifies the maximum number of data nodes. The default value is the number of live data nodes.
`-R`	Builds events with `opcode=Upsert` instead of `Insert`.
`-r rate`	Specifies the requested transmit rate in events per second for publishers. For subscribers, writes the rate to write files when latency mode is enabled.
`-S`	Specifies that CEF events contain the syslog prefix. By default, they do not.
`-s maxfilesize`	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.

Parameter	Description
`-t binary | csv | xml | json | syslog | hdat | cef`	Specifies the file system type. The `syslog` and `cef` values are valid only for publishers. The `hdat` value is valid only for subscribers.
`-U hdatcasusername`	Specifies the CAS server user name.
`-u hdfs_blocksize`	Specifies the Hadoop Distributed File System (HDFS) block size in MB.
`-V`	Specify to restart the adapter if a fatal error is reported.
`-v hdfs_numreplicas`	Specifies the Hadoop Distributed File System (HDFS) number of replicas. The default value is 1.
`-W hdatcaspassword`	Specifies the CAS server password.
`-w hdat_numthreads`	Specifies the adapter thread pool size. A value <= 1 means that no data nodes are used.
`-x header`	Specifies the number of header lines to ignore.
`-y logconfigfile`	Specifies the log configuration file.
`-Z transportconfigfile`	Specifies the publish/subscribe transport configuration file. The default value depends on the transport type specified with the `-l` parameter: For `-l solace`, the default is **`./solace.cfg`**. For `-l tervela`, the default is **`./client.config`**. For `-l rabbitmq`, the default is **`./rabbitmq.cfg`**. For `-l kafka`, the default is **`./kafka.cfg`**. **Note:** No transport configuration file is required for native transport.
`-z hdat_max_stringlength`	Specifies the fixed size to use for string fields in HDAT records. The value must be a multiple of 8.
`-3 doubleprecision`	Specifies the number of fractional digits in the ASCII representation of a double. The default value is 6.

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

Using the HDAT Reader Adapter

The HDAT reader adapter resides in dfx-esp-hdatreader-adapter.jar, which bundles the Java publisher SAS Event Stream Processing client. The adapter converts each row in an HDAT 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 `-s` parameter.

The number of fields in the target Source window schema must match the number of columns in the HDAT row data. Also, all HDAT 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:

```
$ 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/*
```

Usage:

dfesp_hdat_publisher -f *hdfs* -i *inputfile* -u *url* <-b *blocksize* > <-c [*configfilesection*]> <-g *gdconfigfile* > <-l native | solace | tervela | rabbitmq | kafka> <-O *tokenlocation* > <-o severe | warning | info> <-Q> <-s> <-t> <-V> <-W *maxevents*> <-Z *transportconfigfile*>

Parameter	Description
`-b blocksize`	Specifies the number of events per event block.
`-c [configfilesection]`	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.
`-f hdfs`	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.
`-g gdconfigfile`	Specifies the guaranteed delivery configuration file for the client.
`-i inputfile`	Specifies the input CSV file, in the form `"/path/filename.csv"`.
`-l native | solace | tervela | rabbitmq | kafka`	Specifies the transport type. When you specify `solace`, `tervela`, `rabbitmq`, or `kafka` transports instead of the default `native` transport, use the required client configuration files specified in "Using Alternative Transport Libraries for Java Clients" in *SAS Event Stream Processing: Publish/Subscribe API*.
`-O tokenlocation`	Specifies the location of the file in the local file system that contains the OAuth token required for authentication by the publish/subscribe server.
`-o severe | warning | info`	Specifies the application logging level.
`-Q`	Quiesces the project after all events are injected into the Source window. **`javaadapters.config`** parameter name: `quiesceproject`
`-s`	Builds events with `opcode=Upsert` instead of `Insert`. **`javaadapters.config`** parameter name: `publishwithupsert`
`-t`	Specifies that event blocks are transactional. The default event block type is normal. **`javaadapters.config`** parameter name: `transactional`

Parameter	Description
`-u url`	Specifies the publish standard URL in the form `"dfESP://host:port/project/continuousquery/window"`.
`-V`	Restarts the adapter if a fatal error is reported. **`javaadapters.config`** parameter name: `restartonerror`
`-W maxevents`	Specifies the maximum number of events to publish.
`-Z transportconfigfile`	Specifies the publish/subscribe transport configuration file. The default value depends on the transport type specified with the `-l` parameter: For `-l solace`, the default is **`./solace.cfg`**. For `-l tervela`, the default is **`./client.config`**. For `-l rabbitmq`, the default is **`./rabbitmq.cfg`**. For `-l kafka`, the default 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: Advanced Topics*.

Using the HDFS (Hadoop Distributed File System) Adapter

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

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

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

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

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

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

For example, in Linux you would run the following from the command line:

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

Subscriber usage:

dfesp_hdfs_subscriber -f *hdfs* -t *outputfile* -u *url* <-b *hdfsblocksize* > <-c [*configfilesection*]> <-d *dateformat* > <- g *gdconfigfile* > <-j *jaasconf*> <-k *krb5conf*> <-l native | solace | tervela | rabbitmq | kafka> <-m *maxfilesize* >

<-n *hdfsnumreplicas* > <-O *tokenlocation* > <-o severe | warning | info> <-P *opaquestringfield*> <-p *periodicity* > <-V> <-Z *transportconfigfile*>

Parameter	**Description**
`-b hdfsblocksize`	Specifies the HDFS block size in MB. The default value is 64.
`-c [configfilesection]`	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.
`-d dateformat`	Specifies the format of `ESP_DATETIME` and `ESP_TIMESTAMP` fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (`ESP_DATETIME`) or microseconds (`ESP_TIMESTAMP`) since epoch. The `dateformat` parameter accepts any time format that is supported by the UNIX `strftime` function.
`-f hdfs`	Specifies the target file system, in the form `"hdfs://host:port"`. Specifies the file system normally configured in property `fs.defaultFS` in file core-site.xml.
`-g gdconfigfile`	Specifies the guaranteed delivery configuration file for the client.
`-j jaasconf`	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.
`-k krb5conf`	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).
`-l native | solace | tervela | rabbitmq | kafka`	Specifies the transport type. When you specify `solace`, `tervela`, `rabbitmq`, or `kafka` transports instead of the default `native` transport, use the required client configuration files specified in "Using Alternative Transport Libraries for Java Clients" in *SAS Event Stream Processing: Publish/Subscribe API*.
`-m maxfilesize`	Specifies the output file periodicity in bytes.
`-n hdfsnumreplicas`	Specifies the HDFS number of replicas. The default value is 1.
`-O tokenlocation`	Specifies the location of the file in the local file system that contains the OAuth token required for authentication by the publish/subscribe server.
`-o severe | warning | info`	Specifies the application logging level.
`-p periodicity`	Specifies the output file periodicity in seconds.
`-P opaquestringfield`	Specifies the subscribed window field from which to extract the output line.

Parameter	Description
`-t outputfile`	Specifies the output CSV file, in the form `"/path/filename.csv"`.

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

`-V`	Restarts the adapter if a fatal error is reported. **`javaadapters.config`** parameter name:`restartonerror`
`-Z transportconfigfile`	Specifies the publish/subscribe transport configuration file. The default value depends on the transport type specified with the `-l` parameter: For `-l solace`, the default is **`./solace.cfg`**. For `-l tervela`, the default is **`./client.config`**. For `-l rabbitmq`, the default is **`./rabbitmq.cfg`**. For `-l kafka`, the default is **`./kafka.cfg`**. **Note:** No transport configuration file is required for native transport.

Publisher usage:

dfesp_hdfs_publisher -f *hdfs* –i *inputfile* -u *url* <-b *blocksize* > <-C> <-c [*configfilesection*]> <-d *dateformat* > < -e> <-F *eventtype*> <- g *gdconfigfile* > <-j *jaasconf*> <-k *krb5conf*> <-l native | solace | tervela | rabbitmq | kafka> < -m *csvfielddelimiter* > <-n > <-O *tokenlocation* > <-o severe | warning | info> <-Q> <-q> <-s> <-t> <-V> <-W *maxevents*> <-Z *transportconfigfile*>

Parameter	Description
`-b blocksize`	Specifies the number of events per event block.
`-C`	Prepends an opcode and comma to read CSV events. The opcode is Insert unless `-s` is enabled. **`javaadapters.config`** parameter name: `addcsvopcode`
`-c [configfilesection]`	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.

Parameter	Description
`-d dateformat`	Specifies the format of `ESP_DATETIME` and `ESP_TIMESTAMP` fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (`ESP_DATETIME`) or microseconds (`ESP_TIMESTAMP`) since epoch. The `dateformat` parameter accepts any time format that is supported by the UNIX `strftime` function.
`-e`	Specifies that when a field in an input CSV event cannot be parsed, the event is dropped, an error is logged, and publishing continues. **`javaadapters.config`** parameter name: `ignorecsvparseerrors`
`-F eventtype`	Specifies the event type to Insert into input CSV events (with comma). Valid values are `"normal"` and `"partialupdate"`.
`-f hdfs`	Specifies the target file system, in the form `"hdfs://host:port"`.
`-g gdconfigfile`	Specifies the guaranteed delivery configuration file for the client.
`-i inputfile`	Specifies the input CSV file, in the form `"/path/filename.csv"`.
`-j jaasconf`	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.
`-k krb5conf`	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).
`-l native | solace | tervela | rabbitmq | kafka`	Specifies the transport type. When you specify `solace`, `tervela`, `rabbitmq`, or `kafka` transports instead of the default `native` transport, use the required client configuration files specified in "Using Alternative Transport Libraries for Java Clients" in *SAS Event Stream Processing: Publish/Subscribe API*.
`-m csvfielddelimiter`	Specifies the character delimiter for field data in input CSV events. The default delimiter is the `,` character.
`-n`	Specifies that input events are missing the key field that is autogenerated by the source window. **`javaadapters.config`** parameter name: `noautogenfield`
`-O tokenlocation`	Specifies the location of the file in the local file system that contains the OAuth token required for authentication by the publish/subscribe server.
`-o severe | warning | info`	Specifies the application logging level.
`-Q`	Quiesces the project after all events are injected into the Source window. **`javaadapters.config`** parameter name: `quiesceproject`

Parameter	Description
`-q`	Specifies that every line read from the file should be treated as an opaque string. The Source window schema is assumed to be `"index:int64,message:string"`. **`javaadapters.config`** parameter name: `opaquestring`
`-s`	Builds events with `opcode=Upsert` instead of `opcode=Insert`. **`javaadapters.config`** parameter name: `publishwithupsert`
`-t`	Specifies that event blocks are transactional. The default event block type is normal. **`javaadapters.config`** parameter name: `transactional`
`-u url`	Specifies the publish standard URL in the form `"dfESP://host:port/project/continuousquery/window"`.
`-V`	Restarts the adapter if a fatal error is reported. **`javaadapters.config`** parameter name: `restartonerror`
`-W maxevents`	Specifies the maximum number of events to publish.
`-Z transportconfigfile`	Specifies the publish/subscribe transport configuration file. The default value depends on the transport type specified with the `-l` parameter: For `-l solace`, the default is **`./solace.cfg`**. For `-l tervela`, the default is **`./client.config`**. For `-l rabbitmq`, the default is **`./rabbitmq.cfg`**. For `-l kafka`, the default 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: Advanced Topics*.

Using the Java Message Service (JMS) Adapter

The Java Message Service (JMS) adapter resides in dfx-esp-jms-adapter.jar, which 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 subscriber client requires a command line parameter that defines the type of JMS message used to contain events. The publisher client consumes the following JMS message types:

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

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

The JMS password must be passed in unencrypted form unless `-E` is configured. The encrypted version of the password can be generated using OpenSSL, which must be installed on your system. When you install the SAS Event Stream Processing System Encryption and Authentication Overlay, you install the included OpenSSL executable. Use the following command on the console to use OpenSSL to display your encrypted password:

```
echo "jmspassword" | openssl enc -e -aes-256-cbc -a -salt -pass
pass:"SASespJMSadapterUsedByUser=jmsuserid"
```

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

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

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

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

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

Subscriber usage:

dfesp_jms_subscriber -m BytesMessage | TextMessage | MapMessage -u *url* <-a *jndifactname* > <-c [*configfilesection*]> <-d *dateformat* > <-E> <-f *protofile* > <- g *gdconfigfile* > <-i *jmsuserid* > <-j *jndidestname* > <-L> <-l native | solace | tervela | rabbitmq | kafka> <-M> <-O *tokenlocation* > <-o severe | warning | info> <-P *opaquestringfield*> <-p *jmspassword* > <-r *protomsg* ><-V><-x native | axeda> <-Z *transportconfigfile*>

Parameter	Description
`-a jndifactname`	Specifies the JNDI name to look up for the JMS connection factory. The default value is `"ConnectionFactory"`. This option overrides settings in **`jndi.properties`**.
`-c [configfilesection]`	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.
`-d dateformat`	Specifies the format of `ESP_DATETIME` and `ESP_TIMESTAMP` fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (`ESP_DATETIME`) or microseconds (`ESP_TIMESTAMP`) since epoch. The `dateformat` parameter accepts any time format that is supported by the UNIX `strftime` function.
`-E`	Specifies that the JMS password is encrypted. **`javaadapters.config`** parameter name: `pwdencrypted`
`-f protofile`	Specifies the **`.proto`** file that contains the message used for Google Protocol buffer support.

Parameter	Description
`-g gdconfigfile`	Specifies the guaranteed delivery configuration file for the client.
`-i jmsuserid`	Specifies the user ID for the JMS destination.
`-j jndidestname`	Specifies the JNDI name to look up for the JMS destination. The default value is `"jmsDestination"`. This option overrides settings in **`jndi.properties`**.
`-L`	For CSV, specifies to send one message per event block. The default is one message per transactional event block and one message per event in a non-transactional event block. In a multi-event message, CSV lines are delimited by the "line.separator" system property. **`javaadapters.config`** parameter name: `csvmsgpereventblock`
`-l native | solace | tervela | rabbitmq | kafka`	Specifies the transport type. When you specify `solace`, `tervela`, `rabbitmq`, or `kafka` transports instead of the default `native` transport, use the required client configuration files specified in "Using Alternative Transport Libraries for Java Clients" in *SAS Event Stream Processing: Publish/Subscribe API*.
`-M`	For CSV, specifies to send one message per event. The default is one message per transactional event block and one message per event in a non-transactional event block. In a multi-event message, CSV lines are delimited by the "line.separator" system property. **`javaadapters.config`** parameter name: `csvmsgperevent`
`-m BytesMessage | TextMessage | MapMessage`	Specifies the JMS message type.
`-O tokenlocation`	Specifies the location of the file in the local file system that contains the OAuth token required for authentication by the publish/subscribe server.
`-o severe | warning | info`	Specifies the application logging level.
`-P opaquestringfield`	Specifies the subscribed window field from which to extract the JMS TextMessage.
`-p jmspassword`	Specifies the password for the JMS destination.
`-r protomsg`	Specifies the message itself in the **`.proto`** file that is specified by the protofile parameter.

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

Parameter	Description
`-V`	Restarts the adapter if a fatal error is reported. **`javaadapters.config`** parameter name: `restartonerror`
`-x native | axeda`	Specifies the XML format for TextMessage. Specifies `native` when the message is in CSV format.
`-Z transportconfigfile`	Specifies the publish/subscribe transport configuration file. The default value depends on the transport type specified with the `-l` parameter: For `-l solace`, the default is **`./solace.cfg`**. For `-l tervela`, the default is **`./client.config`**. For `-l rabbitmq`, the default is **`./rabbitmq.cfg`**. For `-l kafka`, the default is **`./kafka.cfg`**. **Note:** No transport configuration file is required for native transport.

Publisher usage:

dfesp_jms_publisher -u *url* <-a *jndifactname* > <-b *blocksize* > <-C> <-c [*configfilesection*]> <-d *dateformat* > <-E> <-e> <-F *eventtype*> <-f *protofile* > <- g *gdconfigfile* > <-i *jmsuserid* > <-j *jndidestname* > <-l native | solace | tervela | rabbitmq | kafka> < -m *csvfielddelimiter* > <-n > <-O *tokenlocation* > <-o severe | warning | info> <-p *jmspassword* > <-q> <-r *protomsg* > <-s> <-t> <-V> <-x native | axeda> <-W *maxevents*> <-Z *transportconfigfile*>

Parameter	Description
`-a jndifactname`	Specifies the JNDI name to look up for the JMS connection factory. The default value is `"ConnectionFactory"`. This option overrides settings in **`jndi.properties`**.
`-b blocksize`	Specifies the number of events per event block.
`-C`	Prepends an opcode and comma to input CSV events, or received messages of type = MapMessage that are missing the "eventOpcode" field. The opcode is Insert unless `-s` is enabled. **`javaadapters.config`** parameter name: `addcsvopcode`
`-c [configfilesection]`	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.
`-d dateformat`	Specifies the format of `ESP_DATETIME` and `ESP_TIMESTAMP` fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (`ESP_DATETIME`) or microseconds (`ESP_TIMESTAMP`) since epoch. The `dateformat` parameter accepts any time format that is supported by the UNIX `strftime` function.
`-E`	Specifies that the JMS password is encrypted. **`javaadapters.config`** parameter name: `pwdencrypted`

Parameter	Description
`-e`	Specifies that when a field in an input CSV event cannot be parsed, the event is dropped, an error is logged, and publishing continues. **`javaadapters.config`** parameter name: `ignorecsvparseerrors`
`-F eventtype`	Specifies the event type to Insert into input CSV events (with comma), or received messages of type = MapMessage that are missing the "eventFlags" field. Valid values are `"normal"` and `"partialupdate"`.
`-f protofile`	Specifies the **`.proto`** file that contains the message used for Google Protocol buffer support.
`-g gdconfigfile`	Specifies the guaranteed delivery configuration file for the client.
`-i jmsuserid`	Specifies the user ID for the JMS destination.
`-j jndidestname`	Specifies the JNDI name to look up for the JMS destination. The default value is `"jmsDestination"`. This option overrides settings in **`jndi.properties`**.
`-l native | solace | tervela | rabbitmq | kafka`	Specifies the transport type. When you specify `solace`, `tervela`, `rabbitmq`, or `kafka` transports instead of the default `native` transport, use the required client configuration files specified in “Using Alternative Transport Libraries for Java Clients” in *SAS Event Stream Processing: Publish/Subscribe API*.
`-o severe | warning | info`	Specifies the application logging level.
`-m csvfielddelimiter`	Specifies the character delimiter for field data in input CSV events. The default delimiter is the `,` character.
`-n`	Specifies that input events are missing the key field that is autogenerated by the source window. **`javaadapters.config`** parameter name: `noautogenfield`
`-O tokenlocation`	Specifies the location of the file in the local file system that contains the OAuth token required for authentication by the publish/subscribe server.
`-p jmspassword`	Specifies the password for the JMS destination.
`-q`	Specifies that a received JMS string message should be treated as an opaque string. The source window schema is assumed to be `"index:int64,message:string"`. **`javaadapters.config`** parameter name: `opaquestring`
`-r protomsg`	Specifies the message itself in the **`.proto`** file that is specified by the protofile parameter.
`-s`	Builds events with `opcode=Upsert` instead of `Insert`. **`javaadapters.config`** parameter name: `publishwithupsert`
`-t`	Specifies that event blocks are transactional. The default event block type is normal. **`javaadapters.config`** parameter name: `transactional`

Parameter	Description
`-u url`	Specifies the publish standard URL in the form "`dfESP://host:port/project/continuousquery/window`"
`-V`	Restarts the adapter if a fatal error is reported. **`javaadapters.config`** parameter name: `restartonerror`
`-W maxevents`	Specifies the maximum number of events to publish.
`-x native | axeda`	Specifies the XML format for TextMessage. Specifies `native` when the message is in CSV format.
`-Z transportconfigfile`	Specifies the publish/subscribe transport configuration file. The default value depends on the transport type specified with the `-l` parameter: For `-l solace`, the default is **`./solace.cfg`**. For `-l tervela`, the default is **`./client.config`**. For `-l rabbitmq`, the default is **`./rabbitmq.cfg`**. For `-l kafka`, the default 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: Advanced Topics*.

Using the Kafka Connector and Adapter

Using the Kafka Connector for Kafka 0.9 and MapR Streams

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

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

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

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

Event blocks transmitted through a Kafka cluster can be encoded as binary, CSV, Google protobufs, or JSON 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. Zookeeper C client libraries must be installed even if hot failover operation is not enabled. The connector was tested using Zookeeper version 3.4.8.

You must install Kafka client run-time libraries on the platform that hosts the running instance of the connector. The connector uses the LibrdKafka C and C++ libraries. You can download these libraries from https://github.com/edenhill/librdkafka. Use version 0.9.5 with SAS Event Stream Processing version 4.3 or higher. The run-time environment must define the path to those libraries (for example, specifying **`LD_LIBRARY_PATH`** on Linux platforms). The path must include both the C LibrdKafka library and the C++ LibrdKafka library. The path must also include the Zookeeper libraries, which can be downloaded from http://www.apache.org/dyn/closer.cgi/zookeeper/. Make sure the LibrdKafka and Zookeeper libraries are built on the machine where the connector runs, so that the same system libraries used while building are available at run time. To build the libraries with SSL support, on the ./configure step, use option --`LDFLAGS=-L$DFESP_HOME/lib` so that LibrdKafka is built against the same libraries as SAS Event Stream Processing.

Except for the relevant connector parameters described below, all LibrdKafka configuration parameters are left at default values by the connector. The user 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.

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.

Table 15 Required Parameters for the Kafka Subscriber Connector

Parameter	Description
`type`	Specifies to subscribe. Must be "`sub`".
`kafkahostport`	Specifies one or more Kafka brokers in the following form: `host:port,host:port,...`.
`kafkatopic`	Specifies the Kafka topic.
`kafkapartition`	Specifies the Kafka partition. Specify `-1` to use all partitions in the topic. This value is not supported when hot failover is enabled.
`kafkatype`	Specifies **`binary`**, **`CSV`**, **`JSON`**, or the name of a string field in the subscribed window schema. For **`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 '_'.

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 16 Required Parameters for the Kafka Publisher Connector

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

Table 17 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 ESP_DATETIME and ESP_TIMESTAMP fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (ESP_DATETIME) or microseconds (ESP_TIMESTAMP) 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.

Parameter	Description
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.
csvincludeschema	When kafkatype=CSV, specifies when to prepend output CSV data with the window's serialized schema. Valid values are never, once, and pereventblock. The default value is “never”.
useclientmsgid	Uses the client-generated message ID instead of the engine-generated message ID when performing a failover operation and extracting a message ID from an event block.
configfilesection	Specifies the name of the section in **/opt/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.
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.

Table 18 Optional Parameters for the Kafka Publisher Connector

Parameter	Description
transactional	When kafkatype=CSV, sets the event block type to transactional. The default event block type is normal.
blocksize	When kafkatype=CSV, specifies the number of events to include in a published event block. The default value is 1.
dateformat	Specifies the format of ESP_DATETIME and ESP_TIMESTAMP fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (ESP_DATETIME) or microseconds (ESP_TIMESTAMP) 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.

Parameter	Description
protofile	Specifies the **.proto** file that contains the Google Protocol Buffers message definition used to convert event blocks to protobuf messages. When you specify this parameter, you must also specify the protomsg parameter.
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].
csvfielddelimiter	Specifies the character delimiter for field data in input CSV events. The default delimiter is the , character.
noautogenfield	Specifies that input events are missing the key field that is autogenerated by the source window.
publishwithupsert	Specifies to build events with opcode = Upsert instead of opcode = Insert.
kafkainitialoffset	Specifies the offset from which to begin consuming messages from the Kafka topic and partition. Valid values are smallest, largest, or an integer. The default value is smallest.
addcsvopcode	Prepends an opcode and comma to write CSV events. The opcode is Insert unless publishwithupsert is enabled.
addcsvflags	Specifies the event type to insert into input CSV events (with a comma). Valid values are normal and partialupdate.
kafkaglobalconfig	Specifies a semicolon-separated list of key=value strings to configure librdkafka global configuration values.
kafkatopicconfig	Specifies a semicolon-separated list of key=value strings to configure librdkafka topic configuration values.
useclientmsgid	If the Source window has been restored from a persist to disk, ignore received binary event blocks that contain a message ID less than the greatest message ID in the restored window.
maxevents	Specifies the maximum number of events to publish.

Using the Kafka Adapter for Kafka 0.9 and MapR Streams

The Kafka adapter supports publish and subscribe operations on a Kafka cluster. You must install the LibrdKafka C, C++, and Apache Zookeeper libraries to use the adapter. For more information, see the Apache Kafka documentation.

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

Subscriber usage:

dfesp_kafka_adapter -h *url* -k sub -n *numbufferedmsgs* -o *urlhostport* -p *kafkapartition* -s *kafkahostport* -t *kafkatopic* -z binary | csv | json <-C configfilesection> <-d *dateformat* > <-E tokenlocation> <-f *protofile* > <-G *kafkaglobalconfig*> <-g *gdconfig* > <-j trace | debug | info | warn | error | fatal | off> <-L> <-l native | solace | tervela | rabbitmq | kafka> <-M> <-m *protomsg* > <-T *kafkatopicconfig*> <-V> <-w never | once | pereventblock> <-y logconfigfile> <-Z *transportconfigfile*> <-3 *doubleprecision*>

Publisher usage:

dfesp_kafka_adapter -h *url* -k pub -o *urlhostport* -p *kafkapartition* -s *kafkahostport* -t *kafkatopic* -z binary | csv | json | opaquestring <-A> <-b *blocksize* > <-C *configfilesection* > <-D *csvfielddelimiter* > <-d dateformat> <-E *tokenlocation* > <-e > <-F *eventtype*> <-f protofile> <-G *kafkaglobalconfig*> <-g *gdconfig* > <-I> <-j trace | debug | info | warn | error | fatal | off> <-l native | solace | tervela | rabbitmq | kafka> <-m *protomsg* > <-O> <-P *kafkainitialoffset* > <-Q> <-R> <-T *kafkatopicconfig*> <-V> <-Y *maxevents*> <-y logconfigfile> <-Z *transportconfigfile*>

Parameter	Description
`-A`	Specifies that input events are missing the key field that is autogenerated by the Source window.
`-b blocksize`	Specifies the event block size. The default is 1.
`-C [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]`.
`-D csvfielddelimiter`	Specifies the character delimiter for field data in input CSV events. The default delimiter is the `,` character.
`-d dateformat`	Specifies the format of `ESP_DATETIME` and `ESP_TIMESTAMP` fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (`ESP_DATETIME`) or microseconds (`ESP_TIMESTAMP`) since epoch. The `dateformat` parameter accepts any time format that is supported by the UNIX `strftime` function.
`-E tokenlocation`	Specifies the location of the file in the local file system that contains the OAuth token. This token is required for authentication by the publish/subscribe server.
`-e`	Specifies that events are transactional.
`-F eventtype`	Specifies the event type to Insert into input CSV events (with comma). Valid values are `"normal"` and `"partialupdate"`.

Parameter	Description
`-f protofile`	Specifies the **`.proto`** file to be used for Google protocol buffer support.
`-G kafkaglobalconfig`	Specifies a semicolon-separated list of "`key=value`" strings to configure `librdkafka` global configuration values.
`-g gdconfig`	Specifies the guaranteed delivery configuration file.
`-h url`	Specifies the dfESP publish and subscribe standard URL in the form `dfESP://host:port/project/continuousquery/window`. Append the following for subscribers:`?snapshot=false`. **Note:** `?snapshot=true` is invalid. Append the following for subscribers if needed: `?collapse=true | false` `?rmretdel=true`
`-I`	Specifies that when a field in an input CSV event cannot be parsed, the event is dropped, an error is logged, and publishing continues.
`-j trace | debug | info | warn | error | fatal | off`	Specifies the logging level. The default is “`warn`”.
`-k sub | pub`	Specifies `sub` for subscriber or `pub` for publisher.
`-L`	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.
`-l native | solace | tervela | rabbitmq | kafka`	Specifies the publish/subscribe transport. The default is `native`.
`-M`	For CSV, specifies to send one message per event. The default is one message per transactional event block or else one message per event.
`-m protomsg`	Specifies the message itself in the **`.proto`** file that is specified by the protofile parameter.
`-n numbufferedmsgs`	Specifies the maximum number of messages buffered by a standby subscriber connector.
`-O`	Prepends an opcode and comma to write CSV events. The opcode is Insert unless -`R` is enabled.
`-o urlhostport`	Specifies the `host:port` field in the metadata topic to which the connector subscribes (replace ‘:’ with ‘_’)

Parameter	Description
`-P kafkainitialoffset`	Specifies the offset from which to begin consuming from the Kafka partition. Valid values are "smallest", "largest", or an integer. The default is "smallest".
`-p kafkapartition`	Specifies the Kafka partition. Specify `-1` to use all partitions in the topic. This value is not supported when hot failover is enabled.
`-Q`	When *maxevents* is configured, quiesces the project after all events are injected into the Source window.
`-R`	Builds events with `opcode=Upsert` instead of `opcode=Insert`.
`-s kafkahostport`	Specifies one or more Kafka brokers in the following form: `"host:port,host:port,..."`.
`-T kafkatopicconfig`	Specifies a semicolon-separated list of "`key=value`" strings to configure `librdkafka` topic configuration values.
`-t kafkatopic`	Specifies the Kafka topic.
`-w never | once | pereventblock`	When `rmqtype=CSV`, specifies when to prepend output CSV data with the window's serialized schema. The default value is “`never`”.
`-V`	Specifies to restart the adapter if a fatal error is reported.
`-Y maxevents`	Specifies the maximum number of events to publish.
`-y logconfigfile`	Specifies the logging configuration file. By default, there is none.
`-Z transportconfigfile`	Specifies the publish/subscribe transport configuration file. The default value depends on the transport type specified with the `-l` parameter: For `-l solace`, the default is **`./solace.cfg`**. For `-l tervela`, the default is **`./client.config`**. For `-l rabbitmq`, the default is **`./rabbitmq.cfg`**. For `-l kafka`, the default is **`./kafka.cfg`**. **Note:** No transport configuration file is required for native transport.
`-z binary | csv | json | 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"`. For subscribers, the name of a string field in the subscribed window schema is also supported.
`-3 doubleprecision`	Specifies the number of fractional digits in the ASCII representation of a double. The default value is 6.

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

Using the SAS LASR Analytic Server Adapter

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

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

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

- Running this adapter in a UNIX environment requires the installation of an SSH client.
- Running this adapter in a Microsoft Windows environment requires the installation of the SAS Event Stream Processing System Encryption and Authentication Overlay.
- If the SAS LASR Analytic Server is running in SMP mode without SSH support or SAS LASR Analytic Server authentication services support, the LASR adapter must be running on the same computer system as the SAS LASR Analytic Server. 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 my 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, but they 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, `authpassword` must be specified in non-encrypted form for the -`Z` parameter if encryption is not enabled with the -`N` parameter. The encrypted version of the password can be generated using OpenSSL. The OpenSSL executable is included in the SAS Event Stream Processing System Encryption and Authentication Overlay installation. Use the following command on the console to use OpenSSL to display your encrypted password:

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

Use the encrypted password to specify `authpassword` for the -`Z` parameter and enable encryption with the -`N` parameter.

Subscriber usage:

dfesp_lasr_adapter -H *lasrurl* -h *url1*<, *url2,...urlN*> -k sub -t *lasr_library_server_tag.table_name* -X *tklasrkey* <-A *autocommit* > <-a *obstoanalyze* > <-b *blocksize* > <-C [*configfilesection*]> <-d *dateformat* > <-g *gdconfigfile* > <-L native | solace | tervela | rabbitmq | kafka> <-l *loglevel* > <-N> <-n> <-O *tokenlocation* > <-S> <-s *schema*> <-U *authurl* > <-V> <-Y *authusername* > <-y *transportconfigfile*> <-Z *authpassword* > <-z *bufsize* >

Publisher usage:

dfesp_lasr_adapter -H *lasrurl* -h *url1<, url2,...urlN>* -k pub -t *table* -X *tklasrkey* <-b *blocksize* > <-C [*configfilesection*]> <-c> <-d *dateformat* > <-E> <-e> <-g *gdconfigfile* > <-L native | solace | tervela | rabbitmq | kafka> <-l *loglevel* > <-N> <-O *tokenlocation* > < -Q *action* | -q *action*> <-S> <-T *stimeout* > <-U *authurl* <-u> > <-V> <-W *maxevents*> <-Y *authusername* > <-y *transportconfigfile*> <-Z *authpassword* > <-z *bufsize* >

Parameter	**Description**
`-A autocommit`	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.
`-a obstoanalyze`	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.
`-b blocksize`	For a subscriber, specifies the buffer size (number of observations) to be flushed to the server. For a publisher, specifies the event block size to be published to a Source window. The default is 1.
`-C [configfilesection]`	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.
`-c`	Quiesces the project after all events are injected into the Source window. **`javaadapters.config`** parameter name: `quiesceproject`
`-d dateformat`	Specifies the format of `ESP_DATETIME` and `ESP_TIMESTAMP` fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (`ESP_DATETIME`) or microseconds (`ESP_TIMESTAMP`) since epoch. The `dateformat` parameter accepts any time format that is supported by the UNIX `strftime` function.
`-E`	For a publisher, enables caching as data is fetched from a table. By default, caching is disabled. **`javaadapters.config`** parameter name: `fetchallcached`
`-e`	Specifies that event blocks are transactional. By default, event blocks are normal. **`javaadapters.config`** parameter name:`transactional`
`-g gdconfigfile`	Specifies the guaranteed delivery configuration file.
`-H lasrurl`	Specifies the SAS LASR Analytic Server URL in the form `"host:port"`.

Parameter	Description
`-h url`	Specifies the publish and subscribe standard URL in the form `dfESP://host:port/project/continuousquery/window`. You must 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 the following for subscribers if needed: `?collapse=true | false` `?rmretdel=true | false`. **Note:** You can specify multiple URLs.
`-k sub | pub`	Specifies subscribe or publish.
`-L native | solace | tervela | rabbitmq | kafka`	Specifies the transport type. When you specify `solace`, `tervela`, `rabbitmq`, or `kafka` transports instead of the default `native` transport, use the required client configuration files specified in "Using Alternative Transport Libraries for Java Clients" in *SAS Event Stream Processing: Publish/Subscribe API*.
`-l loglevel`	Specifies the Java standard logging level. Valid values are `SEVERE | WARNING | INFO`. The default value is `WARNING`.
`-N`	Specifies that the authentication services password is encrypted. **`javaadapters.config`** parameter name: `pwdencrypted`
`-n`	Creates a new LASR table. By default, the adapter assumes a LASR table already exists. **`javaadapters.config`** parameter name: `newtable`
`-O tokenlocation`	Specifies the location of the file in the local file system that contains the OAuth token that is required for authentication by the publish/subscribe server.
`-Q action`	For a publisher, specifies the action to get cached results from the LASR table. For example, specify `-q "fetch field1 / to=100 where field2=2"`.
`-q action`	For a publisher, specifies the action to get results from the LASR table. For example, specify `-q "fetch field1 / to=100 where field2=2"`.
`-S`	Does not observe strict SAS LASR Analytic Server adapter schema. The default is to observe strict schema. **`javaadapters.config`** parameter name: `nostrictschema`
`-s 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 `-a`, this option is ignored.

Parameter	Description
`-T stimeout`	Specifies the time out interval (in seconds) to wait for each response from the server. By default, there is no time out interval
`-t lasr_library_server_tag.table_name`	Specifies the full name of the LASR table. You must locate the server tag in the metadata properties for the LASR table.
`-U authurl`	Specifies the URL for SAS LASR Analytic Server authentication services.
`-u`	Builds events with `opcode=Upsert`. By default, events are built with `opcode=Insert`. **`javaadapters.config`** parameter name: `publishwithupsert`
`-V`	Restarts the adapter if a fatal error is reported. **`javaadapters.config`** parameter name: `restartonerror`
`-W maxevents`	Specifies the maximum number of events to publish.
`-X tklasrkey`	Specifies the path to **`tklasrkey.sh`** for Linux or **`tklasrkey.bat`** for Microsoft Windows. By default, these files are located in **`$DFESP_HOME/bin`**.
`-Y authusername`	Specifies the user name for SAS LASR Analytic Server authentication services.
`-y transportconfigfile`	Specifies the publish/subscribe transport configuration file. The default value depends on the transport type specified with the `-l` parameter: For `-l solace` the default is **`./solace.cfg`** For `-l tervela` the default is **`./client.config`** For `-l rabbitmq` the default is **`./rabbitmq.cfg`** For `-l kafka` the default is **`./kafka.cfg`** **Note:** No transport configuration file is required for native transport.
`-Z authpassword`	Specifies the password for SAS LASR Analytic Server authentication services.
`-z bufsize`	Specifies the socket buffer size.

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

Using the Modbus Connector and Adapter

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 19 Modbus Connector Object Types

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

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

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

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

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

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

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

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

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

When the subscriber connector receives an event from SAS Event Stream Processing, the values are pulled from the event and sent to Modbus using the libmodbus library. If you are running the simulator, the values update immediately.

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

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

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

Table 20 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 21 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 22 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 23 Optional Parameters for the Modbus Subscriber Connector

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

Using the Modbus Adapter

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.

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.

Subscriber usage:

dfesp_modbus_adapter -h *url* -k sub -S *slaveID* -s *server* <-C *configfilesection*> <-E *tokenlocation*> <-g *gdconfig*> <-l native | solace | tervela | rabbitmq | kafka> <-V> <-Z *transportconfigfile*>

Publisher usage:

dfesp_modbus_adapter -h *url* -k pub -S *slaveID* -s *server* <-b *blocksize*> <-C *configfilesection*> <-c *coils*> <-E *tokenlocation*> <-g *gdconfig*> <-H *holdingRegisters*> <-I *inputs*> <-i *interval*> <-l native | solace | tervela | rabbitmq | kafka> <-m *maxevents*> <-r *inputRegisters*> <-V> <-Z *transportconfigfile*>

Table 24 Modbus Adapter Parameters

Parameter	Description
`-b blocksize`	Specifies the event block size. The default is `1`.
`-C [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]`.
`-c coils`	Specifies the Modbus coils to read.
`-E tokenlocation`	Specifies the location of the file in the local file system that contains the OAuth token. This token is required for authentication by the publish/subscribe server.
`-g gdconfig`	Specifies the guaranteed delivery configuration file.
`-H holdingRegisters`	Specifies the Modbus holding registers to read.

`-h url`	Specifies the dfESP publish and subscribe standard URL in the form `dfESP://host:port/project/continuousquery/window`.
	Append the following for subscribers:`?snapshot=false`.
	Note: `?snapshot=true` is invalid.
	Append the following for subscribers if needed:
	`?collapse=true | false`
	`?rmretdel=true`

`-I inputs`	Specifies the Modbus inputs to read.
`-i interval`	Specifies the Modbus polling interval, in seconds. The default is `10`.

Parameter	Description
`-k sub | pub`	Specifies to publish or subscribe.
`-l native | solace | tervela | rabbitmq | kafka`	Specifies the publish/subscribe transport. The default is `native`.
`-m maxevents`	Specifies the maximum number of events to publish. The default is no maximum.
`-r inputRegisters`	Specifies the Modbus input registers to read.
`-S slaveID`	Specifies the slave ID of the Modbus device.
`-s server`	Specifies the Modbus server host name. The default port is `502`. To specify a different port, use the form `-s server:port`.
`-V`	Restarts the adapter if a fatal error is reported.
`-Z transportconfigfile`	Specifies the publish/subscribe transport configuration file. The default value depends on the transport type specified with the `-l` parameter: For `-l solace`, the default is **`./solace.cfg`**. For `-l tervela`, the default is **`./client.config`**. For `-l rabbitmq`, the default is **`./rabbitmq.cfg`**. For `-l kafka`, the default is **`./kafka.cfg`**. **Note:** No transport configuration file is required for native transport.

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.

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.
- SSL/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`**).

The MQTT connector is not supported on Windows systems.

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

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

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

Table 25 Required Parameters for Subscriber MQTT Connectors

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

Table 26 Required Parameters for Publisher MQTT Connectors

Parameter	Description
`type`	Specifies to publish. Must be "`pub`".
`mqtthost`	Specifies the MQTT server host name.

Parameter	Description
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, 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 27 Optional Parameters for Publisher MQTT Connectors

Parameter	Description
mqttuserid	Specifies the user name required to authenticate the connector's session with the MQTT server.
mqttpassword	Specifies the password associated with mqttuserid.
mqttport	Specifies the MQTT server port. Default is 1883.
mqttacceptretainedmsg	Sets to true to accept to receive retained message. The default is "false".
mqttcleansession	Set to true to instruct the MQTT Server to clean all messages and subscriptions on disconnect, false to instruct it to keep them. The default is "true".
mqttkeepaliveinterval	Specifies the number of seconds after which the broker should send a PING message to the client if no other messages have been exchanged in that time. The default is 10.
publishwithupsert	Builds events with opcode=Upsert instead of Insert.
transactional	When mqttmsgtype=CSV, sets the event block type to transactional. The default event block type is normal.
blocksize	When mqttmsgtype=CSV, specifies the number of events to include in a published event block. The default value is 1.
ignorecsvparseerrors	Specifies that when a field in an input CSV event cannot be parsed, the event is dropped, an error is logged, and publishing continues.
csvfielddelimiter	Specifies the character delimiter for field data in input CSV events. The default delimiter is the , character.
noautogenfield	Specifies that input events are missing the key field that is automatically generated by the Source window.

Parameter	Description
dateformat	Specifies the format of ESP_DATETIME and ESP_TIMESTAMP fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (ESP_DATETIME) or microseconds (ESP_TIMESTAMP) 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.
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.
mqttssl	Specifies to use SSL/TLS to connect to the MQTT broker. The default is "false". In order to use SSL/TLS, the SAS ESP 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.

Table 28 Optional Parameters for Subscriber MQTT Connectors

Parameter	Description
`mqttuserid`	Specifies the user name required to authenticate the connector's session with the MQTT server.
`mqttpassword`	Specifies the password associated with `mqttuserid`.
`mqttport`	Specifies the MQTT server port. The default is 1883.
`mqttretainmsg`	Sets to true to make the published message retained in the MQTT Server. The default is "`false`".
`mqttdonotcleansession`	Instructs the MQTT Server to keep all messages and subscriptions on disconnect, instead of keeping them. The default is "`false`".
`mqttkeepaliveinterval`	Specifies the number of seconds after which the broker should send a PING message to the client if no other messages have been exchanged in that time. The default is 10.
`mqttmsgmaxdeliveryattempts`	Specifies the number of times the connector tries to resend the message in case of failure. The default is 20.
`mqttmsgdelaydeliveryattempts`	Specifies the delay in milliseconds between delivery attempts specified with `mqttmsgmaxdeliveryattempts` . The default is 500.
`mqttmsgwaitbeforeretry`	Specifies the number of seconds to wait before retrying to send messages to the MQTT broker. This applies to publish messages with QoS > 0. The default is 20.
`mqttmaxinflightmsg`	Specifies the number of QoS 1 and 2 messages that can be simultaneously in flight. The default is 20.
`collapse`	Enables conversion of UPDATE_BLOCK events to make subscriber output publishable.
`rmretdel`	Specifies to remove all delete events from event blocks received by a subscriber that were introduced by a window retention policy
`dateformat`	Specifies the format of `ESP_DATETIME` and `ESP_TIMESTAMP` fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (`ESP_DATETIME`) or microseconds (`ESP_TIMESTAMP`) 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.
`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.
`mqttssl`	Specifies to use SSL/TLS to connect to the MQTT broker. The default is "`false`". In order to use SSL/TLS, the SAS ESP encryption overlay must be installed.

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

Using the MQTT Adapter

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

The MQTT adapter is not supported on Windows systems.

Note: Remove the reference to this connector from the file **`/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.excluded`** (Linux).

Subscriber usage:

dfesp_mqtt_adapter -c *mqttclientid* -h *url* -k sub -q 0 | 1 | 2 -z binary | csv | json -s *mqtthost* -t *mqtttopic* <-a *mqttkeepaliveinterval*> <-B *mqttmsgwaitbeforeretry*> <-C *configfilesection*> <-d *dateformat*> <-E *tokenlocation*> <-f *protofile*> <-G *mqttmaxinflightmsg*> <-g *gdconfig*> <-i *mqttsslcafile*> <-j trace | debug | info | warn | error | fatal | off> <-K *mqttsslkeyfile*> <-L> <-l native | solace | tervela | rabbitmq | kafka> <-M> <-m *protomsg*> <-n> <-o *mqttport*> <-P *mqttsslcapath*> <-p *mqttpassword*> <-r> <-S> <-T *mqttsslcertfile*> <-u *mqttuserid*> <-V> <-v *mqttmsgmaxdeliveryattempts*> <-W *mqttsslpassword*> <-w never | once | pereventblock> <-X> <-x *mqttmsgdelaydeliveryattempts*> <-y *logconfigfile*> <-Z *transportconfigfile*> <-3 *doubleprecision*>

Publisher usage:

dfesp_mqtt_adapter -c *mqttclientid* -h *url* -k pub -q 0 | 1 | 2 -z binary | csv | json -s *mqtthost* -t *mqtttopic* <-A> <-a *mqttkeepaliveinterval*> <-b *blocksize*> <-C *configfilesection*> <-D csvfielddelimiter> <-d *dateformat*> <-E *tokenlocation*> <-e> <-F *eventtype*> <-f *protofile*> <-g *gdconfig*> <-I> <-i *mqttsslcafile*> <-J> <-j trace | debug | info | warn | error | fatal | off> <-K *mqttsslkeyfile*> <-l native | solace | tervela | rabbitmq | kafka> <-m *protomsg*> <-n> <-O> <-o *mqttport*> <-P *mqttsslcapath*> <-p *mqttpassword*> <-Q> <-R> <-S> <-T *mqttsslcertfile*> <-u *mqttuserid*> <-V> <-W *mqttsslpassword*> <-X> <-Y *maxevents*> <-y *logconfigfile*> <-Z *transportconfigfile*>

Parameter	Description
`-A`	Specifies that input events are missing the key field that is automatically generated by the Source window.
`-a 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 seconds.
`-B mqttmsgwaitbeforeretry`	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.
`-b blocksize`	Specifies the event block size. The default is 1.
`-C 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]`.
`-c mqttclientid`	Specifies the string to use as the MQTT Client ID. If NULL, a random client ID will be generated. If NULL, `mqttcleansession` must be `true`. Must be unique among all clients connected to the MQTT server.
`-D csvfielddelimiter`	Specifies the character delimiter for field data in input CSV events. The default delimiter is the , character.
`-d dateformat`	Specifies the format of `ESP_DATETIME` and `ESP_TIMESTAMP` fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (`ESP_DATETIME`) or microseconds (`ESP_TIMESTAMP`) since epoch. The `dateformat` parameter accepts any time format that is supported by the UNIX `strftime` function.
`-E tokenlocation`	Specifies the location of the file in the local file system that contains the OAuth token that is required for authentication by the publish/subscribe server.

Parameter	Description
`-e`	Specifies that events are transactional.
`-F eventtype`	Specifies the event type to Insert into input CSV events (with comma). Valid values are `"normal"` and `"partialupdate"`.
`-f protofile`	Specifies the **`.proto`** file to be used for Google protocol buffer support.
`-G mqttmaxinflightmsg`	Specifies the number of QoS 1 and 2 messages that can be simultaneously in flight. The default is 20.
`-g gdconfig`	Specifies the guaranteed delivery configuration file.

`-h url`	Specifies the dfESP publish and subscribe standard URL in the form `dfESP://host:port/project/continuousquery/window`.
	Append the following for subscribers:`?snapshot=false`.
	Append the following for subscribers if needed:
	`?collapse=true | false`
	`?rmretdel=true`

`-I`	Specifies that when a field in an input CSV event cannot be parsed, the event is dropped, an error is logged, and publishing continues.
`-i 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.
`-J`	Enables receiving of retained messages. The default is “`false`”.
`-j trace | debug | info | warn | error | fatal | off`	Specifies the logging level. The default is “`warn`”.
`-K 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.
`-k sub | pub`	Specifies `sub` for subscriber or `pub` for publisher.
`-L`	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.
`-l native | solace | tervela | rabbitmq | kafka`	Specifies the publish/subscribe transport. The default is `native`.
`-M`	For CSV, specifies to send one message per event. The default is one message per transactional event block or else one message per event.
`-m protomsg`	Specifies the message itself in the **`.proto`** file that is specified by the `protofile` parameter.
`-n`	Instructs the MQTT Server to keep all messages and subscriptions on disconnect (instead of cleaning them). The default is “`false`”.

Parameter	Description
-O	Prepends an opcode and comma to input CSV events. The opcode is Insert unless -R is enabled.
-o mqttport	Specifies the MQTT server port. The default is 1883.
-P 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.
-p mqttpassword	Specifies the password associated with mqttuserid.
-Q	When *maxevents* is configured, quiesces the project after all events are injected into the Source window.
-q 0 | 1 | 2	Specifies the requested Quality of Service.
-R	Builds events with opcode=Upsert instead of opcode=Insert.
-r	Retains the published message in the MQTT Server. The default is “false”.
-S	Uses SSL/TLS to connect to the MQTT broker. The default is “false”. In order to use SSL/TLS, the SAS ESP encryption overlay must be installed.
-s mqtthost	Specifies the MQTT server host name.
-t mqtttopic	Specifies the MQTT topic.
-T 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.
-u mqttuserid	Specifies the user name required to authenticate the session with the MQTT server.
-V	Restarts the adapter if a fatal error is reported.
-v mqttmsgmaxdeliveryattempts	Specifies the number of times to try to resend the message in case of failure. The default is 20.
-W mqttsslpassword	If mqttssl=true and **keyfile** is encrypted, specifies the password for decryption.
-w never | once | pereventblock	When rmqtype=CSV, specifies when to prepend output CSV data with the window's serialized schema. The default value is “never”.
-X	Specifies that mqttpassword is encrypted.
-x mqttmsgdelaydeliveryattempts	Specifies the delay in milliseconds between delivery attempts specified in *mqttmsgmaxdeliveryattempts*. The default is 500.
-Y maxevents	Specifies the maximum number of events to publish.

Parameter	Description
`-y logconfigfile`	Specifies the logging configuration file. By default, there is none.
`-z binary | csv | json | opaquestring`	Specifies the message format. `opaquestring` is valid only for publishers. 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"`. For subscribers, the name of a string field in the subscribed window schema is also supported.
`-Z transportconfigfile`	Specifies the publish/subscribe transport configuration file. The default value depends on the transport type specified with the `-l` parameter: For `-l solace`, the default is **`./solace.cfg`**. For `-l tervela`, the default is **`./client.config`**. For `-l rabbitmq`, the default is **`./rabbitmq.cfg`**. For `-l kafka`, the default is **`./kafka.cfg`**. **Note:** No transport configuration file is required for native transport.
`-3 doubleprecision`	Specifies the number of fractional digits in the ASCII representation of a double. The default value is 6.

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

Using the Nurego Connector

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 29   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 30   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

Using the OPC-UA Connector

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

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

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

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

The Node ID format is a fixed notation that contains the node identifier type and the node identifier, in the form of `s_MyTemperature`. The first character is the identifier type and must be ‘s’ (String), ‘i’ (Integer), ‘g’ (GUID), or ‘b’ (Opaque). Following the underscore is the identifier string. If the type is Opaque, then the string must be base64–encoded.

The window schema field types must match the type of the corresponding OPC-UA node. The supported mappings are:

Table 31 Supported Mappings for a Publisher:

ESP type	OPC-UA type
ESP_INT32	INT32 or UINT32 or INT16 or UINT16 or BOOLEAN
ESP_INT64	INT64 or UINT64 or INT32 or UINT32 or INT16 or UINT16 or BOOLEAN
ESP_DOUBLE	DOUBLE or FLOAT or BOOLEAN
ESP_UTF8STR	STRING
ESP_DATETIME	DATETIME
ESP_TIMESTAMP	DATETIME
ESP_MONEY	Not supported

Table 32 Supported Mappings for a Subscriber:

ESP type	OPC-UA type
ESP_INT32	INT32 or UINT32
ESP_INT64	INT64 or UINT64
ESP_DOUBLE	DOUBLE
ESP_UTF8STR	STRING
ESP_DATETIME	DATETIME
ESP_TIMESTAMP	DATETIME
ESP_MONEY	Not supported

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

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

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

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

Table 33 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://`).

Table 34 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.

Parameter	Description
`opcuanodeids`	Specifies a comma-separated list of Node IDs to map to ESP window schema fields, in the form `<identifier type>_<identifier>`. The list size must be equal to the number of fields in the subscribed window schema - 1. Window field names are in Node ID form by default.
`publishinterval`	Specifies an interval in seconds when current values of all nodes in the Source window schema are published. The default is to publish when one or more values changes.
`transactional`	Sets the event block type to transactional. The default value is “`normal`”.
`blocksize`	Specifies the number of events to include in a published event block. The default value is `1`.
`configfilesection`	Specifies the name of the section in **`/opt/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.

Table 35 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.
`opcuanodeids`	Specifies a comma-separated list of Node IDs to map to ESP 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.

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]`.

Using the OPC-UA Adapter

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.

Subscriber usage:

dfesp_opcua_adapter -h *url* -k sub -s *opcuaendpoint* <-C *configfilesection*> <-E *tokenlocation*> <-g *gdconfig*> <-i *opcuanodeids*> <-j trace | debug | info | warn | error | fatal | off> <-l native | solace | tervela | rabbitmq | kafka> <-n *opcuanamespaceuri*> <-p *opcuapassword*> <-u *opcuausername*> <-V> <-y *logconfigfile*> <-Z *transportconfigfile*>

Publisher usage:

dfesp_opcua_adapter -h *url* -k pub -s *opcuaendpoint* <-b *blocksize*> <-C *configfilesection*> <-E *tokenlocation*> <-e > <-g *gdconfig*> <-i *opcuanodeids*> <-j trace | debug | info | warn | error | fatal | off> <-l native | solace | tervela | rabbitmq | kafka> <-m *maxevents*> <-n *opcuanamespaceuri*> <-p *opcuapassword*> <-Q> <-R> <-u *opcuausername*> <-V> <-v *publishinterval*> <-y *logconfigfile*> <-Z *transportconfigfile*>

Parameter	Description
`-b blocksize`	Specifies the event block size. The default is `1`.
`-C [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]`.
`-E tokenlocation`	Specifies the location of the file in the local file system that contains the OAuth token. This token is required for authentication by the publish/subscribe server. By default, there is none.
`-e`	Specifies that events are transactional.
`-g gdconfig`	Specifies a guaranteed delivery configuration file.

Parameter	Description
`-h url`	Specifies the dfESP publish and subscribe standard URL in the form `dfESP://host:port/project/continuousquery/window`. Append the following for subscribers: `?snapshot=false`. **Note:** `?snapshot=true` is invalid.
`-i opcuanodeids`	Specifies a comma–separated list of Node IDs to map to ESP window schema fields, in the form `<identifier type>_<identifier>`. The default assumes that window field names are in Node ID form.
`-j trace | debug | info | warn | error | fatal | off`	Specifies the logging level. The default is "`warn`".
`-k sub | pub`	Specifies `sub` for subscriber or `pub` for publisher.
`-l native| solace | tervela | rabbitmq | kafka`	Specifies the publish/subscribe transport. The default is `native`.
`-m maxevents`	Specifies the maximum number of events to publish.
`-n opcuanamespaceuri`	Specifies the OPC-UA server namespace URI. The default is the namespace at `index=0`.
`-p opcuapassword`	Specifies the OPC-UA password. By default, there is none.
`-Q`	When *maxevents* is configured, quiesces the project after all events are injected into the Source window.
`-R`	Builds events with `opcode=Upsert` instead of `opcode=Insert`.
`-s opcuaendpoint`	Specifies the OPC-UA server endpoint, without the `opc.tcp://` prefix.
`-u opcuausername`	Specifies OPC-UA user name. By default, there is none.
`-V`	Restarts the adapter after a fatal error.
`–v publishinterval`	Specifies the interval (in seconds) at which all current Node ID values are published into the Source window. By default, values are published when they change.
`-y logconfigfile`	Specifies the logging configuration file. By default, there is none.

Parameter	Description
`-Z transportconfigfile`	Specifies the publish/subscribe transport configuration file. The default value depends on the transport type specified with the `-l` parameter: For `-l solace`, the default is **`./solace.cfg`**. For `-l tervela`, the default is **`./client.config`**. For `-l rabbitmq`, the default is **`./rabbitmq.cfg`**. For `-l kafka`, the default 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: Advanced Topics*.

Using the PI Connector and Adapter

Using the PI Connector

The PI connector supports publish and subscribe operations for a PI Asset Framework (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 Asset Framework (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: Remove the reference to this connector from the file **`/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.excluded`** (Linux) or **`%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.excluded`** (Windows) before using it.

The PI connector also loads the esp_pi_afsdk52 DLL located in the **`$DFESP_HOME/bin/plugins`** directory. Make sure that the run-time environment also defines the path to this DLL.

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

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

Use the following schema values.

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

Schema Value	Description
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 TypeCode::UInt32
ESP_INT64	TypeCode::Int64 TypeCode::UInt64
ESP_UTF8STR	TypeCode::String
ESP_TIMESTAMP	TypeCode::DateTime

If an attribute has `TypeCode::Object`, the connected uses the type of the underlying PI point. Valid event stream processing data types to PI point mappings are as follows:

Event Stream Processor Data Type	PI Point
ESP_INT32	PIPointType::Int16 PIPointType::Int32
ESP_DOUBLE	PIPointType::Float16 PIPointType::Float32
ESP_TIMESTAMP	PIPointType::Timestamp
ESP_UTF8STR	PIPointType::String

Use the following parameters with PI connectors:

Table 36   Required Parameters for Subscriber PI Connectors

Parameter	Description
`type`	Specifies to subscribe. Must be "`sub`".
`afelement`	Specifies the AF element or element template name. Wildcards are supported.
`iselementtemplate`	Specifies whether the `afelement` parameter is an element template name. By default, the `afelement` parameter specifies an element name. Valid values are `TRUE` or `FALSE`.
`snapshot`	Specifies whether to send snapshot data. When `true`, the subscriber receives a collection of Insert events that are contained in the window at that point in time. The subscriber then receives a stream of events produced from the time of the snapshot onward. Those subsequent events can be Inserts, Updates, or Deletes.

Table 37   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 38   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.

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]`.

Table 39 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.

Using the PI Adapter

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.

Subscriber usage:

dfesp_pi_adapter -h *url* -k sub -s *afelement* <-a *afattribute* > <-C [*configfilesection*]> <-d *afdatabase* > <-E *tokenlocation* > <-g *gdconfig* > <-j trace | debug | info | warn | error | fatal> <-l native | solace | tervela | rabbitmq | kafka> <-p *pisystem* > <-r *afrootelement* > <-t> <-V> <-y *logconfigfile* > <-Z *transportconfigfile*>

Publisher usage:

dfesp_pi_adapter -h *url* -k pub -s *afelement* <-a *afattribute* > <-b *blocksize* > <-C [*configfilesection*]> <-c> <-d *afdatabase* > <-E *tokenlocation* > <-e> <-g *gdconfig* > <-j trace | debug | info | warn | error | fatal> <-l native | solace | tervela | rabbitmq | kafka> <-m *maxevents*> <-p *pisystem* > <-Q> <-R> <-r *afrootelement* > <-t> <-V> <-y *logconfigfile* > <-Z *transportconfigfile*>

Parameter	Description
`-a afattribute`	Specifies an attribute in *afelement* and ignore other attributes there.
`-b blocksize`	Specifies the block size. The default value is 1.
`-C [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]`.
`-c archivetimestamp`	Specifies that when connecting to the AF server, retrieve all archived values from the specified timestamp onwards.
`-d afdatabase`	Specifies the AF database
`-E tokenlocation`	Specifies the location of the file in the local file system that contains the OAuth token that is required for authentication by the publish/subscribe server.
`-e`	Specifies that events are transactional.
`-g gdconfig`	Specifies the guaranteed delivery configuration file.
`-h url`	Specifies the dfESP publish and subscribe 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 the following for subscribers if needed: `?collapse=true | false ?rmretdel=true | false`.
`-j 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.
`-k`	Specifies `sub` for subscriber use and `pub` for publisher use
`-l 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.

Parameter	Description
`-m maxevents`	Specifies the maximum number of events to publish.
`-p pisystem`	Specifies the PI system.
`-Q`	When *maxevents* is configured, quiesces the project after all events are injected into the Source window.
`-R`	Builds events with `opcode=Upsert` instead of `Insert`.
`-r afrootelement`	Specifies a root element in the AF hierarchy from which to search for *afelement*.
`-s afelement`	Specifies the AF element or element template name. Wildcards are supported.
`-t`	Specifies that *afelement* is the name of an element template.
`-V`	Restarts the adapter if a fatal error is reported.
`-y logconfigfile`	Specifies the log configuration file.
`-Z transportconfigfile`	Specifies the publish/subscribe transport configuration file. The default value depends on the transport type specified with the `-l` parameter: For `-l solace`, the default is **`./solace.cfg`**. For `-l tervela`, the default is **`./client.config`**. For `-l rabbitmq`, the default is **`./rabbitmq.cfg`**. For `-l kafka`, the default 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: Advanced Topics*.

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 40 Required Parameters for the Project Publish Connector

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

Parameter	Description
`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 41 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

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

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

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

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

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

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

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

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

Table 42 Required Parameters for Pylon Connectors

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

Table 43 Optional Parameters for Pylon Connectors

Parameter	Description
`cameraipaddress`	Specifies the camera IP address. The default value is the address of the first camera found on the local subnet.
`maxnumframes`	Specifies the maximum number of frames to publish. The default value is no maximum.
`maxframerate`	Specifies the maximum number of frames per second to publish. The default value is the rate at which frames are received from the camera.
`camerafeaturesfile`	Specifies a Pylon Features Stream (*.pfs) configuration file to load. The default is to use the current camera configuration unmodified.
`camerawidth`	Specifies the Area-Of-Interest width. The default value is the value in the current camera configuration.
`cameraheight`	Specifies the Area-Of-Interest height. The default value is the value in the current camera configuration.
`camerapixelformat`	Specifies the image pixel format. The default value is the format in the current camera configuration.
`camerapacketsize`	Specifies the Ethernet packet size. The default value is the value in the current camera configuration.
`transactional`	Sets the event block type to transactional. The default event block type is normal.

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`	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.

Using the Pylon Publisher Adapter

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

Usage:

dfesp_pylon_adapter -h *url* <-a *camerafeaturesfile*> <-C *configfilesection*> <-E *tokenlocation*> <-e> <-f *maxframerate*> <-g *gdconfig*> <-i *cameraipaddress*> <-j trace | debug | info | warn | error | fatal | off> <-l native | solace | tervela | rabbitmq | kafka> <-m *maxevents*> <-n *maxnumframes*> <-p *camerapixelformat*> <-Q> <-R> <-s *camerapacketsize*> <-t *cameraheight*> <-V> <-w *camerawidth*> <-y *logconfigfile*> <-Z *transportconfigfile*> <-X *cameraxoffset*> <-Y *camerayoffset*>

Parameter	Description
`-a camerafeaturesfile`	Specifies a Pylon Features Stream (*.pfs) configuration file to load. The default is to use the current camera configuration unmodified.
`-C [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]`.
`-E tokenlocation`	Specifies the location of the file in the local file system that contains the OAuth token that is required for authentication by the publish/subscribe server.

Parameter	Description
`-e`	Specifies that events are `transactional`.
`-f 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.
`-g gdconfig`	Specifies the guaranteed delivery configuration file.
`-h url`	Specifies the dfESP publish and subscribe standard URL in the form dfESP://host:port/project/continuousquery/window .
`-i cameraipaddress`	Specifies the camera IP address. The default value is the address of the first camera found on the local subnet.
`-j 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.
`-l 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.
`-m maxevents`	Specifies the maximum number of events to publish.
`-n maxnumframes`	Specifies the maximum number of frames to publish. The default value is no maximum.
`-p camerapixelformat`	Specifies the image pixel format. The default value is the format in the current camera configuration.
`-Q`	When *maxevents* is configured, quiesces the project after all events are injected into the Source window.
`-R`	Builds events with `opcode=Upsert` instead of `opcode=Insert`.
`-s camerapacketsize`	Specifies the Ethernet packet size. The default value is the value in the current camera configuration.
`-t cameraheight`	Specifies the Area-Of-Interest height. The default value is the value in the current camera configuration.
`-V`	Restarts the adapter if a fatal error is reported.
`-w camerawidth`	Specifies the Area-Of-Interest width. The default value is the value in the current camera configuration.
`-y logconfigfile`	Specifies the log configuration file.

Parameter	Description
`-Z transportconfigfile`	Specifies the publish/subscribe transport configuration file. The default value depends on the transport type specified with the `-l` parameter: For `-l solace`, the default is **`./solace.cfg`**. For `-l tervela`, the default is **`./client.config`**. For `-l rabbitmq`, the default is **`./rabbitmq.cfg`**. For `-l kafka`, the default is **`./kafka.cfg`**. **Note:** No transport configuration file is required for native transport.
`-X cameraxoffset`	Specifies the Area-Of-Interest horizontal offset. The default value is the value in the current camera configuration.
`-Y camerayoffset`	Specifies the Area-Of-Interest vertical offset. The default value is the value in the current camera configuration.

Using the Rabbit MQ Connector and Adapter

Using the Rabbit MQ Connector

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

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

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

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

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

A corresponding event stream processing publish/subscribe client plug-in library is available. This library enables a standard event stream processing publish/subscribe client application to exchange event blocks with an event stream processing server through a Rabbit MQ server. The exchange takes place through the Rabbit MQ server instead of through direct TCP connections. To enable this exchange, add a call to `C_dfESPpubsubSetPubsubLib()`.

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

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

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

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

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

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

You must install Rabbit MQ client run-time libraries on the platform that hosts the running instance of the connector. The connector uses the `rabbitmq-c` v0.5.2 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 SSL support enabled, make sure to include the path to those SSL libraries in your run-time environment.

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

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 Rabbit MQ exchange. The publisher connector enforces this by consuming the queue in exclusive mode when `buspersistence` is enabled.

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

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

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

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

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

Table 44 *Required Parameters for Subscriber Rabbit MQ Connectors*

Parameter	Description
`type`	Specifies to subscribe. Must be “`sub`”.
`rmquserid`	Specifies the user name required to authenticate the connector’s session with the Rabbit MQ server.
`rmqpassword`	Specifies the password associated with `rmquserid`.
`rmqhost`	Specifies the Rabbit MQ server host name.
`rmqport`	Specifies the Rabbit MQ server port.
`rmqexchange`	Specifies the Rabbit MQ exchange created by the connector, if nonexistent.
`rmqtopic`	Specifies the Rabbit MQ routing key to which messages are published.
`rmqtype`	Specifies `binary`, `CSV`, `JSON`, or the name of a string field in the subscribed window schema.
`urlhostport`	Specifies the *host:port* field in the metadata topic subscribed to on start-up to field metadata requests.
`numbufferedmsgs`	Specifies the maximum number of messages buffered by a standby subscriber connector. When exceeded, the oldest message is discarded. When the connector goes active, the buffer is flushed and buffered messages are sent to the fabric as required to maintain message ID sequence.

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. This parameter is invalid if buspersistence or hotfailover is enabled.

Table 45 Required Parameters for Publisher Rabbit MQ Connectors

Parameter	Description
type	Specifies to publish. Must be "pub".
rmquserid	Specifies the user name required to authenticate the connector's session with the Rabbit MQ server.
rmqpassword	Specifies the password associated with rmquserid.
rmqhost	Specifies the Rabbit MQ server host name.
rmqport	Specifies the Rabbit MQ server port.
rmqexchange	Specifies the Rabbit MQ exchange created by the connector, if nonexistent.
rmqtopic	Specifies the Rabbit MQ routing key to which messages are published.
rmqtype	Specifies binary, CSV, JSON, 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 46 Optional Parameters for Subscriber Rabbit MQ Connectors

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

Parameter	Description
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.
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.
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 Rabbit MQ vhost. The default is /.
csvmsgperevent	For CSV, specifies to send one message per event. The default is one message per transactional event block or else one message per event.
csvmsgpereventblock	For CSV, specifies to send one message per event block. The default is one message per transactional event block or else one message per event.
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.

Parameter	Description
rmqssl	Enables SSL encryption on the connection to the Rabbit MQ server.
rmqsslcacert	When rmqssl is enabled, specifies the full path of the SSL CA certificate .pem file.
rmqsslkey	When rmqssl is enabled, specifies the full path of the SSL key .pem file.
rmqsslcert	When rmqssl is enabled, specifies the full path of the SSL certificate .pem file.
doubleprecision	Specifies the number of fractional digits in the ASCII representation of a double. The default value is 6.

Table 47 Optional Parameters for Publisher Rabbit MQ Connectors

Parameter	Description
transactional	When rmqtype=CSV, sets the event block type to transactional. The default event block type is normal.
blocksize	When rmqtype=CSV, specifies the number of events to include in a published event block. The default value is 1.
dateformat	Specifies the format of ESP_DATETIME and ESP_TIMESTAMP fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (ESP_DATETIME) or microseconds (ESP_TIMESTAMP) 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.
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].
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.
`ackwindow`	Specifies the time period (in seconds) to leave messages that are received from Rabbit MQ unacknowledged. Applies only when `buspersistence = true`, when, by default, messages are never acknowledged. When configured, messages are acknowledged this number of seconds after having been received. Must be configured if `acktimer` is configured.
`acktimer`	Specifies the time interval (in seconds) for how often to check whether to send acknowledgments that are triggered by the `ackwindow` parameter. Must be configured if `ackwindow` is configured.
`publishwithupsert`	Builds events with `opcode=Upsert` instead of `Insert`.
`rmqpasswordencrypted`	Specifies that `rmqpassword` is encrypted.
`addcsvopcode`	Prepends an opcode and comma to input CSV events. The opcode is Insert unless `publishwithupsert` is enabled.
`addcsvflags`	Specifies the event type to insert into input CSV events (with a comma). Valid values are `normal` and `partialupdate`.
`rmqvhost`	Specifies the Rabbit MQ vhost. The default is `/`.
`useclientmsgid`	If the Source window has been restored from a persist to disk, ignores received binary event blocks that contain a message ID less than the greatest message ID in the restored window.
`rmqssl`	Enables SSL encryption on the connection to the Rabbit MQ server.
`rmqsslcacert`	When `rmqssl` is enabled, specifies the full path of the SSL CA certificate .pem file.
`rmqsslkey`	When `rmqssl` is enabled, specifies the full path of the SSL key .pem file.
`rmqsslcert`	When `rmqssl` is enabled, specifies the full path of the SSL certificate .pem file.
`maxevents`	Specifies the maximum number of events to publish.

Using the Rabbit MQ Adapter

The Rabbit MQ adapter supports publish and subscribe operations on a Rabbit MQ server. You must install the **`rabbitmq-c`** V0.5.2 client run-time libraries to use the adapter.

Note: Remove the reference to this connector from the file **`/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.excluded`** (Linux) or **`%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.excluded`** (Windows) before using it.

Subscriber usage:

dfesp_rmq_adapter -h *url* -k sub -n *numbufferedmsgs* -o *urlhostport* -p *rmqpassword* -r *rmqport* -s *rmqhost* -t *rmqtopic* -u *rmquserid* -v *rmqexchange* -z binary | csv | json <-C *configfilesection* > <-c *rmqcontenttype*> <-d *dateformat* > <-E *tokenlocation* > <-f *protofile* > <-g *gdconfig* > <-H *rmqheaders*> <-j trace | debug | info | warn | error | fatal | off> <-K *rmqsslkey*> <-L> <-l native | solace | tervela | rabbitmq | kafka> <-M> <-m *protomsg* > <-S> <-T *rmqsslcacert*> <-U *rmqsslcert*> <-V> <-w never | once | pereventblock> <-X> <-x > <-Y *rmqvhost*> <-y *logconfigfile* > <-Z *transportconfigfile*> <-3 *doubleprecision*>

Publisher usage:

dfesp_rmq_adapter -h *url* -k pub -o *urlhostport* -p *rmqpassword* -r *rmqport* -s *rmqhost* -t *rmqtopic* -u *rmquserid* -v *rmqexchange* -z binary | csv | json | opaquestring <-A> < -a *ackwindow* > <-b *blocksize* > <-C *configfilesection* > <-D *csvfielddelimiter* > <-d *dateformat* > <-E *tokenlocation* > <-e > <-F *eventtype*> <-f *protofile* > <-g *gdconfig* > <-I > <-i *acktimer* > <-j trace | debug | info | warn | error | fatal | off> <-K *rmqsslkey*> <-l native | solace | tervela | rabbitmq | kafka> <-m *protomsg* > <-N *maxevents*> <-O> <-Q> <-q *buspersistencequeue* > <-R> <-S> <-T *rmqsslcacert*> <-U *rmqsslcert*> <-V> <-X> < -x > <-Y *rmqvhost*> <-y *logconfigfile* > <-Z *transportconfigfile*>

Parameter	Description
`-A`	Specifies that input events are missing the key field that is autogenerated by the source window.
`-a ackwindow`	Specifies the time period to leave unacknowledged messages received from Rabbit MQ when `buspersistence` is enabled.
`-b blocksize`	Specifies the event block size. The default is 1.
`-C 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]`.

`-c 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"`

`-D csvfielddelimiter`	Specifies the character delimiter for field data in input CSV events. The default delimiter is the `,` character.

Parameter	Description
`-d dateformat`	Specifies the format of `ESP_DATETIME` and `ESP_TIMESTAMP` fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (`ESP_DATETIME`) or microseconds (`ESP_TIMESTAMP`) since epoch. The `dateformat` parameter accepts any time format that is supported by the UNIX `strftime` function.
`-E tokenlocation`	Specifies the location of the file in the local file system that contains the OAuth token that is required for authentication by the publish/subscribe server.
`-e`	Specifies that events are transactional.
`-F eventtype`	Specifies the event type to Insert into input CSV events (with comma). Valid values are `"normal"` and `"partialupdate"`.
`-f protofile`	Specifies the **`.proto`** file to be used for Google protocol buffer support.
`-g gdconfig`	Specifies the guaranteed delivery configuration file.
`-H rmqheaders`	Specifies a comma–separated list of key value optional headers in messages sent to RabbitMQ. The default value is no headers.
`-h url`	Specifies the dfESP publish and subscribe 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 the following for subscribers if needed: ■ `?collapse=true | false` ■ `?rmretdel=true | false`
`-I`	Specifies that when a field in an input CSV event cannot be parsed, the event is dropped, an error is logged, and publishing continues.
`-i acktimer`	Specifies the time interval for how often to check whether to send acknowledgments that are triggered by the `ackwindow` parameter.
`-j trace | debug | info | warn | error | fatal | off`	Specifies the logging level. The default is “`warn`”.

Parameter	Description
-K rmqsslkey	When rmqssl is enabled, specifies the full path of the SSL key .pem file.
-k	Specifies sub for subscriber or pub for publisher.
-L	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.
-l 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.
-M	For CSV, specifies to send one message per event. The default is one message per transactional event block or else one message per event.
-m protomsg	Specifies the message itself in the **.proto** file that is specified by the protofile parameter.
-N maxevents	Specifies the maximum number of events to publish.
-n numbufferedmsgs	Specifies the maximum number of messages buffered by a standby subscriber connector.
-O	Prepends an opcode and comma to input CSV events. The opcode is Insert unless -R is enabled.
-o urlhostport	Specifies the *host:port* field in the metadata topic to which the connector subscribes.
-p rmqpassword	Specifies the Rabbit MQ password.
-Q	When *maxevents* is configured, quiesces the project after all events are injected into the Source window.
-q buspersistencequeue	Specifies the queue name used by a persistent publisher.
-R	Builds events with opcode=Upsert instead of Insert.
-r rmqport	Specifies the Rabbit MQ port.
-S	Enables SSL encryption on the connection to the Rabbit MQ server.
-s rmqhost	Specifies the Rabbit MQ host.
-T rmqsslcacert	When rmqssl is enabled, specifies the full path of the SSL CA certificate .pem file.
-t rmqtopic	Specifies the Rabbit MQ routing key.

Parameter	Description
`-U rmqsslcert`	When `rmqssl` is enabled, specifies the full path of the SSL certificate .pem file.
`-u rmquserid`	Specifies the Rabbit MQ user name.
`-V`	Restarts the adapter if a fatal error is reported.
`-v rmqexchange`	Specifies the Rabbit MQ exchange.
`-w never | once | pereventblock`	When `rmqtype=CSV`, specifies when to prepend output CSV data with the window's serialized schema. The default value is "`never`".
`-X`	Specifies that `rmqpassword` is encrypted.
`-x`	Uses durable queues and persistent messages.
`-Y rmqvhost`	Specifies the Rabbit MQ vhost. The default is `/`.
`-y logconfigfile`	Specifies the logging configuration file. By default, there is none.
`-Z transportconfigfile`	Specifies the publish/subscribe transport configuration file. The default value depends on the transport type specified with the `-l` parameter: For `-l solace`, the default is **`./solace.cfg`**. For `-l tervela`, the default is **`./client.config`**. For `-l rabbitmq`, the default is **`./rabbitmq.cfg`**. For `-l kafka`, the default is **`./kafka.cfg`**. **Note:** No transport configuration file is required for native transport.
`-z binary | csv | json | 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"`. For subscribers, the name of a string field in the subscribed window schema is also supported.
`-3 doubleprecision`	Specifies the number of fractional digits in the ASCII representation of a double. The default value is 6.

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

Using the REST Subscriber Adapter

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

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

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

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

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

The response event fields are appended to the fields in the original subscribed event. Thus, you must be careful to ensure that the beginning fields in the schema of the Source window in `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.

Usage:

dfesp_rest_subscriber -r *resturl* -t *httpcontenttype* -u *url* <-c [*configfilesection*]> <-d *dateformat* > <-e *esprespurl* > <- g *gdconfigfile* > <-i> <-j *opaquejson* > <-l native | solace | tervela | rabbitmq | kafka> <-m *maxnumthreads* > <-O *tokenlocation* > <-o severe | warning | info> <-p *httpretries* > <-q *httprequestproperties*> <-s *httpstatuscodes* > <-V> <-v *httpretryinterval*> <-Z *transportconfigfile*>

Parameter	Description
`-c [configfilesection]`	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.
`-d dateformat`	Specifies the format of `ESP_DATETIME` and `ESP_TIMESTAMP` fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (`ESP_DATETIME`) or microseconds (`ESP_TIMESTAMP`) since epoch. The `dateformat` parameter accepts any time format that is supported by the UNIX `strftime` function.
`-e esprespurl`	Specifies the publish/subscribe standard URL to which responses should be published.
`-g gdconfigfile`	Specifies the guaranteed delivery configuration file for the client.

Parameter	Description
`-i`	Specifies to drop the subscribed event and continue when an HTTP connect fails. The default is to exit. **`javaadapters.config`** parameter name: `ignorefailedhttpconnects`
`-j opaquejson`	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.
`-l 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.
`-m maxnumthreads`	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.
`-O tokenlocation`	Specifies the location of the file in the local file system that contains the OAuth token that is required for authentication by the publish/subscribe server.
`-o severe | warning | info`	Specifies the application logging level.
`-p httpretries`	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.
`-r resturl`	Specifies the URL of the target REST service.
`-s httpstatuscodes`	Specifies a comma-separated list of acceptable status codes in response to the HTTP post. The default required response is 201.
`-t httpcontenttype`	Specifies the value of the Content-Type string used in the HTTP post.

`-u url`	Specifies the dfESP subscribe standard URL in the form "dfESP://*host*:*port*/*project*/*contquery*/*window*?snapshot=true|false".
	When `?snapshot=true`, the subscriber receives a collection of Insert events that are contained in the window at that point in time. The subscriber then receives a stream of events produced from the time of the snapshot onward. Those subsequent events can be Inserts, Updates, or Deletes.
	Append the following if needed:
	`?collapse=true | false`
	`?rmretdel=true | false`

`-V`	Restarts the adapter if a fatal error is reported. **`javaadapters.config`** parameter name: `restartonerror`
`-v httpretryinterval`	Specifies the number of seconds between HTTP post retries. The default is 0.

Parameter	Description
`-Z transportconfigfile`	Specifies the publish/subscribe transport configuration file. The default value depends on the transport type specified with the `-l` parameter: For `-l solace`, the default is **`./solace.cfg`**. For `-l tervela`, the default is **`./client.config`**. For `-l rabbitmq`, the default is **`./rabbitmq.cfg`**. For `-l kafka`, the default is **`./kafka.cfg`**. Note: No transport configuration file is required for native transport.
`-q httprequestproperties`	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.

Using the SAS Data Set Adapter

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

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

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

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

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

The user password must be passed in encrypted form unless `-w true` is configured. The encrypted version of the password can be generated by using OpenSSL, which must be installed on your system. When you install the SAS Event Stream Processing System Encryption and Authentication Overlay, you install the included OpenSSL executable.

Use the following command on the console to invoke OpenSSL to display your encrypted *password*:

echo "*password*" | **openssl** enc -e -aes-256-cbc -a -salt -pass pass:"SASespDSadapterUsedByUser=*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 -d *wssurl* -f *dsname* -h *url1* … *urlN* -k sub -u *username* -x *password* <-a *obstoanalyze* > <-C *configfilesection* > <-g *gdconfigfile* > <-L native | solace | tervela | rabbitmq | kafka> <-l *loglevel* > <-m *maxnumrows* > <-O *tokenlocation* > <-p *periodicity* > <-s *dsschema*> <-V> <-w> <-Z *transportconfigfile*> <-z *bufsize* >

Publisher usage:

dfesp_dataset_adapter -d *wssurl* -f *dsname* -h *url1* … *urlN* -k pub -u *username* -x *password* <-b *blocksize* > <-C *configfilesection* > <-e> <-g *gdconfigfile* > <-L native | solace | tervela | rabbitmq | kafka> <-l *loglevel* > <-O *tokenlocation* > <-Q> <-q *sqlquery* > <-r> <-V> <-W *maxevents*> <-w> <-Z *transportconfigfile*> <-z *bufsize* >

Parameter	Description
`-a obstoanalyze`	Specifies the number of observations to analyze to define the output SAS data set structure. Default value is 4096. If you had previously specified the `-s` option, this option is ignored.
`-b blocksize`	For a subscriber, specifies the number of event blocks to be appended to the data set at once. For a publisher, specifies the event block size to be published to the ESP Source window. Default value is 1.
`-C [configfilesection]`	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.
`-d wssurl`	Specifies the SAS Workspace Server connection URL in the form `"host:port"`.

Parameter	Description
-e	Specifies that event blocks are transactional. By default, event blocks are normal. **javaadapters.config** parameter name: transactional
-f dsname	Specifies the subscriber output file or publisher input file. The full path, including the extension, must be specified for dsname. Here is an example: -f "D:/sas/SASBI/Lev1/AppData/filename.sas7bdat"
-g gdconfig	Specifies the guaranteed delivery configuration file.
-h url	Specifies one or more standard dfESP URLs in the following form: dfESP://host:port/project/continuousquery/window Append the following to the URL 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 the following for subscribers if needed: ?collapse=true | false ?rmretdel=true | false
-k pub | sub	Specifies a publisher or a subscriber.
-L native | solace | tervela | rabbitmq | kafka	Specifies the transport type. When you specify solace, tervela, rabbitmq, or kafka transports instead of the default native transport, use the required client configuration files specified in "Using Alternative Transport Libraries for Java Clients" in *SAS Event Stream Processing: Publish/Subscribe API*.
-l loglevel	Specifies the Java standard logging level. Use one of the following values: SEVERE | WARNING | INFO. Default value is INFO.
-m maxnumrows	Specifies the output file periodicity in number of rows. Invalid if data is not insert-only.
-O tokenlocation	Specifies the location of the file in the local file system that contains the OAuth token that is required for authentication by the publish/subscribe server
-p periodicity	Specifies the output file periodicity in seconds. Invalid if data is not insert-only.
-Q	Quiesces the project after all events are injected into the Source window. **javaadapters.config** parameter name: quiesceproject
-q sqlquery	Specifies an SQL query to get SAS data set content. For example: "select * from dataset". **Note:** Use the *dataset* specified for the -f option without an extension.
-r	Builds events with opcode=Upsert. By default, events are built with opcode=Insert. **javaadapters.config** parameter name: publishwithupsert

Parameter	Description

`-s dsschema`	Specifies the output data set schema in the following form:
	`"rowname1:sastype1:sasformat1:label1,...,rownameN:sastypeN:sasformatN:labelN"`
	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.

`-u username`	Specifies the SAS Workspace Server user name.
`-V`	Restarts the adapter if a fatal error is reported. **`javaadapters.config`** parameter name: `restartonerror`
`-W maxevents`	Specifies the maximum number of events to publish.
`-w`	Specifies that the configured SAS Workspace Server password is unencrypted. By default, the password is encrypted. **`javaadapters.config`** parameter name: `unencryptedpassword`
`-x password`	Specifies the SAS Workspace Server password. When `-w true` is configured, this password must be encrypted.
`-Z transportconfigfile`	Specifies the publish/subscribe transport configuration file. The default value depends on the transport type specified with the `-L` parameter: For `-L solace`, the default is **`./solace.cfg`**. For `-L tervela`, the default is **`./client.config`**. For `-L rabbitmq`, the default is **`./rabbitmq.cfg`**. For `-L kafka`, the default is **`./kafka.cfg`**. **Note:** No transport configuration file is required for native transport.
`-z bufsize`	Specifies the socket buffer size.

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

Using the SMTP Connector and Adapter

Using the SMTP Subscribe Connector

You can use the Simple Mail Transfer Protocol (SMTP) subscribe connector to e-mail 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 e-mail sender and receiver addresses are required information for the connector. The e-mail 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 e-mail body contains data for one or more events encoded in CSV format.

The parameters for the SMTP connector are as follows:

Table 48 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 e-mail address to be used in the “from” field of the e-mail.
`destaddress`	Specifies the e-mail address to which to send the e-mail message.
`snapshot`	Specifies whether to send snapshot data. When `true`, the subscriber receives a collection of Insert events that are contained in the window at that point in time. The subscriber then receives a stream of events produced from the time of the snapshot onward. Those subsequent events can be Inserts, Updates, or Deletes.

Table 49 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 e-mail 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.

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]`.
`dateformat`	Specifies the format of `ESP_DATETIME` and `ESP_TIMESTAMP` fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (`ESP_DATETIME`) or microseconds (`ESP_TIMESTAMP`) 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.

Using the SMTP Subscriber Adapter

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

Subscriber use:

dfesp_smtp_adapter -d *destaddress* -h *url* -m *smtpserver* -u *sourceaddress* <-a *dateformat*> <-C [*configfilesection*]> <-E *tokenlocation* > <-g *gdconfig* > <-j trace | debug | info | warn | error | fatal | off> <-l native | solace | tervela | rabbitmq | kafka> <-p> <-V> <-y *logconfigfile* ><-Z *transportconfigfile*> <-3 *doubleprecision*>

Parameter	Description
`-a dateformat`	Specifies the format of `ESP_DATETIME` and `ESP_TIMESTAMP` fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (`ESP_DATETIME`) or microseconds (`ESP_TIMESTAMP`) since epoch. The `dateformat` parameter accepts any time format that is supported by the UNIX `strftime` function.
`-C [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]`.
`-d destaddress`	Specifies the destination email address
`-E tokenlocation`	Specifies the location of the file in the local file system that contains the OAuth token that is required for authentication by the publish/subscribe server
`-g gdconfig`	Specifies the guaranteed delivery configuration file.

Parameter	Description
`-h url`	Specifies the dfESP subscribe standard URL in the form "`dfESP://host:port/project/continuousquery/window?snapshot=true |false`. When `?snapshot=true`, the subscriber receives a collection of Insert events that are contained in the window at that point in time. The subscriber then receives a stream of events produced from the time of the snapshot onward. Those subsequent events can be Inserts, Updates, or Deletes. You can append the following if needed: ■ `?collapse=true | false` ■ `?rmretdel=true | false`
`-j 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.
`-l 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.
`-m smtpserver`	Specifies the SMTP server name.
`-p`	Specifies that each email contains a single event instead of a complete event block containing one or more events.
`-u sourceaddress`	Specifies the source email address.
`-V`	Restarts the adapter if a fatal error is reported.
`-y logconfigfile`	Specifies the log configuration file.
`-Z transportconfigfile`	Specifies the publish/subscribe transport configuration file. The default value depends on the transport type specified with the `-l` parameter: For `-l solace`, the default is **`./solace.cfg`**. For `-l tervela`, the default is **`./client.config`**. For `-l rabbitmq`, the default is **`./rabbitmq.cfg`**. For `-l kafka`, the default is **`./kafka.cfg`**. **Note:** No transport configuration file is required for native transport.
`-3 doubleprecision`	Specifies the number of fractional digits in the ASCII representation of a double. The default value is 6.

Using the Sniffer Connector and Adapter

Using the Sniffer Publish Connector

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

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.
- SSL/TLS Client Hello packets sent to port 443 over TCP or UDP.
- Traffic on other ports is supported only to the extent that the IP source and destination address, TCP or UDP source and destination port, and the verbatim payload are captured. Other packet fields are not available for capture.

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

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

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

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

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

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

The `packetfields` configuration parameter uses SAS Event Stream Processing schema-like syntax, and must match the Source window schema. There are three exceptions:

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

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

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

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

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

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

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

Table 50 *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.

Parameter	Description
`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_RequestURI:string`. An example for SSL/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`

Table 51 *Optional Parameters for the Sniffer Connector*

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

Parameter	Description
`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

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.

Note: Remove the reference to this connector from the file **`/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.excluded`** (Linux) or **`%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.excluded`** (Windows) before using it.

Publisher use:

dfesp_sniffer_adapter -f *packetfields* -h *url* -i *interface* -p *protocol* <-b *blocksize* > <-C *configfilesection* > <-d *vendorID* > <-E *tokenlocation* > <-e > <-F *pcapfilter* > <-g *gdconfig* > <-I> <-j trace | debug | info | warn | error | fatal | off> <-l native | solace | tervela | rabbitmq | kafka> <-m *maxevents*> <-n *indexfieldname* > <-o *httpports* > <-Q> <-R> <-t> <-V> <-v *vendorType* > <-y *logconfigfile* > <-Z *transportconfigfile*>

Parameter	Description
`-b blocksize`	Specifies the event block size. The default value is 1.
`-C 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]`.
`-d 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.

Parameter	Description
`-E tokenlocation`	Specifies the location of the file in the local file system that contains the OAuth token that is required for authentication by the publish/subscribe server.
`-e`	Specifies that events are transactional.
`-F pcapfilter`	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.
`-f packetfields`	Specifies a list of fields to be extracted from captured packets. You must specify `"payload:string"` when the protocol type is not 80 (`http`) or 1813 (`radius`).
`-g gdconfig`	Specifies the name of the guaranteed delivery configuration file.
`-h url`	Specifies the dfESP publish standard URL in the form "`dfESP://host:port/project/contquery/window`".
`-I`	Specifies whether to ignore packets that have no payload, as calculated by subtracting the TCP or UDP header size from the packet size.
`-i interface`	Specifies the name of the network interface on the local machine from which to capture packets.
`-j trace | debug | info | warn | error | fatal | off`	Specifies the logging level. The default is “`warn`”.
`-l native | solace | tervela | rabbitmq | kafka`	Specifies the publish/subscribe transport. When 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.
`-m maxevents`	Specifies the maximum number of events to publish.
`-n indexfieldname`	Specifies the name to use instead of `index` for the `index:int65` field in the Source window schema.
`-o 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.
`-p protocol`	Specifies the port number associated with the protocol type of packets to be captured.
`-Q`	When *maxevents* is configured, quiesces the project after all events are injected into the Source window.
`-R`	Builds events with `opcode=Upsert` instead of `Insert`.
`-t`	Appends an `ESP_TIMESTAMP` field to each published event.
`-V`	Restarts the adapter if a fatal error is reported.

Parameter	Description
`-v vendorType`	Specifies the vendor type filed to match when capturing the Attribute-Specific field in a Vendor-Specific attribute in a Radius Accounting-Request packet.
`-y logconfigfile`	Specifies the name of the logging configuration file. By default, there is none.
`-Z transportconfigfile`	Specifies the publish/subscribe transport configuration file. The default value depends on the transport type specified with the `-l` parameter: For `-l solace`, the default is **`./solace.cfg`**. For `-l tervela`, the default is **`./client.config`**. For `-l rabbitmq`, the default is **`./rabbitmq.cfg`**. For `-l kafka`, the default 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: Advanced Topics*.

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: Remove the reference to this connector from the file **`/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.excluded`** (Linux) or **`%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.excluded`** (Windows) before using it.

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 -a -salt -pass pass:"SASespSOLconnectorUsedByUser=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 52 Required Parameters for Subscriber Solace Systems Connectors

Parameter	Description
`type`	Specifies to subscribe. Must be “`sub`”.
`soluserid`	Specifies the user name required to authenticate the connector’s session with the appliance.
`solpassword`	Specifies the password associated with `soluserid`.
`solhostport`	Specifies the appliance to connect to, in the form `host:port`.
`solvpn`	Specifies the appliance message VPN to assign the client to which the session connects.
`soltopic`	Specifies the Solace destination topic to which to publish.
`urlhostport`	Specifies the `host:port` field in the metadata topic subscribed to on start-up to field metadata requests.
`numbufferedmsgs`	Specifies the maximum number of messages buffered by a standby subscriber connector. If exceeded, the oldest message is discarded. If the connector goes active the buffer is flushed, and buffered messages are sent to the fabric as required to maintain message ID sequence.
`snapshot`	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 53 Required Parameters for Publisher Solace Systems Connectors

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

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 topic to which to subscribe.
urlhostport	Specifies the host:port field in the metadata topic subscribed to on start-up to field metadata requests.

Table 54 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 ESP_DATETIME and ESP_TIMESTAMP fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (ESP_DATETIME) or microseconds (ESP_TIMESTAMP) since epoch. The dateformat parameter accepts any time format that is supported by the UNIX strftime function.

Parameter	Description
solpasswordencrypted	Specifies that solpassword is encrypted.

Table 55 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 ESP_DATETIME and ESP_TIMESTAMP fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (ESP_DATETIME) or microseconds (ESP_TIMESTAMP) since epoch. The dateformat parameter accepts any time format that is supported by the UNIX strftime function.
solpasswordencrypted	Specifies that solpassword is encrypted.
getmsgfromdestattr	Extracts the payload from the destination attribute instead of the message body.
transactional	When getmsgfromdestattr is enabled, sets the event block type to transactional. The default event block type is normal.
blocksize	When getmsgfromdestattr is enabled, specifies the number of events to include in a published event block. The default value is 1.
maxevents	Specifies the maximum number of events to publish.

Using the Solace Systems Adapter

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.

Subscriber usage:

dfesp_sol_adapter -h *url* -k sub -n *numbufferedmsgs* -o *urlhostport* -p *solpassword* -t *soltopic* -u *soluserid* -v *solvpn* <-b > <-C [*configfilesection*]> <-d *dateformat* > <-E *tokenlocation* > <-f *protofile* > <-g *gdconfig* > <-j trace | debug | info | warn | error | fatal | off> <-l native | solace | tervela | rabbitmq | kafka> <-m *protomsg* > <-V> <-X> <-y *logconfigfile* > <-Z *transportconfigfile*>

Publisher usage:

dfesp_sol_adapter -h *url* -k pub -o *urlhostport* -p *solpassword* -t *soltopic* -u *soluserid* -v *solvpn* <-B *blocksize*> <-b > <-C [*configfilesection*]> <-D> <-d *dateformat* > <-E *tokenlocation* > <-e> <-f *protofile* > <-g *gdconfig* > <-j trace | debug | info | warn | error | fatal | off> <-l native | solace | tervela | rabbitmq | kafka> <-m *protomsg* > <-Q> <-q *buspersistencequeue* > <-R> <-V> <-X> <-Y maxevents> <-y *logconfigfile* > <-Z *transportconfigfile*>

Parameter	Description
`-B blocksize`	When `getmsgfromdestattr` is enabled, specifies the number of events to include in a published event block. The default value is 1.
`-b`	Uses Solace Guaranteed Messaging. By default, Solace Direct Messaging is used.
`-C [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]`.
`-D`	Extracts the payload from the destination attribute instead of the message body.
`-d dateformat`	Specifies the format of `ESP_DATETIME` and `ESP_TIMESTAMP` fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (`ESP_DATETIME`) or microseconds (`ESP_TIMESTAMP`) since epoch. The `dateformat` parameter accepts any time format that is supported by the UNIX `strftime` function.
`-E tokenlocation`	Specifies the location of the file in the local file system that contains the OAuth token that is required for authentication by the publish/subscribe server
`-e`	When `getmsgfromdestattr` is enabled, sets the event block type to `transactional`. The default event block type is `normal`.
`-f protofile`	Specifies the **`.proto`** file that contains the message used for Google Protocol buffer support.
`-g gdconfig`	Specifies the guaranteed delivery configuration file.

Parameter	Description
-h url	Specifies the dfESP publish and subscribe 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 the following for subscribers if needed: ■ ?collapse= true | false ■ ?rmretdel=true | false
-j 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.
-k	Specifies sub for subscriber use and pub for publisher use
-l 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.
-m protomsg	Specifies the message itself in the **.proto** file that is specified by the protofile parameter.
-n numbufferedmsgs	Specifies the maximum number of messages buffered by a standby subscriber connector.
-o urlhostport	Specifies the *host*:*port* field in the metadata topic subscribed to by the connector.
-p solpassword	Specifies the Solace password.
-Q	When *maxevents* is configured, quiesces the project after all events are injected into the Source window.
-q buspersistencequeue	Specifies the queue name used by Solace Guaranteed Messaging publisher.
-R	Builds events with opcode=Upsert instead of opcode=Insert.
-s solhostport	Specifies the Solace *host*:*port*.
-t soltopic	Specifies the Solace topic.
-u soluserid	Specifies the Solace user name.
-V	Restarts the adapter if a fatal error is reported.
-v solvpn	Specifies the Solace VPN name.
-X	Specifies that solpassword is encrypted.

Parameter	Description
`-Y maxevents`	Specifies the maximum number of events to publish.
`-y logconfigfile`	Specifies the log configuration file.
`-Z transportconfigfile`	Specifies the publish/subscribe transport configuration file. The default value depends on the transport type specified with the `-l` parameter: For `-l solace`, the default is **`./solace.cfg`**. For `-l tervela`, the default is **`./client.config`**. For `-l rabbitmq`, the default is **`./rabbitmq.cfg`**. For `-l kafka`, the default 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: Advanced Topics*.

Using the Teradata Connector and Adapter

Using the Teradata Connector

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

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

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

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

Operator	Description
`stream`	Works like a standard 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).

Operator	Description
`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 "tdatauserpwd" | openssl enc -e -aes-256-cbc -a -salt
-pass pass:"SASespTDATAconnectorUsedByUser=tdatausername"
```

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

Table 56 Required Parameters for Teradata Connectors

Parameter	Description
`type`	Specifies to subscribe. Must be “`sub`”.
`desttablename`	Specifies the target table name.
`tdatausername`	Specifies the user name for the user account on the target Teradata server.
`tdatauserpwd`	Specifies the user password for the user account on the target Teradata server.
`tdatatdpid`	Specifies the target Teradata server name
`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 57 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 (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

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: Remove the reference to this connector from the file **`/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.excluded`** (Linux) or **`%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.excluded`** (Windows) before using it.

Note: A separate database adapter uses generic ODBC drivers for the Teradata platform. For more information, see "Using the Database Adapter".

Usage:

dfesp_tdata_adapter -a *maxsessions* -d stream | update | load -h *url* -i true | false -n *minsessions* -s *servername* -t *tablename* -u *username* -x *userpwd* <-b *batchperiod* > <-C [*configfilesection*]> <-c *connectstring* > <-E *tokenlocation* > <-g *gdconfig* > <-j trace | debug | info | warn | error | fatal | off> <-l native | solace | tervela | rabbitmq | kafka> <-q *stage1table* > <-r *stage2table* > <-V> <-X> <-y *logconfigfile* > <-Z *transportconfigfile*> <-z *tracelevel* >

Parameter	Description
-a maxsessions	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.
-b batchperiod	Specifies the batch period in seconds. This parameter is required when -d specifies the update operator.
-C [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].
-c connectstring	Specifies the connect string to be used by the ODBC driver to access the staging and target tables on the Teradata server. See "Using the Database Connector" for Teradata connectstring requirements. This parameter is required when -d load is specified.
-d stream | update | load	Specifies the Teradata operator. For more information about these operators, see the documentation provided with the Teradata Tools and Utilities.
-E tokenlocation	Specifies the location of the file in the local file system that contains the OAuth token that is required for authentication by the publish/subscribe server
-g gdconfig	Specifies the guaranteed delivery configuration file.
-h url	Specifies the dfESP subscribe standard URL in the form dfESP://host:port/project/continuousquery/window?snapshot=true |false. When ?snapshot=true, the subscriber receives a collection of Insert events that are contained in the window at that point in time. The subscriber then receives a stream of events produced from the time of the snapshot onward. Those subsequent events can be Inserts, Updates, or Deletes. Append ?collapse=true | false if needed. Append ?rmretdel=true | false if needed.
-i	Specifies that the subscribed window contains insert-only data.
-j trace | debug | info | warn | error | fatal | off	Sets the logging level for the adapter. Use the same range of logging levels that you can set in the C_dfESPpubsubInit() publish/subscribe API call or in the engine initialize() call.
-l 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.
-n minsessions	Specifies the minimum number of sessions on the Teradata server.
-q stage1table	Specifies the name of the first staging table on the Teradata server. This parameter is required when -d specifies the load operator.

Parameter	Description
`-r stage2table`	Specifies the name of the second staging table on the Teradata server. This parameter is required when `-d` specifies the load operator.
`-s servername`	Specifies the name of the target Teradata server.
`-t tablename`	Specifies the name of the target table on the Teradata server.
`-u username`	Specifies the Teradata user name.
`-V`	Restarts the adapter if a fatal error is reported.
`-X`	Specifies that `tdatauserpwd` is encrypted.
`-x userpwd`	Specifies the Teradata user password.
`-y logconfigfile`	Specifies the log configuration file.
`-Z transportconfigfile`	Specifies the publish/subscribe transport configuration file. The default value depends on the transport type specified with the `-l` parameter: For `-l solace`, the default is **`./solace.cfg`**. For `-l tervela`, the default is **`./client.config`**. For `-l rabbitmq`, the default is **`./rabbitmq.cfg`**. For `-l kafka`, the default is **`./kafka.cfg`**. **Note:** No transport configuration file is required for native transport.
`-z tracelevel`	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).

Using the Teradata Listener Connector and Adapter

Using the Teradata Listener Connector

The Teradata Listener connector sends data from SAS Event Stream Processing to the Teradata Listener application.

Note: Teradata Listener version 2.02 and higher versions are not supported.

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

Blocks of events are sent to the Teradata Listener Ingest service using a REST API. The Teradata Listener connector supports Listener Ingest REST API (version 1).

After data is delivered to the Listener Ingest service, Listener queues the data and sends it to all targets that subscribe to that Listener source stream. Targets can be one or more Teradata, Aster, or Hadoop systems, as well as other applications that read from the same Listener source using the Listener Stream service.

The Teradata Listener connector collects events from SAS Event Stream Processing and transforms them into JSON or plaintext format, as specified by the `contentType` parameter. The connection to the Listener Ingest service is defined by the `ingestUrl` and `SSLCert` parameters. The Listener source (data stream) where the data will be delivered is identified by the `sourceKey` parameter.

The Teradata Listener connector is a subscribe-only. To send data from Teradata Listener to SAS Event Stream Processing, use the WebSocket connector with `contentType=JSON`.

Note: Because the Teradata Listener connector uses SSL to establish an encrypted connection to the Teradata Listener Ingest service, the run-time environment must define the path to your SSL libraries (for example, specifying LD_LIBRARY_PATH on Linux platforms). For more information about SSL libraries that are used in SAS Event Stream Processing, see *SAS Event Stream Processing: Security*.

Table 58 *Required Parameters for Teradata Listener Connectors*

Parameter	Description
`type`	Specifies to subscribe. Must be “`sub`”.
`ingestUrl`	Specifies the URL for the Listener Ingest REST API (version 1).
`SSLCert`	Specifies the path to a file that contains SSL certificates securely connect to the Listener Ingest service. Listener uses TLS 1.2.
`sourceKey`	Specifies the Listener source secret key that identifies the Listener source feed to which SAS Event Stream Processing sends data. Contact your Teradata Listener administrator to obtain the source secret key.
`snapshot`	Specifies whether to send snapshot data. When `true`, the subscriber receives a collection of Insert events that are contained in the window at that point in time. The subscriber then receives a stream of events produced from the time of the snapshot onward. Those subsequent events can be Inserts, Updates, or Deletes.

Table 59 *Optional Parameters for Teradata Listener Connectors*

Parameter	Description
`ingestBlocksize`	Specifies the maximum number of data rows to send in one Listener Ingest message. Matching the connector block size to the Source window block size is recommended. The default block size is 256.
`contentType`	Specifies the format of the data sent from SAS Event Stream Processing to Listener, either `JSON` or `plaintext` (comma-delimited). The default is “`JSON`”.
`ingestDelim`	Specifies the character that delimits data rows in a multi-row message from SAS Event Stream Processing to the Listener Ingest REST API. The delimiter must not be a JSON punctuation character. The default is a tilde (~).
`-3 doubleprecision`	Specifies the number of fractional digits in the ASCII representation of a double. The default value is 6.

Using the Teradata Listener Adapter

The Teradata Listener subscriber adapter provides the same functionality as the Teradata Listener connector in addition to several optional, adapter-only parameters.

Usage:

dfesp_tdlistener_adapter -h *url* -i *ingestUrl* -k *sourceKey* -s *SSLCert* <-b *ingestBlocksize*> <-C *configfilesection*> <-c JSON | plaintext> <-d *ingestDelim*> <-E *tokenlocation*> <-g *gdconfig*> <-j trace | debug | info | warn | error | fatal | off> <-l native | solace | tervela | rabbitmq | kafka> <-V> <-y *logconfigfile*> <-Z *transportconfigfile*> <-3 *doubleprecision*>

Parameter	Description
`-b ingestBlocksize`	Specifies the maximum number of data rows to send in one Listener Ingest message. The default is `256`.
`-C [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]`.
`-c JSON | plaintext`	Specifies the format of the data sent from SAS Event Stream Processing to the Listener. The default is “`JSON`”.
`-d ingestDelim`	Specifies the character that delimits data rows in a multi-row message from SAS Event Stream Processing to the Listener Ingest REST API. The delimiter must not be a JSON punctuation character. The default is tilde (~).
`-E tokenlocation`	Specifies the location of the file in the local file system that contains the OAuth token that is required for authentication by the publish/subscribe server.
`-g gdconfig`	Specifies the guaranteed delivery configuration file.
`-h url`	Specifies the dfESP publish and subscribe 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 the following if needed: `?rmretdel=true | false`
`-i ingestUrl`	Specifies the URL for the Listener Ingest REST API (version 1).
`-j 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.
`-k sourceKey`	Specifies the Listener source secret key that identifies the Listener source feed where SAS Event Stream Processing sends data. Contact your Teradata Listener administrator to obtain the source key.

Parameter	Description
`-l 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.
`-n numbufferedmsgs`	Specifies the maximum number of messages buffered by a standby subscriber connector.
`-s SSLCert`	Specifies the path to a file that contains SSL certificates for securely connecting to the Listener Ingest service. Listener uses TLS 1.2.
`-V`	Restarts the adapter if a fatal error is reported.
`-y logconfigfile`	Specifies the log configuration file.
`-Z transportconfigfile`	Specifies the publish/subscribe transport configuration file. The default value depends on the transport type specified with the `-l` parameter: For `-l solace`, the default is **`./solace.cfg`**. For `-l tervela`, the default is **`./client.config`**. For `-l rabbitmq`, the default is **`./rabbitmq.cfg`**. For `-l kafka`, the default is **`./kafka.cfg`**. **Note:** No transport configuration file is required for native transport.
`-3 doubleprecision`	Specifies the number of fractional digits in the ASCII representation of a double. The default value is 6.

Using the Tervela Connector and Adapter

Using the Tervela Data Fabric Connector

The Tervela Data Fabric connector communicates with a software or hardware-based Tervela Data Fabric for publish and subscribe operations.

A Tervela subscriber connector receives events blocks and publishes to the following Tervela topic: "SAS.ENGINES.*enginename*.*projectname*.*queryname*.*windowname*.OUT."

A Tervela publisher connector reads event blocks from the following Tervela topic: "SAS.ENGINES.*enginename*.*projectname*.*queryname*.*windowname*.IN" and injects them into Source windows.

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

You must install the Tervela run-time libraries on the platform that hosts the running instance of the connector. The run-time environment must define the path to those libraries (specify **`LD_LIBRARY_PATH`** on Linux platforms, for example).

Note: Remove the reference to this connector from the file **`/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.excluded`** (Linux) or **`%ProgramData%\SAS`**

\Viya\SASEventStreamProcessingEngine\default\connectors.excluded (Windows) before using it.

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 -a -salt -pass pass:"SASespTVAconnectorUsedByUser=tvauserid"
```

Then copy the encrypted password into your `tvapassword` parameter and enable the `tvapasswordencrypted` parameter.

Use the following parameters with Tervela connectors.

Table 60 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 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 61 Required Parameters for Publisher Tervela Connectors

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

Parameter	Description
`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 62 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.

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].
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 ESP_DATETIME and ESP_TIMESTAMP fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (ESP_DATETIME) or microseconds (ESP_TIMESTAMP) since epoch. The dateformat parameter accepts any time format that is supported by the UNIX strftime function.
tvapasswordencrypted	Specifies that tvapassword is encrypted.

Table 63 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.

Parameter	Description
`publishwithupsert`	Specifies to build events with `opcode = Upsert` instead of `opcode = Insert`.
`dateformat`	Specifies the format of `ESP_DATETIME` and `ESP_TIMESTAMP` fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (`ESP_DATETIME`) or microseconds (`ESP_TIMESTAMP`) 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

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.

Subscriber usage:

dfesp_tva_adapter -b *numbufferedmsgs* -c *tvaclientname* -f *tvatopic* -h *url* -k sub -m *tvamaxoutstand* -o *urlhostport* -p *tvapassword* -t *tvaprimarytmx* -u *tvauserid* <-a *protofile* > <-C [*configfilesection*]> <-D *dateformat* > <-d *protomsg* > <-E *tokenlocation* > <-e *tvapubmsgexp* > <-g *gdconfig* > <-i *tvalogfile* > <-j trace | debug | info | warn | error | fatal | off> <-l native | solace | tervela | rabbitmq | kafka> <-r *tvapubrate* > <-s *tvasecondarytmx* > <-V> <-w *tvapubbwlimit* > <-X> <-y *logconfigfile* > <-Z *transportconfigfile*>

Publisher usage:

dfesp_tva_adapter -c *tvaclientname* -f *tvatopic* -h *url* -k pub -n *tvasubname* -o *urlhostport* -p *tvapassword* -t *tvaprimarytmx* -u *tvauserid* <-a *protofile* > <-C [*configfilesection*]> <-D *dateformat* > <-E *tokenlocation* > <-g *gdconfig* > <-i *tvalogfile* > <-j trace | debug | info | warn | error | fatal | off> <-l native | solace | tervela | rabbitmq | kafka> <-m *protomsg* > <-Q> <-R> <-s *tvasecondarytmx* > <-V> <-X> <-Y *maxevents*> <-y *logconfigfile* > <-Z *transportconfigfile*>

Parameter	Description
`-a protofile`	Specifies the **`.proto`** file that contains the message used for Google Protocol buffer support.
`-b numbufferedmsgs`	Specifies the maximum number of messages buffered by a standby subscriber connector.
`-C [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]`.
`-c tvaclientname`	Specifies the Tervela client name.

Parameter	Description
`-D dateformat`	Specifies the format of `ESP_DATETIME` and `ESP_TIMESTAMP` fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (`ESP_DATETIME`) or microseconds (`ESP_TIMESTAMP`) since epoch. The `dateformat` parameter accepts any time format that is supported by the UNIX `strftime` function.
`-d protomsg`	Specifies the message itself in the **`.proto`** file that is specified by the protofile parameter.
`-E tokenlocation`	Specifies the location of the file in the local file system that contains the OAuth token that is required for authentication by the publish/subscribe server.
`-e tvapubmsgexp`	Specifies the Tervela maximum time to cache published messages (seconds); the default value is 1.
`-f tvatopic`	Specifies the Tervela topic.
`-g gdconfig`	Specifies the guaranteed delivery configuration file.
`-h url`	Specifies the dfESP publish and subscribe 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 the following for subscribers if needed: ■ `?collapse=true | false` ■ `?rmretdel=true | false`
`-i tvalogfile`	Specifies the Tervela log file. The default is “`syslog`”.
`-j 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.
`-k`	Specifies `sub` for subscriber use and `pub` for publisher use
`-l 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.
`-m tvamaxoutstand`	Specifies the Tervela maximum number of unacknowledged messages.
`-o urlhostport`	Specifies the *host:port* string sent in connector metadata message
`-p tvapassword`	Specifies the Tervela password.
`-Q`	When *maxevents* is configured, quiesces the project after all events are injected into the Source window.

Parameter	Description
`-R`	Builds events with `opcode=Upsert` rather than `opcode=Insert`.
`-r tvapubrate`	Specifies the Tervela publish rate (Kbps). The default value is 30.
`-s tvasecondarytmx`	Specifies the Tervela secondary TMX.
`-t tvaprimarytmx`	Specifies the Tervela primary TMX.
`-u tvauserid`	Specifies the Tervela user name.
`-V`	Specifies to restart the adapter if a fatal error is reported.
`-w tvapubbwlimit`	Specifies the Tervela maximum bandwidth of published data (Mbps). The default value is 100.
`-X`	Specifies that `tvapassword` is encrypted.
`-Y maxevents`	Specifies the maximum number of events to publish.
`-y logconfigfile`	Specifies the log configuration file.
`-Z transportconfigfile`	Specifies the publish/subscribe transport configuration file. The default value depends on the transport type specified with the `-l` parameter: For `-l solace`, the default is **`./solace.cfg`**. For `-l tervela`, the default is **`./client.config`**. For `-l rabbitmq`, the default is **`./rabbitmq.cfg`**. For `-l kafka`, the default 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: Advanced Topics*.

Using the Tibco Rendezvous Connector and Adapter

Using the Tibco Rendezvous (RV) Connector

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

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

Note: Remove the reference to this connector from the file **`/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.excluded`** (Linux) or **`%ProgramData%\SAS`**

\Viya\SASEventStreamProcessingEngine\default\connectors.excluded (Windows) before using it.

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 64 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**, 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 65 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**, or **opaquestring**. For **opaquestring**, the Source window schema is assumed to be `index:int64,message:string`.

Table 66 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 `ESP_DATETIME` and `ESP_TIMESTAMP` fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (`ESP_DATETIME`) or microseconds (`ESP_TIMESTAMP`) 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.
`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.
`csvmsgperevent`	For CSV, specifies to send one message per event. The default is one message per transactional event block or else one message per event.
`csvmsgpereventblock`	For CSV, specifies to send one message per event block. The default is one message per transactional event block or else one message per event.
`doubleprecision`	Specifies the number of fractional digits in the ASCII representation of a double. The default value is 6.

Table 67 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.

Parameter	Description
transactional	Sets the event block type to transactional. The default event block type is normal.
dateformat	Specifies the format of ESP_DATETIME and ESP_TIMESTAMP fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (ESP_DATETIME) or microseconds (ESP_TIMESTAMP) 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 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.
csvfielddelimiter	Specifies the character delimiter for field data in input CSV events. The default delimiter is the , character.
noautogenfield	Specifies that input events are missing the key field that is autogenerated by the source window.
publishwithupsert	Specifies to build events with opcode=Upsert instead of opcode=Insert.
addcsvopcode	Prepends an opcode and comma to input CSV events. The opcode is Insert unless publishwithupsert is enabled.
addcsvflags	Specifies the event type to insert into input CSV events (with a comma). Valid values are normal and partialupdate.
maxevents	Specifies the maximum number of events to publish.

Using the Tibco Rendezvous (RV) Adapter

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.

Subscriber usage:

dfesp_tibrv_adapter -f *tibrvsubject* -h *url* -k sub -s *tibrvservice* -t *tibrvtype* <-A> <-a *protofile* > <-b *protomsg* > <-C [*configfilesection*]> <-D *csvfielddelimiter* > <-d *dateformat* > <-E *tokenlocation* > <-g *gdconfig* > <-j trace | debug | info | warn | error | fatal | off> <-L> <-l native | solace | tervela | rabbitmq | kafka> <-M> <-m *tibrvdaemon* > <-n *tibrvnetwork* > <-R> <-V> <-y *logconfigfile* > <-Z *transportconfigfile*> <-3 *doubleprecision*>

Publisher usage:

dfesp_tibrv_adapter -f *tibrvsubject* -h *url* -k pub -s *tibrvservice* -t *tibrvtype* <-a *protofile* > <-b *protomsg* > <-C [*configfilesection*]> <-d *dateformat* > <-E *tokenlocation* > <-e> <-F *eventtype*> <-g *gdconfig* > <-I> <-j trace | debug | info | warn | error | fatal | off> <-l native | solace | tervela | rabbitmq | kafka> <-m *tibrvdaemon* > <-n *tibrvnetwork* > <-O> <-Q> <-r *blocksize* > <-V> <-Y *maxevents*> <-y *logconfigfile* ><-Z *transportconfigfile*>

Parameter	Description
`-A`	Specifies that input events are missing the key field that is autogenerated by the source window.
`-a protofile`	Specifies the **`.proto`** file that contains the message used for Google Protocol buffer support.
`-b blocksize`	Specifies the block size. The default value is 1.
`-C [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]`.
`-D csvfielddelimiter`	Specifies the character delimiter for field data in input CSV events. The default delimiter is the `,` character.
`-d dateformat`	Specifies the format of `ESP_DATETIME` and `ESP_TIMESTAMP` fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (`ESP_DATETIME`) or microseconds (`ESP_TIMESTAMP`) since epoch. The `dateformat` parameter accepts any time format that is supported by the UNIX `strftime` function.
`-E tokenlocation`	Specifies the location of the file in the local file system that contains the OAuth token that is required for authentication by the publish/subscribe server
`-e`	Specifies that events are transactional.
`-F eventtype`	Specifies the event type to Insert into input CSV events (with comma). Valid values are `"normal"` and `"partialupdate"`.
`-f tibrvsubject`	Specifies the Tibco RV subject.
`-g gdconfig`	Specifies the guaranteed delivery configuration file.

Parameter	Description
`-h url`	Specifies the dfESP publish and subscribe 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 the following for subscribers if needed: ■ `?collapse= true | false` ■ `?rmretdel= true | false`
`-I`	Specifies that when a field in an input CSV event cannot be parsed, the event is dropped, an error is logged, and publishing continues.
`-j 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.
`-k`	Specifies `sub` for subscriber use and `pub` for publisher use.
`-L`	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.
`-l 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.
`-M`	For CSV, specifies to send one message per event. The default is one message per transactional event block or else one message per event.
`-m tibrvdaemon`	Specifies the Tibco RV daemon.
`-n tibrvnetwork`	Specifies the Tibco RV network.
`-O`	Prepends an opcode and comma to input CSV events. The opcode is Insert unless `-R` is enabled.
`-Q`	When *maxevents* is configured, quiesces the project after all events are injected into the Source window.
`-R`	Builds events with `opcode=Upsert` instead of Insert.
`-r protomsg`	Specifies the message itself in the **`.proto`** file that is specified by the protofile parameter.
`-s tibrvservice`	Specifies the Tibco RV service.
`-t tibrvtype`	Specifies `binary`, `CSV`, or `JSON`. `opaquestring` is supported for publishers. For `opaquestring`, the Source window schema is assumed to be `"index:int64,message:string"`. For subscribers, the name of a string field in the subscribed window schema is also supported.

Parameter	Description
`-V`	Restarts the adapter if a fatal error is reported.
`-Y maxevents`	Specifies the maximum number of events to publish.
`-y logconfigfile`	Specifies the log configuration file.
`-Z transportconfigfile`	Specifies the publish/subscribe transport configuration file. The default value depends on the transport type specified with the `-l` parameter: For `-l solace`, the default is **`./solace.cfg`**. For `-l tervela`, the default is **`./client.config`**. For `-l rabbitmq`, the default is **`./rabbitmq.cfg`**. For `-l kafka`, the default is **`./kafka.cfg`**. **Note:** No transport configuration file is required for native transport.
`-3 doubleprecision`	Specifies the number of fractional digits in the ASCII representation of a double. The default value is 6.

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

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 68 Required Parameters for Timer Connectors

Parameter	Description
`type`	Specifies to publish or subscribe.

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

Table 69 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.

Usage:

dfesp_timer_adapter -b *basetime* -h *url* -i *interval* -u second | minute | hour | day | week | month | year <-C *configfilesection*> <-E *tokenlocation*> <-e> <-f *timeformat*> <-g *gdconfig*> <-j trace | debug | info | warn | error | fatal | off> <-L *label*> <-l native | solace | tervela | rabbitmq | kafka> <-m *maxevents*> <-Q> <-R> <-V> <-y *logconfigfile*> <-Z *transportconfigfile*>

Usage:

Parameter	Description
-b basetime	Specifies the start time in the format defined by the timeformat parameter.

Parameter	Description
`-C [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]`.
`-E tokenlocation`	Specifies the location of the file in the local file system that contains the OAuth token that is required for authentication by the publish/subscribe server.
`-e`	Specifies that events are `transactional`.
`-f timeformat`	Specifies the format of the `basetime` parameter. The default value is `%Y-%m-%d %H:%M:%S`.
`-g gdconfig`	Specifies the guaranteed delivery configuration file.
`-h url`	Specifies the dfESP publish and subscribe standard URL in the form dfESP://host:port/project/continuousquery/window.
`-i interval`	Specifies the interval length in units defined by the `unit` parameter.
`-j 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.
`-L label`	Specifies the string to be written to the Source window `label` field. The default value is the connector name.
`-l 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.
`-m maxevents`	Specifies the maximum number of events to publish.
`-Q`	When *maxevents* is configured, quiesces the project after all events are injected into the Source window.
`-R`	Builds events with `opcode=Upsert` instead of `opcode=Insert`.
`-u second | minute | hour | day | week | month | year`	Specifies the units of the `interval` parameter.
`-V`	Restarts the adapter if a fatal error is reported.
`-y logconfigfile`	Specifies the log configuration file.

Parameter	Description
`-Z transportconfigfile`	Specifies the publish/subscribe transport configuration file. The default value depends on the transport type specified with the `-l` parameter: For `-l solace`, the default is **`./solace.cfg`**. For `-l tervela`, the default is **`./client.config`**. For `-l rabbitmq`, the default is **`./rabbitmq.cfg`**. For `-l kafka`, the default is **`./kafka.cfg`**. **Note:** No transport configuration file is required for native transport.

Using the Twitter Publisher Adapter

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

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

Usage:

dfesp_twitter_publisher -m sample | filter | links | firehose | retweet -u *url* <-a *languages* > < -b *blocksize* > <-C *prevcount* > <-c [*configfilesection*]> <-D *dumpfile* > <-d *dateformat* > <-f *followlist* > < -g *gdconfigfile* > <-k *tracklist* > < -l native | solace | tervela | rabbitmq | kafka> <-O *tokenlocation* > <-o severe | warning | info> < -p *locations* > <-s> <-T *twitterpropfile* > <-t> <-V> <-W *maxevents*> <-Z *transportconfigfile*>

Parameter	Description
`-a languagelist`	Specifies a comma-separated list of BCP 47 language identifiers to return only Tweets that have been detected as being written in the specified languages. For example, connecting with “en,fr” streams only Tweets detected to be in the English or French language. **Note:** This parameter must be used in conjunction with at least one of the filter predicates of the filter access method. The `-a languagelist` parameter is only supported with the filter access method, specified at the command line with the `-m filter` argument. The filter access method requires at least one of its supported filter predicates. For more information, see Table 70.
`-b blocksize`	Specifies the number of events per event block.
`-C prevcount`	Used with filter, link, or retweet method. Indicates the number of previous statuses to stream before transitioning to the live stream. Default is 0. The supplied value can be an integer from 1 to 150000 or from -1 to -150000. If a positive number is specified, the stream transitions to live values after the backfill has been delivered to the client. When a negative number is specified, the stream disconnects after the backfill has been delivered to the client, which might be useful for debugging. This parameter requires elevated access to use. See Streaming API request parameters for more information.
`-c [configfilesection]`	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.
`-D dumpfile`	Specifies the full path of a file in which to dump binary event blocks for replay. Use `dfESP_fs_adapter` to replay event blocks.
`-d dateformat`	Specifies the format of `ESP_DATETIME` and `ESP_TIMESTAMP` fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (`ESP_DATETIME`) or microseconds (`ESP_TIMESTAMP`) since epoch. The `dateformat` parameter accepts any time format that is supported by the UNIX `strftime` function.
`-f followlist`	Specifies the list of user IDs to follow with the filter method in the form “user1,user2,user3”. The default access level allows up to 400 follower user IDs. See Streaming API request parameters for more details.
`-g gdconfigfile`	Specifies the guaranteed delivery configuration file for the client.

Parameter	Description
-k tracklist	Specifies the list of keywords to track with the filter method in the form “keyword1,keyword2,keyword3”. The default access level allows up to 200 track keywords.
-l native | solace | tervela | rabbitmq | kafka	Specifies the transport type. When you specify solace, tervela, rabbitmq, or kafka transports instead of the default native transport, use the required client configuration files specified in “Using Alternative Transport Libraries for Java Clients” in *SAS Event Stream Processing: Publish/Subscribe API* .
-m sample | filter | links | firehose | retweet	Specifies the method type for subscribing to Twitter streams. See the table that follows.
-O tokenlocation	Specifies the location of the file in the local file system that contains the OAuth token that is required for authentication by the publish/subscribe server
-o severe | warning | info	Specifies the application logging level.
-p locationlist	Specifies the list of locations to track with the filter method in the form of a comma-separated list of longitude,latitude pairs. Each pair specifies a set of bounding boxes with the southwest corner of the bounding box coming first. Sample: “-122.75,36.8,-121.75,37.8” for San Francisco. Note that bounding boxes do not act as filters for other filter parameters. The default access level allows up to 10 1-degree location boxes.
-s	Builds events with opcode=Upsert instead of Insert. **javaadapters.config** parameter name: publishwithupsert
-T twitterpropfile	Specifies the name of the file in your configuration directory to parse for Twitter configuration parameters. Default is **twitterpublisher.properties**.
-t	Specifies that event blocks are transactional. The default event block type is normal. **javaadapters.config** parameter name: transactional
-u url	Specifies the publish standard URL in the form "**dfESP://*host*:*port*/*project*/*continuousquery*/*window***".
-V	Restarts the adapter if a fatal error is reported. **javaadapters.config** parameter name: restartonerror
-W maxevents	Specifies the maximum number of events to publish.

Parameter	Description
`-Z transportconfigfile`	Specifies the publish/subscribe transport configuration file. The default value depends on the transport type specified with the `-l` parameter: For `-l solace`, the default is **`./solace.cfg`**. For `-l tervela`, the default is **`./client.config`**. For `-l rabbitmq`, the default is **`./rabbitmq.cfg`**. For `-l kafka`, the default 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: Advanced Topics*.

Twitter sets limitations to access streams and uses some methods depending on the user-account level of access:

Table 70 *Twitter Stream Access Methods*

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

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

To authorize a Twitter user account to be used by the Twitter adapter, you need to grant access to your Twitter user account on behalf of the SAS Twitter app ID. The Twitter properties file located in the SAS Event Stream Processing configuration directory must include an encrypted access token in order for a user account to access the adapter. By default, the Twitter properties file is named **`twitterpublisher.properties`** and does not include the access token. To generate and write an encrypted access token to the properties file, run the dfesp_twitter_auth utility that is located in **`$DFESP_HOME/bin`**. Once the token is included in the properties file, verify that the adapter runs successfully. You do not need to run the dfesp_twitter_auth utility more than once for the same Twitter account.

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

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

```
tw_ID*:int64,tw_AuthorScreenName:string,tw_AuthorId:int64,tw_AuthorDescrip
tion:string,tw_AuthorFavouritesCount:int32,tw_AuthorFollowersCount:int32,tw
_AuthorFriendsCount:int32,tw_AuthorProfileImageURL:string,tw_AuthorLang:str
ing,tw_AuthorLocation:string,tw_AuthorName:string,tw_AuthorTimeZone:string,
tw_AuthorURL:string,tw_AuthorStatusesCount:int32,tw_AuthorListedCount:int32
,tw_Text:string,tw_CreatedAt:stamp,tw_CurrentUserRetweetId:int64,tw_Favorit
eCount:int32,tw_GeolocLatitude:double,tw_GeolocLongitude:double,tw_InReplyT
oScreenName:string,tw_InReplyToStatusId:int64,tw_InReplyToUserId:int64,tw_L
ang:string,tw_RetweetCount:int32,tw_RetweetedStatusId:int64,tw_SourceText:s
tring,tw_Source:string,tw_isFavorited:int32,tw_isPossiblySensitive:int32,tw
_isRetweet:int32,tw_isRetweeted:int32,tw_isTruncated:int32,tw_PlaceFullName
:string,tw_MentionedUserNames:string,tw_MentionedUserScreenNames:string,tw_
MentionedUserIds:string,tw_StatusLinks:string,tw_StatusTags:string,f_method
:string,f_followed_IDs:string,f_tracked_terms:string,f_GeoLocations:string,
f_Languages:string
```

Field	Type	Description
`tw_ID`	int64	The ID of the status
`tw_AuthorScreenName`	string	The screen name of the user ID associated with the status
`tw_AuthorId`	int64	The user ID associated with the status.
`tw_AuthorDescription`	string	The description of the author
`tw_AuthorFavouritesCount`	int32	The number of favorites of the author
`tw_AuthorFollowersCount`	int32	The number of followers of the author
`tw_AuthorFriendsCount`	int32	The number of friends of the author
`tw_AuthorProfileImageURL`	string	The profile image url of the author
`tw_AuthorLang`	string	The preferred language of the author
`tw_AuthorLocation`	string	The location of the author
`tw_AuthorName`	string	The name of the author
`tw_AuthorTimeZone`	string	The time zone of the author

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

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

Using the Twitter Gnip Publisher Adapter

Using the Adapter

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

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

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

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

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

Usage:

dfesp_twittergnip_publisher -esp-url *url* -streampassword *streampassword* -streamurl *streamurl* -streamuserid *streamuserid* < <-esp-dateformat *dateformat* > > <-esp-ignoremissingschemafields > <-esp-logconfigfile *logconfigfile* > < <-esp-loglevel *loglevel* > > <-esp-matchsubstrings *substrings* > <-esp-publishwithupsert> <-gnipkeepalive *gnipkeepalive*><--maxrecords *maxrecords* >

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

Parameter	Description
`-streamurl streamurl`	Specifies the Gnip streaming URL, in the form `https://stream.gnip.com:443/accounts/account/publishers/twitter/streams/stream/label.json`
`-streamuserid streamuserid`	Specifies the Gnip user name.

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

Using the Twitter Gnip Query Registration Utility

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

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

Viewing registered queries shows all matching queries unless you use the `groupname` parameter to filter them. This parameter corresponds to the `tag` key that is associated with the `value` key in a `rules` array entry. A rule is shown when the specified `groupname` appears as a substring in the rule's `tag` value.

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

Using the URL Connector

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

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

A single URL request can generate many events. The URL connector uses event loop technology to parse, transform, and publish the data into SAS Event Stream Processing 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 SAS Event Stream Processing. For example configuration files, see the URL connector examples located in the examples folder of your SAS Event Stream Processing installation directory:

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

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

The URL connector enables you to specify any number of publishers for the associated Source window. Each publisher has its own transformation rules to prepare the data for event stream processing publishing. Each publisher can have any number of requests defined for that 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 and publish them into SAS Event Stream Processing using a single publisher and several request definitions.

The URL connector can only be used to publish events.

Table 71 *Required Parameters for Publisher URL Connectors*

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

Table 72 *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 resolution of the streaming photos, the format of the image, the quality of the camera, and 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. 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, 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, which might not be the same as the published frame rate that is specified by the `frame_rate` parameter. The image that is published is always the latest captured image.

V4L2–compatible cameras support blocking and non-blocking modes. V4L2 uses output buffers to deliver photos to users. If the output buffer is not filled with data and available, a connector in blocking mode waits, and a connector in non-blocking mode returns immediately without data. Enable blocking with the `blocking` parameter.

The UVC connector can be used only to publish events.

Table 73 Required Parameters for the UVC Publisher Connector

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

Table 74 Optional Parameters for the UVC Publisher Connector

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

Parameter	Description
`device`	Specifies the device name the camera is using on the Linux operating system. The device name is typically **`/dev/videoX`**.
`blocking`	Specifies whether the connector is in blocking mode. The default is `false` (non-blocking).
`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.

Using the UVC Adapter

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

Publisher usage:

dfesp_uvc_adapter -h *url* <-B> <-b *blocksize*> <-C *configfilesection*> <-d *device*> <-E *tokenlocation*> <-e> <-g *gdconfig*> <-H *height*> <-j trace | debug | info | warn | error | fatal | off> <-l native | solace | tervela | rabbitmq | kafka> <-m *maxevents*> <-Q> <-R> <-r *frame_rate*> <-V> <-W *width*> <-y *logconfigfile*> <-Z *transportconfigfile*>

Parameter	Description
`-B true|false`	If `true`, sets the adapter to blocking mode. The default is “`false`”.
`-b blocksize`	Specifies event block size. The default is `1`.
`-C [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]`.
`-D delay`	Specifies the number of seconds to delay before a retry if unable to start camera. The default is `0`.
`-d devicename`	Specifies the device name. The default is “`/dev/video0`”.

Parameter	Description
`-E tokenlocation`	Specifies the location of the file in the local file system that contains the OAuth token. This token is required for authentication by the publish/subscribe server.
`-e`	Specifies that events are transactional.
`-g gdconfig`	Specifies the guaranteed delivery configuration file.
`-H height`	Specifies the height for the frame resolution. The default is `480`.
`-h url`	Specifies the ESP publish/subscribe standard URL in the form dfESP://<host>:<port>/<project>/<contquery>/<window> .
`-j trace | debug | info | warn | error | fatal | off`	Specifies the logging level. The default is “`warn`”.
`-l native | solace | tervela | rabbitmq | kafka`	Specifies the publish/subscribe transport. The default is `native`.
`-m maxevents`	Specifies the maximum number of events to publish.
`-Q`	When *maxevents* is configured, quiesces the project after all events are injected into the Source window.
`-R`	Builds events with `opcode=Upsert` instead of `Insert`.
`-r framespersecond`	Specifies the number of frames per second. The default is `15`.
`-V`	Restarts the adapter if a fatal error is reported.
`-W width`	Specifies the width for the frame resolution. The default is `640`.
`-y logconfigfile`	Specifies the logging configuration file. By default, there is none.
`-Z transportconfigfile`	Specifies the publish/subscribe transport configuration file. The default value depends on the transport type specified with the `-l` parameter: For `-l solace`, the default is **`./solace.cfg`**. For `-l tervela`, the default is **`./client.config`**. For `-l rabbitmq`, the default is **`./rabbitmq.cfg`**. For `-l kafka`, the default 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: Advanced Topics*.

Using the WebSocket Connector

Overview

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 75 Required Parameters for WebSocket Connectors

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

Table 76 Optional Parameters for WebSocket Connectors

Parameter	Description
`sslCertificate`	Specifies the location of the SSL certificate to use when connecting to a secure server.
`sslPassphrase`	Specifies the password for the SSL certificate.

Parameter	Description
`requestHeaders`	Specifies a comma-separated list of request headers to send to the server. The list must consist of name-value pairs in `name:value` format.
`maxevents`	Specifies the maximum number of events to publish.

Example: Using the WebSocket Connector to Publish from Teradata Listener

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

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

```
<connectors>
  <connector class='websocket'>
    <properties>
      <property name='type'>pub</property>
      <property name='url'>wss://listener-streamer-services-poc.labs.teradata.com/v1/streamer/
                444186fb-b652-485f-9dc3-c00c60863476?secret=e52378aa-b7aa-4722-a226-ca18225481f9</property>
      <property name='contentType'>json</property>
      <property name='configUrl'>file://ws.xml</property>
      <property name='sslCertificate'>/etc/pki/tls/certs/listener.crt</property>
    </properties>
  </connector>
</connectors>
```

For the `url` WebSocket connector parameter, specify the URL for the Listener Stream API (version 1) that includes the Stream identifier and the Stream secret key. To obtain these values, contact your Teradata Listener administrator.

For the `SSLCert` WebSocket connector parameter, specify the path to a file that contains SSL certificates for securely connecting to the Listener Stream service. Listener uses TLS 1.2.

Using the IBM WebSphere MQ Connector and Adapter

Using the IBM WebSphere MQ Connector

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

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

Note: Remove the reference to this connector from the file **/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.excluded** (Linux) or **%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.excluded** (Windows) before using it.

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

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

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

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

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

These parameters are as follows:

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

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

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

Use the following parameters for MQ connectors.

Table 77 Required Parameters for Subscriber MQ Connectors

Parameter	Description
`type`	Specifies to subscribe. Must be "`sub`".
`mqtype`	Specifies `binary`, `CSV`, `JSON`, `XML`, 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 78 Required Parameters for Publisher MQ Connectors

Parameter	Description
`type`	Specifies to publish. Must be “`pub`”.
`mqtype`	Specifies `binary`, `CSV`, `JSON`, `XML`, or `opaquestring`. For `opaquestring`, the Source window schema is assumed to be `index:int64,message:string`.

Table 79 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 `ESP_DATETIME` and `ESP_TIMESTAMP` fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (`ESP_DATETIME`) or microseconds (`ESP_TIMESTAMP`) since epoch. The `dateformat` parameter accepts any time format that is supported by the UNIX `strftime` function.
`rmretdel`	Specifies to remove all delete events from event blocks received by a subscriber that were introduced by a window retention policy.
`configfilesection`	Specifies the name of the section in **`/opt/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.
`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.

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 80 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 `ESP_DATETIME` and `ESP_TIMESTAMP` fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (`ESP_DATETIME`) or microseconds (`ESP_TIMESTAMP`) 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.
`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.
`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.

Parameter	Description
`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.

Using the IBM WebSphere MQ Adapter

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.

Subscriber usage:

dfesp_mq_adapter -h *url* -k sub -t *mqtype* <-a *protofile* > <-C [*configfilesection*]> <-c> <-d *dateformat* > <-E *tokenlocation* > <-f *mqtopic*> <-g *gdconfig* > <-j trace | debug | info | warn | error | fatal | off> <-L> <-l native | solace | tervela | rabbitmq | kafka> <-M> <-m *protomsg* > <-q *mqqueuemanager* > <-U *mqqueue*> <-V> <-y *logconfigfile* > <-Z *transportconfigfile*> <-3 *doubleprecision*>

Publisher usage:

dfesp_mq_adapter -h *url* -k pub -t *mqtype* < -A> <-a *protofile* > <-b *blocksize* > <-C [*configfilesection*]> <-c> < -D *csvfielddelimiter* > <-d *dateformat* > <-E *tokenlocation* > <-e> <-F *eventtype*> <-f *mqtopic*> <-g *gdconfig* > <-I> <-j trace | debug | info | warn | error | fatal | off> <-l native | solace | tervela | rabbitmq | kafka> <-m *protomsg* > <-n *mqsubname* > <-O> <-o> <-Q> <-q *mqqueuemanager* > <-R> <-U *mqqueue*> <-V> <-Y *maxevents*> <-y *logconfigfile* > <-Z *transportconfigfile*>

Parameter	Description
`-A`	Specifies that input events are missing the key field that is autogenerated by the Source window.
`-a protofile`	Specifies the **`.proto`** file that contains the message used for Google Protocol buffer support.
`-b blocksize`	Specifies the block size. The default value is 1.

Parameter	Description
`-C [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]`.
`-c`	Copies the MQ correlation ID to or from the **`correlid`** string field in the SAS Event Stream Processing event.
`-D csvfielddelimiter`	Specifies the character delimiter for field data in input CSV events. The default delimiter is the `,` character.
`-d dateformat`	Specifies the format of `ESP_DATETIME` and `ESP_TIMESTAMP` fields in CSV events. The default behavior is that these fields are interpreted as an integer number of seconds (`ESP_DATETIME`) or microseconds (`ESP_TIMESTAMP`) since epoch. The `dateformat` parameter accepts any time format that is supported by the UNIX `strftime` function.
`-E tokenlocation`	Specifies the location of the file in the local file system that contains the OAuth token required for authentication by the publish/subscribe server.
`-e`	Specifies that events are transactional.
`-F eventtype`	Specifies the event type to Insert into input CSV events (with comma). Valid values are `"normal"` and `"partialupdate"`.
`-f mqtopic`	Specifies the MQ topic name. Required if `mqtopic` is not configured.
`-g gdconfig`	Specifies the guaranteed delivery configuration file.
`-h url`	Specifies the dfESP publish and subscribe 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 the following for subscribers if needed: ■ `?collapse=true | false` ■ `?rmretdel=true | false`
`-I`	Specifies that when a field in an input CSV event cannot be parsed, the event is dropped, an error is logged, and publishing continues.
`-j 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.
`-k`	Specifies `sub` for subscriber use and `pub` for publisher use
`-L`	For CSV, specifies to send one message per event block. The default is one message per transactional event block or else one message per event.

Parameter	Description
`-l 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.
`-M`	For CSV, specifies to send one message per event. The default is one message per transactional event block or else one message per event.
`-m protomsg`	Specifies the message itself in the **`.proto`** file that is specified by the protofile parameter.
`-n mqsubname`	Specifies the MQ subscription name. Required if `mqqueue` is not configured.
`-O`	Prepends an opcode and comma to input CSV events. The opcode is Insert unless `-R` is enabled.
`-o`	Specifies to ignore the value of the Message Descriptor Format parameter, and assume the message format is compatible with the `mqtype` parameter setting.
`-Q`	When *maxevents* is configured, quiesces the project after all events are injected into the Source window.
`-q mqqueuemanager`	Specifies the MQ queue manager name.
`-R`	Builds events with `opcode=Upsert` instead of `Insert`.
`-t mqtype`	Specifies **`binary`**, **`CSV`**, **`XML`**, or **`JSON`**. For publishers, **`opaquestring`** is also supported. For **`opaquestring`**, the Source window schema is assumed to be `"index:int64,message:string"`. For subscribers, the name of a string field in the subscribed window schema is also supported.
`-U mqqueue`	Specifies the MQ queue name.
`-V`	Restarts the adapter if a fatal error is reported.
`-Y maxevents`	Specifies the maximum number of events to publish.
`-y logconfigfile`	Specifies the log configuration file.
`-Z transportconfigfile`	Specifies the publish/subscribe transport configuration file. The default value depends on the transport type specified with the `-l` parameter: For `-l solace`, the default is **`./solace.cfg`**. For `-l tervela`, the default is **`./client.config`**. For `-l rabbitmq`, the default is **`./rabbitmq.cfg`**. For `-l kafka`, the default is **`./kafka.cfg`**. **Note:** No transport configuration file is required for native transport.
`-3 doubleprecision`	Specifies the number of fractional digits in the ASCII representation of a double. The default value is 6.

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

Writing and Integrating a Custom Connector

Writing a Custom Connector

When you write your own connector, the connector class must inherit from base class `dfESPconnector`.

Connector configuration is maintained in a set of key or value pairs where all keys and values are text strings. A connector can obtain the value of a configuration item at any time by calling `getParameter()` and passing the key string. An invalid request returns an empty string.

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

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

A connector must define these static data structures:

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

A connector must define these static methods:

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

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

A connector must define these virtual methods:

Virtual Method	Description
`start()`	Starts the connector. Must call base class method `checkConfig()` to validate connector configuration before starting. Must also call base class method `start()`. Must set `variable _started = true` upon success.
`stop()`	Stops the connector. Must call base class method `stop()`. Must leave the connector in a state whereby `start()` can be subsequently called to restart the connector.
`callbackFunction()`	Specifies the method invoked by the engine to pass event blocks generated by the window to which it is connected.
`errorCallbackFunction()`	Specifies the method invoked by the engine to report errors detected by the engine. Must call user callback function `_errorCallback`, if nonzero.
`setupCallbackFunction()`	Specifies the method invoked by the engine to set up any connector that requires the Source window schema.

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

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

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

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

An application would invoke the method as follows:

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

Integrating a Custom Connector

All connectors are managed by a global connector manager. The default connectors shipped with 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 in **`/opt/sas/viya/config/etc/SASEventStreamProcessingEngine/default/connectors.excluded`** (Linux) or **`%ProgramData%\SAS\Viya\SASEventStreamProcessingEngine\default\connectors.excluded`** (Windows).

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.