Overview

When it comes to ingesting data from ***SAS ESP*** directly into a CAS server, a couple of different techniques are available to users:

- Data ingestion through the ***SAS ESP*** CAS adapter, or
- Data ingestion through the ***loadStream*** CAS actionSet

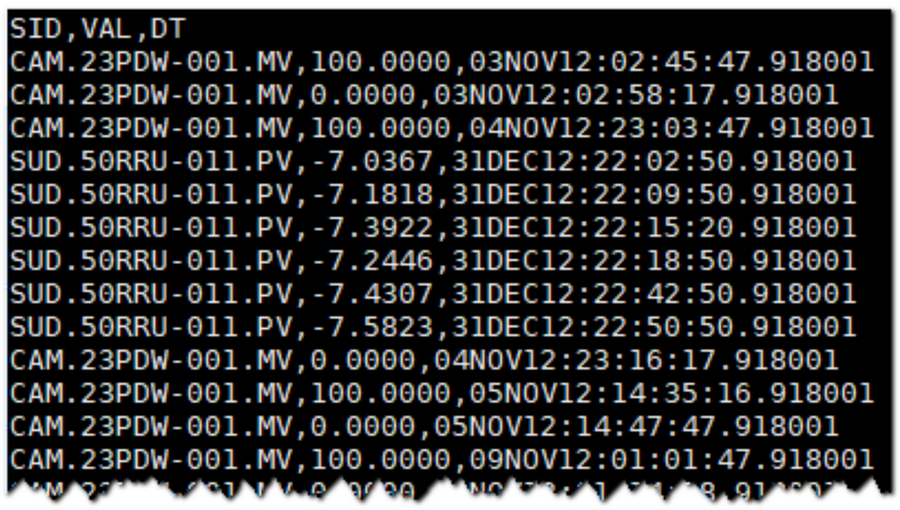
The attached document focuses on the latter technique, showing how CAS actions can be used programmatically to process any amounts of data in a very simple way. The document applies to all versions of Viya, cloud and non-cloud based. The sample file referenced in the document is also provided to make it easy to replicate the process on your own environment.

Key Takeaways:

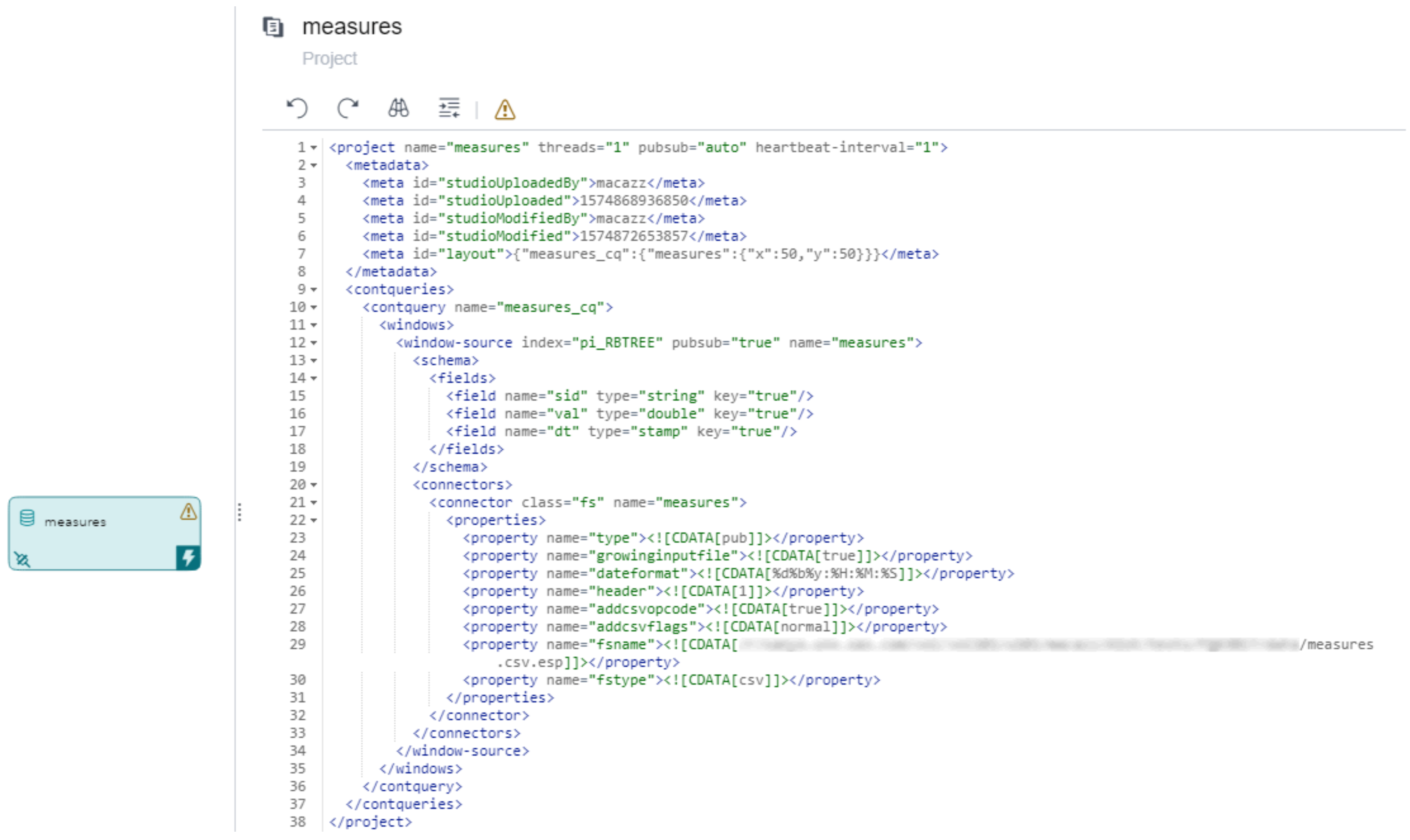
• Learn how to use ***SAS ESP Studio*** to create and execute projects to stream data into CAS
• Learn how to access a data stream through the ***SAS Viya CASL*** language and its ***loadStream*** actionSet

The following is a how-to document that shows how to programmatically ingest data from SAS ESP into CAS using the CAS loadStream actionSet. The scenario described in the document applies to both cloud-based (Viya 4 and later) and non-cloud based (Viya 3.5 and earlier) environments and assumes the data to be stored in a CSV file. However, any other storage mean could be used so long as ESP has access to it through one of its adapters, as the overall process is independent of the location of the input data. Please check the SAS documentation for a comprehensive list of available adapters based on the version of ESP used.

1. Using SAS ESP Studio, create a project to describe the layout of the data being ingested. The two pictures below show a snapshot of the sample input file and its corresponding project definition:



```
SID,VAL,DT
CAM.23PDW-001.MV,100.0000,03NOV12:02:45:47.918001
CAM.23PDW-001.MV,0.0000,03NOV12:02:58:17.918001
CAM.23PDW-001.MV,100.0000,04NOV12:23:03:47.918001
SUD.50RRU-011.PV,-7.0367,31DEC12:22:02:50.918001
SUD.50RRU-011.PV,-7.1818,31DEC12:22:09:50.918001
SUD.50RRU-011.PV,-7.3922,31DEC12:22:15:20.918001
SUD.50RRU-011.PV,-7.2446,31DEC12:22:18:50.918001
SUD.50RRU-011.PV,-7.4307,31DEC12:22:42:50.918001
SUD.50RRU-011.PV,-7.5823,31DEC12:22:50:50.918001
CAM.23PDW-001.MV,0.0000,04NOV12:23:16:17.918001
CAM.23PDW-001.MV,100.0000,05NOV12:14:35:16.918001
CAM.23PDW-001.MV,0.0000,05NOV12:14:47:47.918001
CAM.23PDW-001.MV,100.0000,09NOV12:01:01:47.918001
```



```
<project name="measures" threads="1" pubsub="auto" heartbeat-interval="1">
  <metadata>
    <meta id="studioUploadedBy">macazz</meta>
    <meta id="studioUploaded">1574868936850</meta>
    <meta id="studioModifiedBy">macazz</meta>
    <meta id="studioModified">1574872653857</meta>
    <meta id="layout">{"measures_cq":{"measures":{"x":50,"y":50}}}</meta>
  </metadata>
  <contqueries>
    <contquery name="measures_cq">
      <windows>
        <window-source index="pi_RBTREE" pubsub="true" name="measures">
          <schema>
            <fields>
              <field name="sid" type="string" key="true"/>
              <field name="val" type="double" key="true"/>
              <field name="dt" type="stamp" key="true"/>
            </fields>
          </schema>
          <connectors>
            <connector class="fs" name="measures">
              <properties>
                <property name="type"><![CDATA[pub]]></property>
                <property name="growinginputfile"><![CDATA[true]]></property>
                <property name="dateformat"><![CDATA[%d%b%y:%H:%M:%S]]></property>
                <property name="header"><![CDATA[1]]></property>
                <property name="addcsvopcode"><![CDATA[true]]></property>
                <property name="addcsvflags"><![CDATA[normal]]></property>
                <property name="fsname"><![CDATA[[redacted]/measures
                    .csv.esp]]></property>
                <property name="fstype"><![CDATA[csv]]></property>
              </properties>
            </connector>
          </connectors>
        </window-source>
      </windows>
    </contquery>
  </contqueries>
</project>
```

A key factor in the project definition is represented by the list of data attributes used:

- **Growinginputfile**: Allows for real-time streaming of the input data file, which translates into "new" records being visible in SAS ESP Studio once the project is running;
- **Dateformat**: Specifies the date/datetime format of the date time field in the input data set;

- **Header**: Specifies the number of records to skip when the input data is first read. In this specific case, the first record is skipped given that it's a header;
- **Addcsvopcode**: Adds an "insert" op-code to the input record, allowing for its ingestion into ESP;
- **Addcsvflags**: Flags the input record as a "normal" insert, as opposed to an update of existing data;
- **Fsname**: Specifies the name of the input data file. With Viya 4, running on Kubernetes means the file location must be local to the pod where the project runs. Alternatively, this piece of information can be provided via the **dfesp_fs_adapter** command-line command. Please refer to the SAS documentation for its syntax and list of valid options.

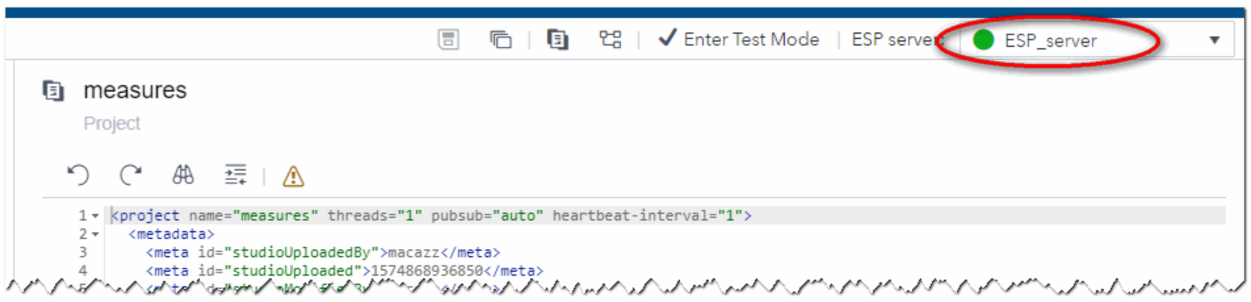
Another important consideration applies to the input data fields. When ingesting data, ESP Studio requires users to provide a unique key. If a natural one is not present, and data duplication is possible, consider specifying a multi-field key as in the scenario hereby described.

2. Data ingestion into CAS from ESP requires that a project be up and running on an active ESP server. The server is started via the **dfesp_xml_server** command-line command. Among the options that can be specified, "**-model**" allows for the full name of an existing XML model file to be specified. If this scenario applies to you and don't need SAS ESP Studio to create a model file, you have the option of starting the ESP server and have it run your project automatically once active. In this case, steps #3 and #4 do not apply and can be skipped.

With Viya 4, running an ESP server from the command line is not an option as the infrastructure is Kubernetes-based. Instead, when a project is executed via ESP Studio, an ESP server is launched automatically inside the auto-generated pod.

Viya 3.5 and earlier

3. If an ESP server definition already exists in SAS ESP Studio for an active ESP server, such definition should be associated with the project before running it:

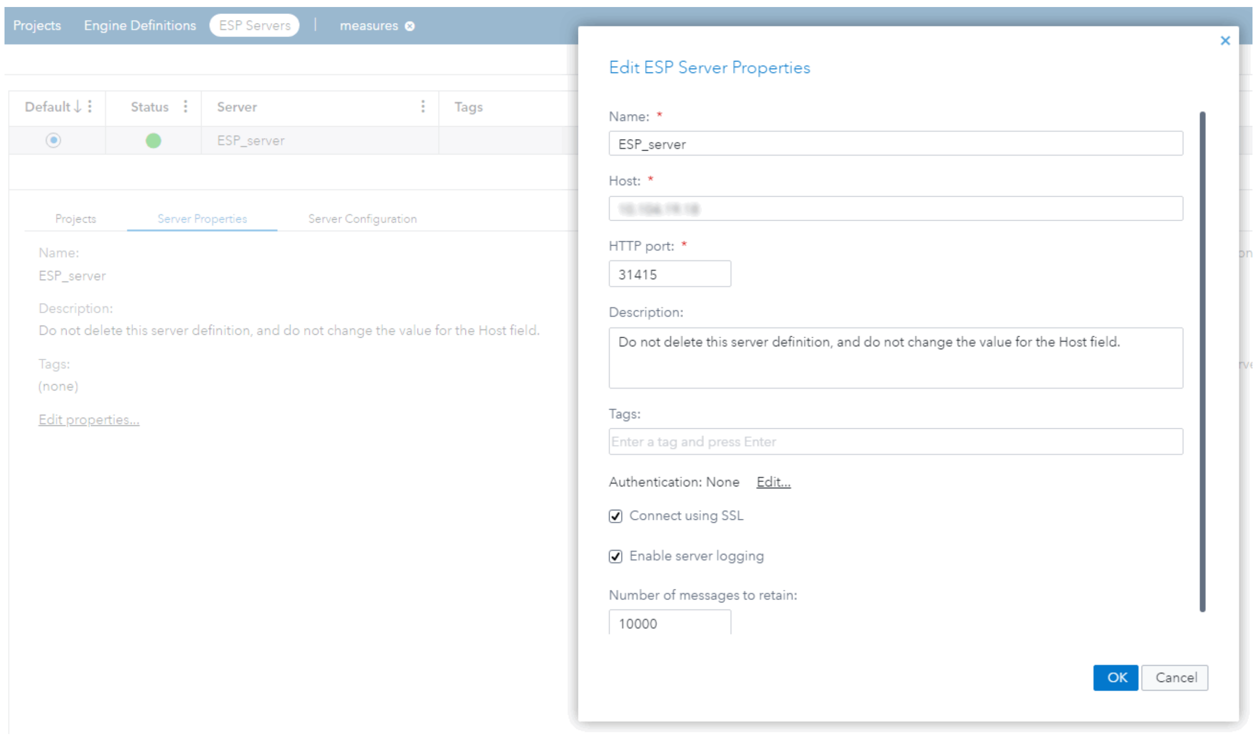


```
<project name="measures" threads="1" pubsub="auto" heartbeat-interval="1">
  <metadata>
    <meta id="studioUploadedBy">macazz</meta>
    <meta id="studioUploaded">1574868936850</meta>
```

If a server definition is not present, follow these steps:
- On the physical box where ESP is supposed to run, verify that a ESP server is active:

```
> ps -ef | grep xml_server
sas       3957     1 15 Nov25 pts/1    13:22:23 /opt/sas/viya/home/SASEventStreamProcessingEngine/current/bin/dfesp_xml_server -loglevel DF.ESP.STATS=trace -http 31415s -pubsub 31416
```

- As a side note, in this example the "**s**" after the HTTP port number indicates that secure communication (SSL) is being used for HTTP. Keep in mind that SSL is not a requirement. If needed, make sure to reference the SAS documentation for more information on how to secure communication for an ESP server. If no active server is found, start one using the **dfesp_xml_server** command-line command. Please refer to the SAS documentation for the command syntax and list of options.
- In SAS ESP Studio, add the definition of the server as shown in the example below. The HTTP port number must match the value used when the ESP server was started. The "**Connect using SSL**" option should be checked if secure communication was selected at start time:

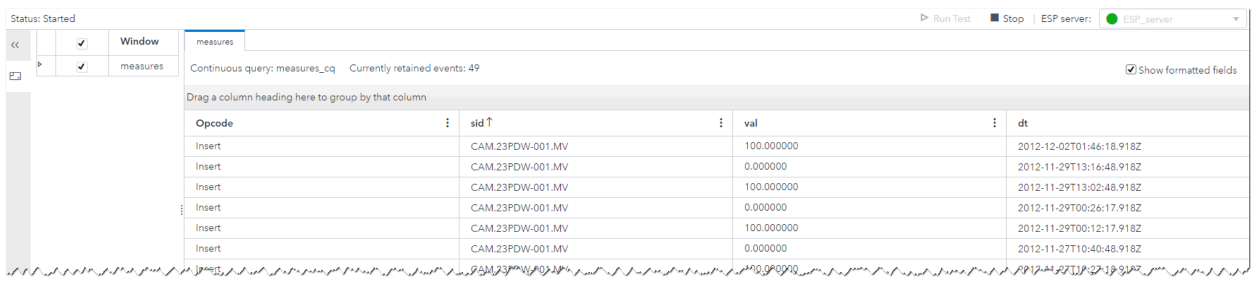


Name:
ESP_server
Description:
Do not delete this server definition, and do not change the value for the Host field.
Tags:
(none)

Edit ESP Server Properties

Name: *
ESP_server
Host: *
[redacted]
HTTP port: *
31415
Description:
Do not delete this server definition, and do not change the value for the Host field.
Tags:
Enter a tag and press Enter
Authentication: None Edit...
☑ Connect using SSL
☑ Enable server logging
Number of messages to retain:
10000

3. Once a project is associated with an active ESP server, it can finally be tested by selecting the ✓ Enter Test Mode option first, and then the ▷ Run Test one on the next screen. If there are no errors, a screen similar to the following should appear:



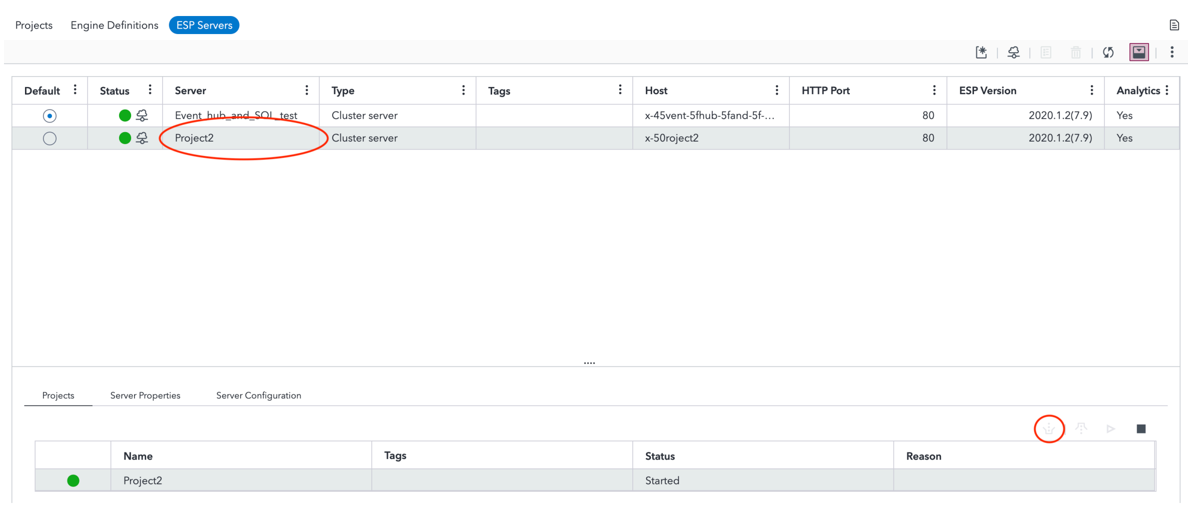
Continuous query: measures_cq Currently retained events: 49
☑ Show formatted fields
Drag a column heading here to group by that column

Opcode	sid ↑	val	dt
Insert	CAM.23PDW-001.MV	100.000000	2012-12-02T01:46:18.918Z
Insert	CAM.23PDW-001.MV	0.000000	2012-11-29T13:16:48.918Z
Insert	CAM.23PDW-001.MV	100.000000	2012-11-29T13:02:48.918Z
Insert	CAM.23PDW-001.MV	0.000000	2012-11-29T00:26:17.918Z
Insert	CAM.23PDW-001.MV	100.000000	2012-11-29T00:12:17.918Z
Insert	CAM.23PDW-001.MV	0.000000	2012-11-27T10:40:48.918Z

Running in test mode allows for a project to be debugged more easily by, for instance, verifying the data flow through its various windows. It's not, however, the only way to do it, as projects can be loaded and executed directly from the command line (as mentioned in #2, and assuming a pre-Viya 4 environment), or through SAS ESP Studio by:

Viya 3.5 and earlier

- Selecting the **ESP Servers** tab, highlighting an active server, then loading a project using the icon circled in red on the bottom-right side of the screen:

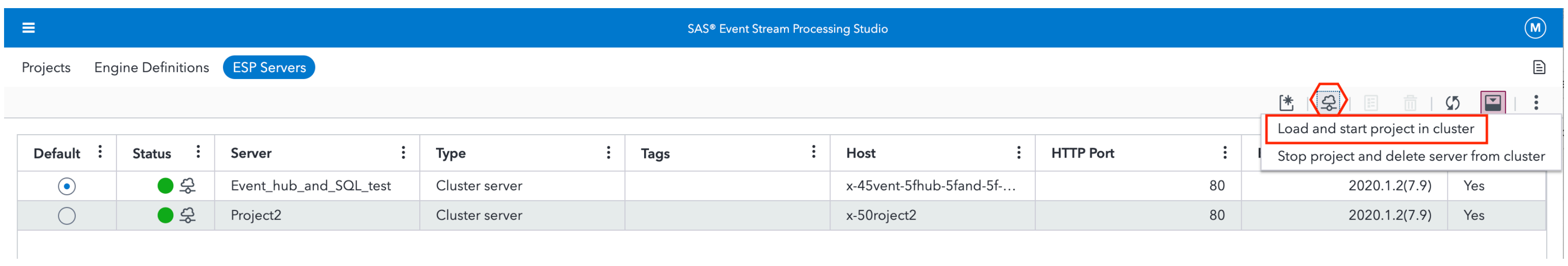


Default	Status	Server	Type	Tags	Host	HTTP Port	ESP Version	Analytics
◉	●	Event_hub_and_SQL_test	Cluster server		x-45vent-5fhub-5fand-5f-...	80	2020.1.2(7.9)	Yes
○	●	Project2	Cluster server		x-50roject2	80	2020.1.2(7.9)	Yes

	Name	Tags	Status	Reason
●	Project2		Started	

Viya 4

- Clicking on the **Cluster Options** icon on the top-right corner of the screen, and selecting the

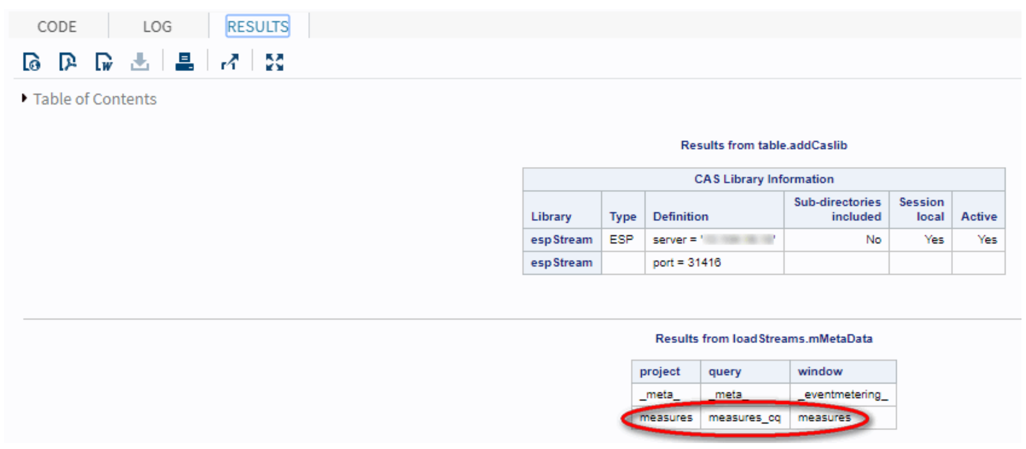

SAS® Event Stream Processing Studio
Projects
Engine Definitions
ESP Servers
Load and start project in cluster
Stop project and delete server from cluster

Default	Status	Server	Type	Tags	Host	HTTP Port		
◉	●	Event_hub_and_SQL_test	Cluster server		x-45vent-5fhub-5fand-5f-...	80	2020.1.2(7.9)	Yes
○	●	Project2	Cluster server		x-50roject2	80	2020.1.2(7.9)	Yes

4. Log on to SAS Studio and execute the following code:

```
cas mySession sessopts=(timeout=1800 locale="en_US");

/* Add caslib. Port number is the PUBSUB port */
proc cas;
     table.dropcaslib caslib="espStream" quiet=true;

     table.addCaslib
           dataSource={port=31416,server="[redacted]",srcType="esp"}
           name="espStream";

     loadStreams.mMetaData casLib="espStream";
     run;
quit;
```

As stated in the comment before Proc CAS, the "**port=**" option specifies the ESP pub/sub port number. The value must match the one used to start the server. "**EspStream**" is an arbitrary name that represents the CASLIB being defined. The **table.dropcaslib** and **loadStreams.mMetaData** action calls are optional. The former clears the definition of a CASLIB should one already exists with the same name of the one to be created, whereas the latter is used to provide details about any running projects on an ESP server. Those details, circled in red below, will be used next to access the data. From a programmatical standpoint, the **mMetaData** CAS action plays an important role as it can reveal whether a project is running on an ESP server, which is a precondition to enable the streaming of data from ESP to CAS. The generated output is similar to the following:



Results from table.addCaslib

CAS Library Information

Library	Type	Definition	Sub-directories included	Session local	Active
espStream	ESP	server = '[redacted]'	No	Yes	Yes
espStream		port = 31416			

Results from loadStreams.mMetaData

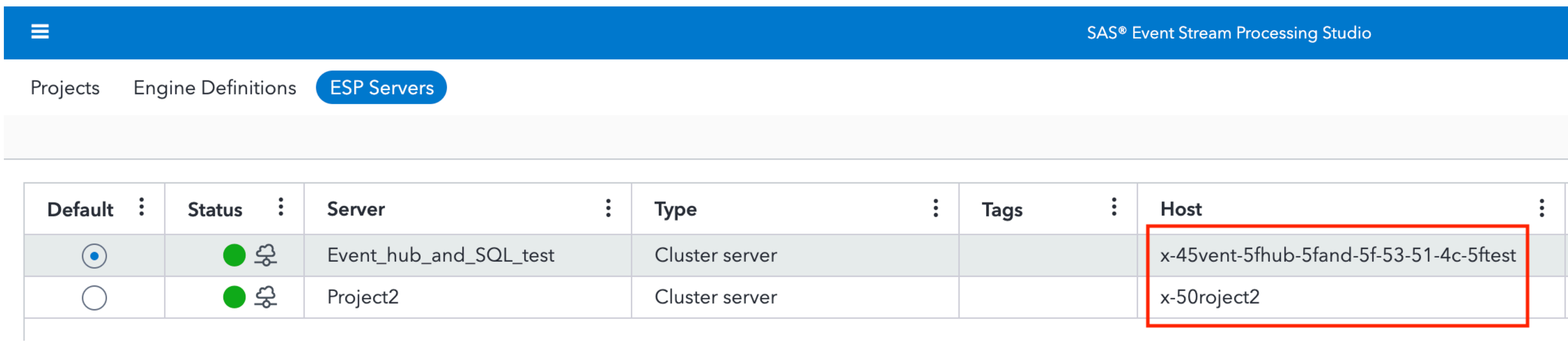
project	query	window
meta	_meta_	_eventmetering_
measures	measures_cq	measures

Viya 4

In Viya 4, **31416** is always the value for the port number. As for the **server=** caslib option, its value is the IP address of the Kubernetes pod where the project is running. To obtain it, once the project is running, a SAS code snippet similar to the one below can be used:

```
%macro find_esp_server(project);
       %let command = %bquote(")%str(nslookup )%str(&project)%str(-pubsub | grep Address | tail -1 | awk '{print $2}')%bquote(");

       filename shell pipe %unquote(&command);

       data _null_;
            length buffer $15.;
            infile shell dlm="¬";
            input buffer;
            call symputx('espserver', buffer, 'G');
       run;

       %put The ESP Server for the &project host is &espserver;
%mend find_esp_server;

%find_esp_server(project host name);
```

Where "**project host name**" is the name of the host on which the project is running as shown in SAS ESP Studio:



Default	Status	Server	Type	Tags	Host
●	🟩	Event_hub_and_SQL_test	Cluster server		x-45vent-5fhub-5fand-5f-53-51-4c-5ftest
○	🟩	Project2	Cluster server		x-50roject2

NOTE: The **nslookup** utility is not shipped with every flavor of Unix, in which case the macro code would fail. As an alternative, the **getent** command can be use as shown below:

```
%let command = %bquote(")%str(getent hosts )%str(&project)%str(-pubsub | awk '{print $1}')%bquote(");
```

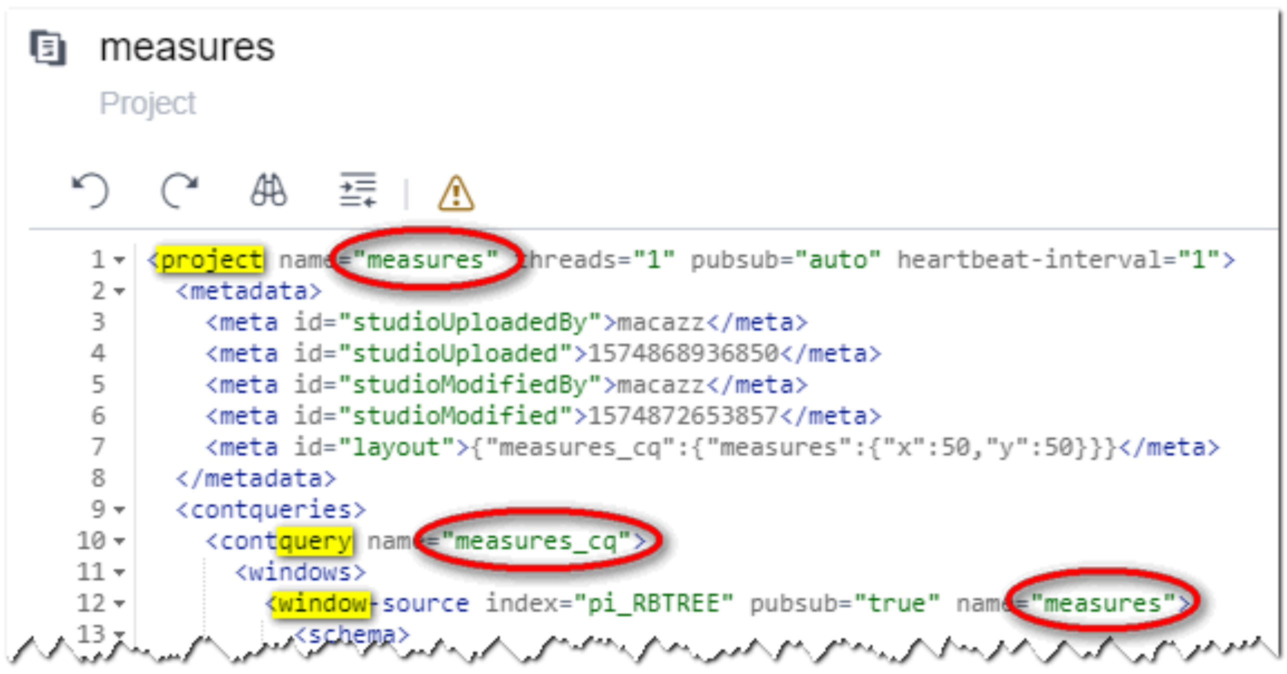
The execution of the code results in the "**espserver**" macro variable to be created and populated with the IP address of the pod running the project. The variable can then be used in the assignment of the ESP caslib:

```
proc cas;
      session "&_sessref_";

      table.dropcaslib caslib="espStream" quiet=true;

      table.addCaslib
            dataSource={port=31416,server="&espserver",srcType="esp"}
            name="espStream";

      loadStreams.mMetaData caslib="espStream";
      run;
quit;
```

5. Once a valid CASLIB has been issued, and once it's been verified that the metadata about our project is available, we can proceed loading the data into CAS. Three options are available:

   a. **Load data as a continuous stream into a CAS table**;
   b. **Load a snapshot of the data into a CAS table**;
   c. **Append a snapshot of the data to an existing CAS table.**

As mentioned earlier, the output from the **loadStreams.mMetaData** CAS action call is important because it is used to populate the **espUri=** option of the load command, regardless of the desired load type. The syntax for that option is "**<project>/<query>/<window>**" from the column names shown in the above generated output. Even though the same information can be obtained by looking at the XML definition of the project (as shown below), using the CAS action allows to determine whether our project is currently running, something we cannot infer by looking at its XML definition:



```
<project name="measures" threads="1" pubsub="auto" heartbeat-interval="1">
  <metadata>
    <meta id="studioUploadedBy">macazz</meta>
    <meta id="studioUploaded">1574868936850</meta>
    <meta id="studioModifiedBy">macazz</meta>
    <meta id="studioModified">1574872653857</meta>
    <meta id="layout">{"measures_cq":{"measures":{"x":50,"y":50}}}</meta>
  </metadata>
  <contqueries>
    <contquery name="measures_cq">
      <windows>
        <window-source index="pi_RBTREE" pubsub="true" name="measures">
          <schema>
```

The next section takes a closer look at each one of the available load options. As mentioned below, options like data filtering, table replacement, and table promotion are available regardless of the chosen data load technique (snapshot or continuous stream). Please refer to the SAS documentation for more information about the CAS actions used below.

a. **Load data as a continuous stream into a CAS table**

The following sample code loads a continuous stream of data into the CASUSER.MEASURES CAS table:

```
/* Load a continuos stream. Project must be running! */
proc cas;
     loadStreams.loadStream /
                 casLib="espStream"
                 espUri="measures/measures_cq/measures"
                 casOut={caslib="casuser",name="measures"};
     run;
quit;
```

There are some important factors to consider when loading data from a continuous stream. When the code shown above executes, control is never returned to the user until either of the following happens:

- The user cancels the submitted Proc CAS statement, which results in the execution of any other statements following the one in question to be halted. The input data is regularly added to the target table.
- A call to the **session.stopAction** CAS action is made to stop the session where the **loadStream** CAS action is running. As in the previous case, any other statements following the **loadStream** call are not executed. The input data is added to the target table, with the load process resembling a load from a snapshot rather than one from a continuous stream.

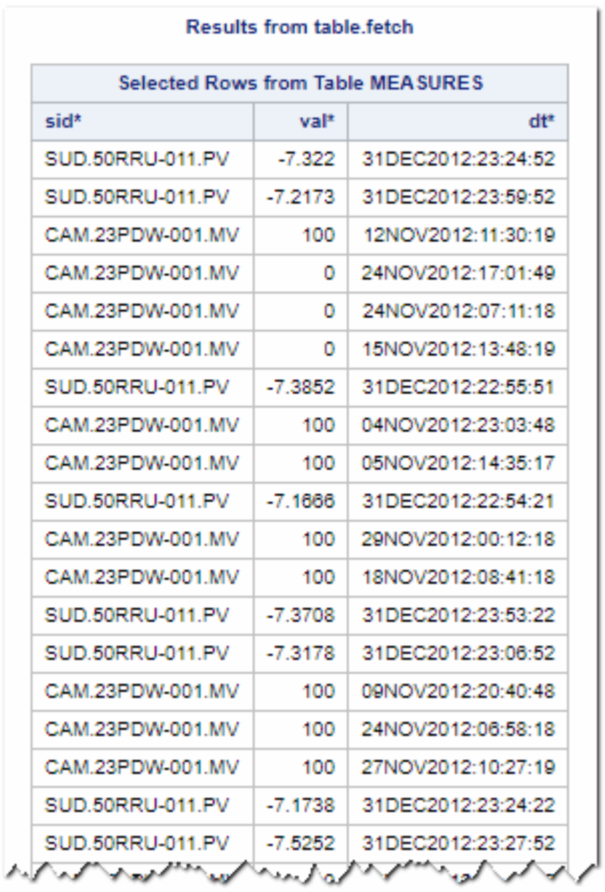
This kind of behavior is by design. Should this type be the preferred load method, it is highly recommended that the load run in batch mode or in a separate session.

b. **Load a snapshot of the data into a CAS table**

The following sample code loads a snapshot of the input data into the CASUSER.MEASURES CAS table:

```
proc cas;
     loadStreams.loadSnapshot /
                 casLib="espStream"
                 espUri="measures/measures_cq/measures"
                 casOut={caslib="casuser",name="measures"};

     table.fetch /
           table={caslib="casuser",name="measures",
           vars={'sid*','val*','dt*'}}
           index=false
           to=1000;
     run;
quit;
```

The **table.fetch** action call is optional, and it is used for verification purposes to make sure the data loads successfully. The generated output is shown below:



Results from table.fetch

Selected Rows from Table MEASURES

sid*	val*	dt*
SUD.50RRU-011.PV	-7.322	31DEC2012:23:24:52
SUD.50RRU-011.PV	-7.2173	31DEC2012:23:59:52
CAM.23PDW-001.MV	100	12NOV2012:11:30:19
CAM.23PDW-001.MV	0	24NOV2012:17:01:49
CAM.23PDW-001.MV	0	24NOV2012:07:11:18
CAM.23PDW-001.MV	0	15NOV2012:13:48:19
SUD.50RRU-011.PV	-7.3852	31DEC2012:22:55:51
CAM.23PDW-001.MV	100	04NOV2012:23:03:48
CAM.23PDW-001.MV	100	05NOV2012:14:35:17
SUD.50RRU-011.PV	-7.1666	31DEC2012:22:54:21
CAM.23PDW-001.MV	100	29NOV2012:00:12:18
CAM.23PDW-001.MV	100	18NOV2012:08:41:18
SUD.50RRU-011.PV	-7.3708	31DEC2012:23:53:22
SUD.50RRU-011.PV	-7.3178	31DEC2012:23:06:52
CAM.23PDW-001.MV	100	09NOV2012:20:40:48
CAM.23PDW-001.MV	100	24NOV2012:06:58:18
CAM.23PDW-001.MV	100	27NOV2012:10:27:19
SUD.50RRU-011.PV	-7.1738	31DEC2012:23:24:22
SUD.50RRU-011.PV	-7.5252	31DEC2012:23:27:52

Note how the **table.fetch** call suffixes each column name with a star. That is not by accident as every field that is part of the input data key is imported with an asterisk added to the end of its name.

Multiple calls to the **loadSnapshot** CAS action cause data to be appended to the target table by default. That behavior can be overridden through the "**replace=true**" option as shown below:

```
/* Load a snapshot. Project must be running! */
proc cas;
     loadStreams.loadSnapshot /
                 casLib="espStream"
                 espUri="measures/measures_cq/measures"
                 casOut={caslib="casuser",name="measures",replace=true};

     table.fetch /
           table={caslib="casuser",name="measures",
           vars={'sid*','val*','dt*'}}
           index=false
           to=1000;
     run;
quit;
```

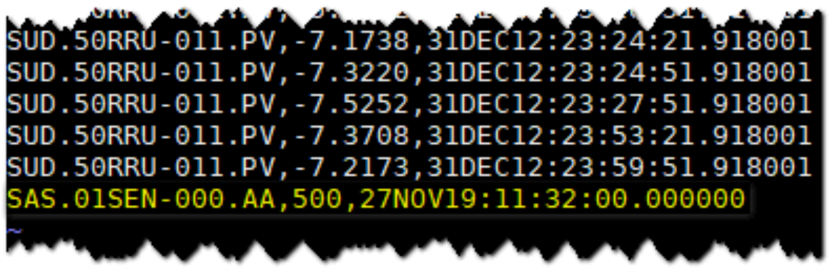
The **replace=** option is available to each of the three load methods. As in the previous example, filtering is also possible to avoid adding the same data repeatedly to the target table. An example of data filtering is shown next.

c. **Append a snapshot of the data to an existing CAS table**

The following sample code appends a filtered snapshot of the input data into the CASUSER.MEASURES CAS table:

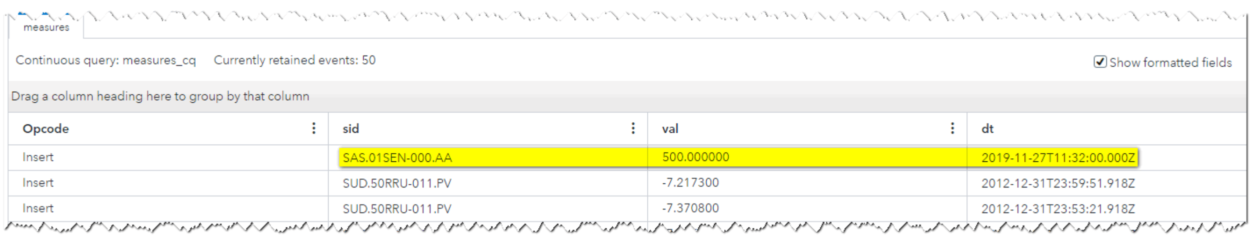
```
/* Append a snapshot. Project must be running! */
proc cas;
     loadStreams.appendSnapshot /
                 casLib="espStream"
                 espUri="measures/measures_cq/measures"
                 casOut={caslib="casuser",name="measures",replace=false,where="'dt*'n > '01Nov2019:08:00:00.000000'dt"};

     table.fetch /
           table={caslib="casuser",name="measures",
           vars={'sid*','val*','dt*'}}
           index=false
           to=1000;
     run;
quit;
```

To better understand how this type of load works, assume a new record is added to the input data:



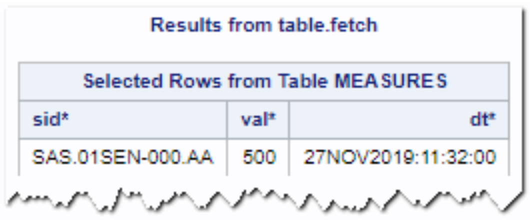
```
SUD.50RRU-011.PV,-7.1738,31DEC12:23:24:21.918001
SUD.50RRU-011.PV,-7.3220,31DEC12:23:24:51.918001
SUD.50RRU-011.PV,-7.5252,31DEC12:23:27:51.918001
SUD.50RRU-011.PV,-7.3708,31DEC12:23:53:21.918001
SUD.50RRU-011.PV,-7.2173,31DEC12:23:59:51.918001
SAS.01SEN-000.AA,500,27NOV19:11:32:00.000000
```

Switching to SAS ESP Studio, we can see how the new record has already been streamed in:


measures
Continuous query: measures_cq
Currently retained events: 50
☑ Show formatted fields
Drag a column heading here to group by that column

Opcode	sid	val	dt
Insert	SAS.01SEN-000.AA	500.000000	2019-11-27T11:32:00.000Z
Insert	SUD.50RRU-011.PV	-7.217300	2012-12-31T23:59:51.918Z
Insert	SUD.50RRU-011.PV	-7.370800	2012-12-31T23:53:21.918Z

After running the SAS code, the CASUSER.MEASURES CAS table shows the new record was loaded successfully:

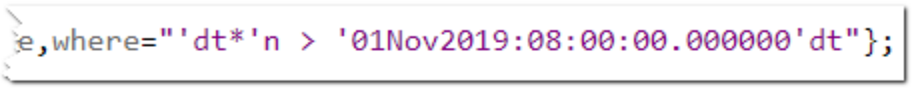


Results from table.fetch

Selected Rows from Table MEASURES

sid*	val*	dt*
SAS.01SEN-000.AA	500	27NOV2019:11:32:00

One important consideration applies to the filtering of the data. The filter condition specified through the "**where=**" option is based on pure SAS syntax. That means, for example, that comparisons with date time values require the "**d**", "**t**", or "**dt**" suffixes to be added to the end of the value. Furthermore, when a filter condition references a column that is part of the key for the input data, proper syntax must be used for the where clause can be processed successfully, given that the column is suffixed with an asterisk. Both situations can be seen at work in the following example:



```
e,where="'dt*'n > '01Nov2019:08:00:00.000000'dt"};
```

As mentioned earlier, data filtering through the "**where=**" option is available to each of the three load techniques mentioned in this document.

Should you have any questions or concerns, or if you simply want to provide feedback, feel free to contact me at mauro.cazzari@sas.com.
Thanks!