Paper 6640-2016

How to create a SAS Event Stream Processing model via the graphical user interface

Lei Xiao, Fang Meng, SAS Institute Inc., Beijing, China

ABSTRACT

SAS® Event Stream Processing is designed to quickly analyze and process large volumes of streaming data in motion. The new version of SAS Event Stream Processing introduces a browser-based user interface that enables you to create and test event stream processing models in the visual drag-and-drop environment. This environment delivers a highly interactive and intuitive user experience.

This paper describes the visual interactive interface for building models to monitor event stream activity. It provides examples that demonstrate how you can easily build a model using SAS Event Stream Processing Studio, the graphical user interface of SAS Event Stream Processing. In these examples, SAS Event Stream Processing serves as the front end to process high-velocity streams. On the back end, SAS® Real-Time Decision Manager consumes events and makes the final decision to push the suited offer to the customer. This paper explains the concepts of windows, retention policies, edges, and connectors. It also explains how SAS Event Stream Processing integrates with SAS Real-Time Decision Manager.

INTRODUCTION

The SAS Event Stream Processing provides an environment in which the users can create and execute a model. SAS Event Stream Processing 3.2 includes a web-based studio for building modular, continuous queries that use SAS advanced analytic algorithms and rules to determine event relevance. The model in SAS Event Stream Processing Studio is displayed as a data flow-centric diagram. This is designed so that engineers and application developers can see and control how windows relate and flow into one another. The logic and results can also be interactively validated and refined in the test model before the model is deployed.

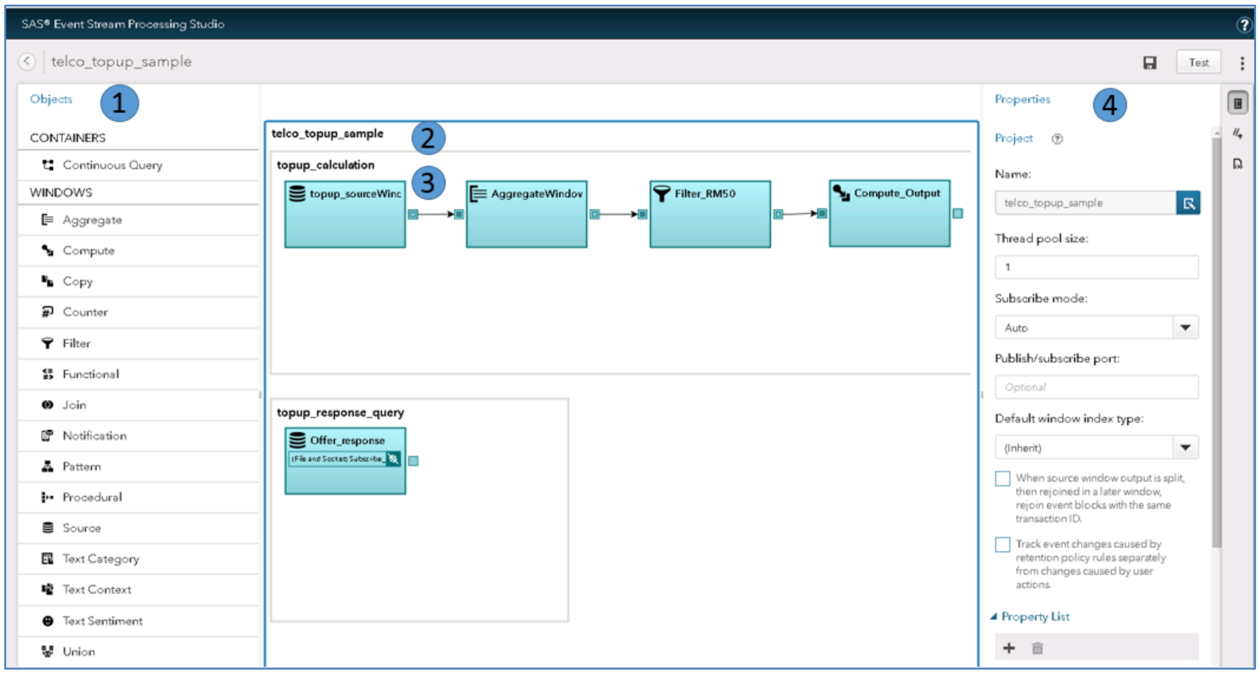
SAS Event Stream Processing provides adapters and connectors as part of its publish-and-subscribe architecture. With a suite of adapters and connectors, SAS Event Stream Processing can either be integrated into SAS solutions or become the front end for those solutions. The event data is first read through adapters and connectors and published into a defined window of an event stream processor. A down-stream application can subscribe to receive the results via adapters or connectors. This paper contains an example to explain how to build SAS Event Stream Processing models and integrate those models into other SAS solutions using adapters. This example uses SAS Event Stream Processing to monitor the live stream event data of amounts greater than $50 that have been spent by customers for prepay top up telecom plans. The SAS Real-Time Decision Manager in the back end consumes the results that have been passed through SAS Event Stream Processing, and sends a personalized message to customers.

SAS EVENT STREAM PROCESSING STUDIO

WORKSPACE

SAS Event Stream Processing contains a highly visual, interactive interface for building modular, continuous queries that use SAS advanced analytic algorithms and rules to determine event relevance. Within the user interface environment is the main workspace. Modeling objects appear on the left pane. The directed graphic is in the middle area. The properties of the selected modeling object appear in the right pane. By selecting and dragging a modeling object, a Continuous Query or Window can be added to the project. At the top of the SAS Event Stream Processing hierarchy is the engine. The engine contains one or more projects that run in a dedicated thread pool. Its size is defined in the project properties. The project contains one or more Continuous Queries.

The following figure shows an example workspace.



Name: telco_topup_sample

Thread pool size: 1

Subscribe mode: Auto

Publish/subscribe port: Optional

Default window index type: (Inherit)

☐ When source window output is split, then rejoined in a later window, rejoin event blocks with the same transaction ID.

☐ Track event changes caused by retention policy rules separately from changes caused by user actions.

Figure 1. Example Workspace

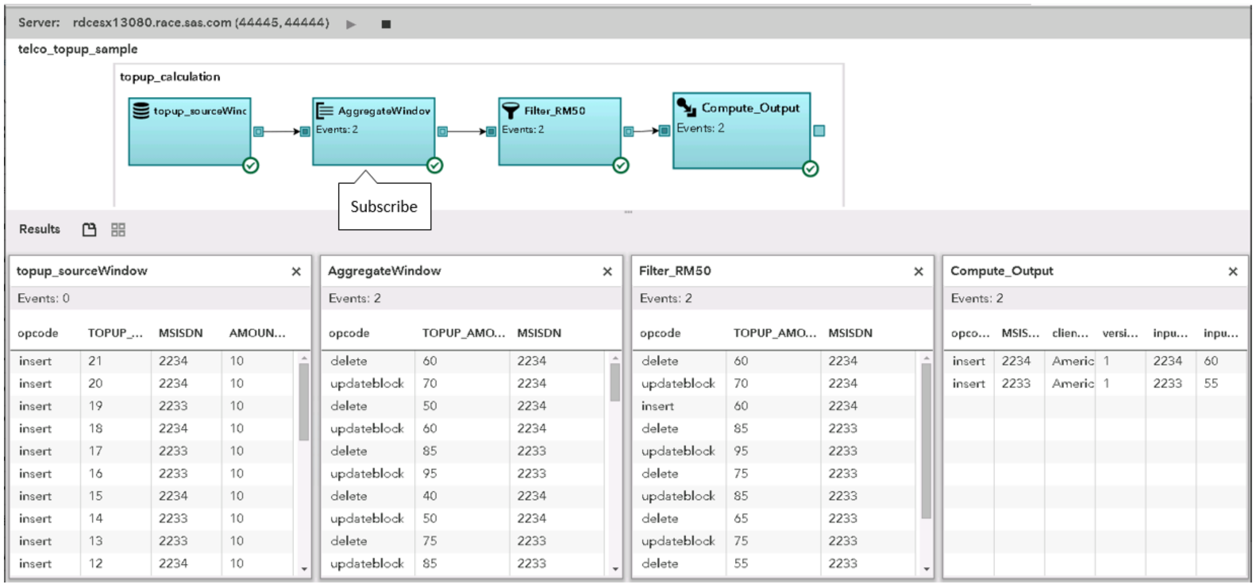
The workspace contains the following:

1. An Objects area that provides objects with different functions that can be inserted to manipulate or transform event data.
2. A Project, “telco_topup_sample”, that contains two continuous queries: “topup_calculation” and “topup_response_query”.
3. A Continuous Query, “topup_calculation”, that contains a directed graphic. Each window is connected by edges.
4. The Properties area, which enables you to set the properties for the object that you clicked in the studio area.

TEST MODE

SAS Event Stream Processing Studio contains an interactive test mode that enables users to evaluate the model’s logic and validate results before deployment. This test mode is very important because the model designer can quickly subscribe to the results in each window, run the model, and see how it runs. The model designer can then revise the model if anything is incorrect about the result during model execution.

An example of a model in test mode is shown in Figure 2 below.



topup_sourceWindow

Events: 0

opcode	TOPUP_...	MSISDN	AMOUN...
insert	21	2234	10
insert	20	2234	10
insert	19	2233	10
insert	18	2234	10
insert	17	2233	10
insert	16	2233	10
insert	15	2234	10
insert	14	2233	10
insert	13	2233	10
insert	12	2234	10

AggregateWindow

Events: 2

opcode	TOPUP_AMO...	MSISDN
delete	60	2234
updateblock	70	2234
delete	50	2234
updateblock	60	2234
delete	85	2233
updateblock	95	2233
delete	40	2234
updateblock	50	2234
delete	75	2233
updateblock	85	2233

Filter_RM50

Events: 2

opcode	TOPUP_AMO...	MSISDN
delete	60	2234
updateblock	70	2234
insert	60	2234
delete	85	2233
updateblock	95	2233
delete	75	2233
updateblock	85	2233
delete	65	2233
updateblock	75	2233
delete	55	2233

Compute_Output

Events: 2

opco...	MSIS...	clien...	versi...	inpu...	inpu...
insert	2234	Americ	1	2234	60
insert	2233	Americ	1	2233	55

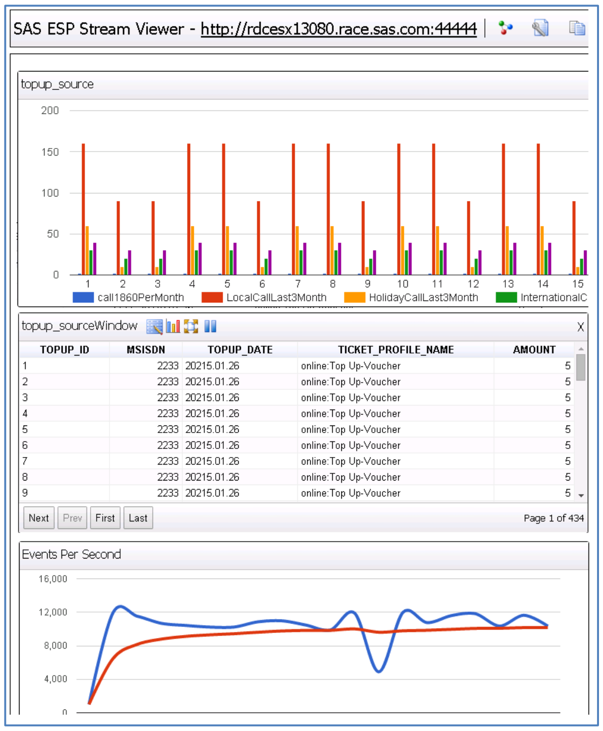
Figure 2. Test Mode

The project, “telco_topup_sample”, within the Continuous Query of “topup_calculation”, appears on the top pane of the test mode viewer. Each window in the project is shown. By clicking on each window and selecting “Subscribe”, each window’s event data can be displayed in the Results section below when you click the play icon to run the test.

EVENT STREAM VIEWER

The SAS Event Stream Processing Stream Viewer enables you to subscribe to a running event stream as it is processing a model. The stream processing detail can be monitored visually through the use of various charts or tables.

Figure 3 shows an example of the viewer.



TOPUP_ID	MSISDN	TOPUP_DATE	TICKET_PROFILE_NAME	AMOUNT
1	2233	20215.01.26	online:Top Up-Voucher	5
2	2233	20215.01.26	online:Top Up-Voucher	5
3	2233	20215.01.26	online:Top Up-Voucher	5
4	2233	20215.01.26	online:Top Up-Voucher	5
5	2233	20215.01.26	online:Top Up-Voucher	5
6	2233	20215.01.26	online:Top Up-Voucher	5
7	2233	20215.01.26	online:Top Up-Voucher	5
8	2233	20215.01.26	online:Top Up-Voucher	5
9	2233	20215.01.26	online:Top Up-Voucher	5

Page 1 of 434

Figure 3. SAS Event Stream Processing Viewer

The viewer contains a vertical bar chart, a table, and a line chart. The table represents the live stream detail of the “topup_sourceWindow” window, which can be seen by subscribing to the window. You can use the graphical representations to compare current processing with historical activity.

ADAPTERS

SAS Event Stream Processing Adapters are stand-alone executable files that use the Publish/Subscribe API to publish event streams into an engine or subscribe to event streams from engine windows. The pre-defined adapters enable you to read and write events from and to the following products: Hadoop, Database, OSIsoft PI, Axeda, Rabbit MQ, Solace, Tervela Data Fabric, XML/JSON File Socket Adapter, SAS® LASR™ Analytic Server, IBM DB2, IBM Netezza, IBM WebSphere MQ, SAP Sybase ASE, Tibco Rendezvous, JMS, File/Socket, Database ODBC, and SAS data sets.

The following examples explain how to use the File and Socket adapter to publish and subscribe to operations on files.

To use a publish adapter to inject the event’s file “topup.csv” into the source window “topup_sourceWindow” of the running project “telco_topup_sample”, execute the following command:

```
dfesp_fs_adapter -k pub -h
dfESP://rdcesx13080.race.sas.com:55555/telco_topup_sample/topup_calculation/t
opup_sourceWindow -t csv -b 256 -f topup.csv
```

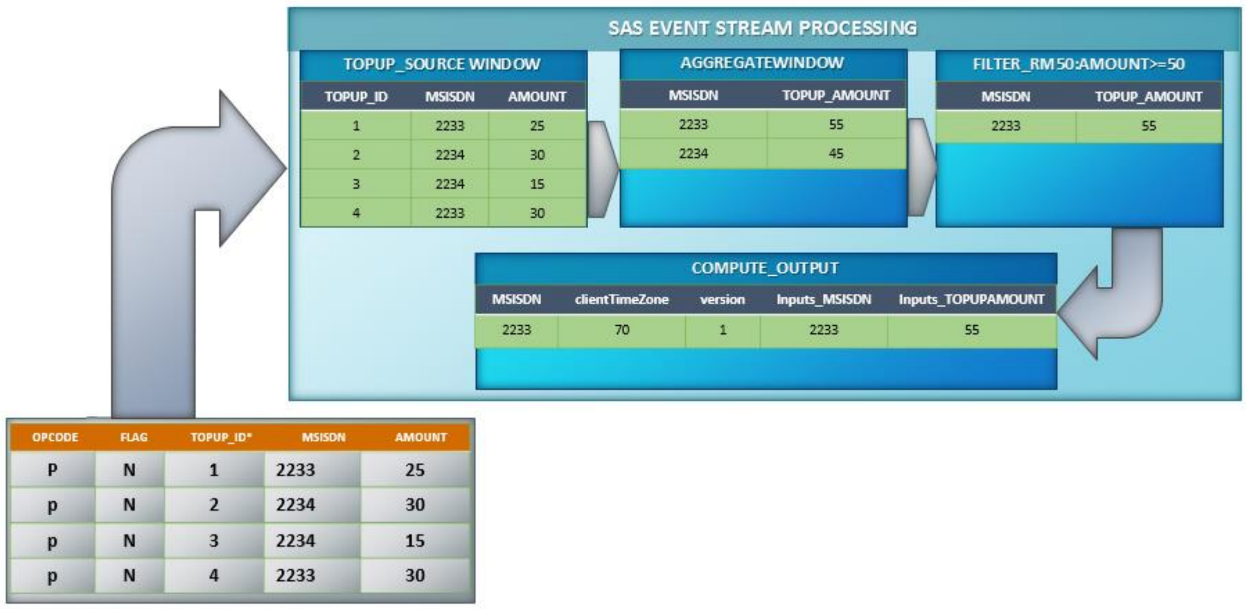
To use a subscribe adapter to grab events in the “RM50_OUTPUT” window and output these events to the file “RM50_OUTPUT.CSV”, execute the following command:

```
dfesp_fs_adapter -k sub -h
dfESP://rdcesx13080.race.sas.com:55555/telco_topup_sample/topup_calculation/C
ompute_Output?snapshot=true -t csv -f RM50_OUTPUT.CSV
```

EXAMPLE OF CREATING EVENT STREAM PROCESSING MODEL

For business purposes, we want to create a model to monitor prepay top-ups and accumulate the amount spent by a customer over a10-day period. The model provides output if the total amount spent is greater than $50. Thus, we need to design a SAS Event Stream Processing model that consists of following directed windows as shown in Figure 4:

1. A Source window that serves as the main entry of a model receiving data events.
2. A Copy window that uses a retention policy to remove events when they exceed 10 days.
3. An Aggregation window that calculates the total top-up amount, “TOPUP_AMOUNT”, by aggregating individual amounts by the MSISDN key.
4. A Filter window that only includes the “TOPUP_AMOUNT” when it is more than $50.
5. A Compute window that generates the output fields that will be passed to the downstream subscriber as required.



TOPUP_SOURCE WINDOW

TOPUP_ID	MSISDN	AMOUNT
1	2233	25
2	2234	30
3	2234	15
4	2233	30

AGGREGATEWINDOW

MSISDN	TOPUP_AMOUNT
2233	55
2234	45

FILTER_RM50:AMOUNT>=50

MSISDN	TOPUP_AMOUNT
2233	55

COMPUTE_OUTPUT

MSISDN	clientTimeZone	version	Inputs_MSISDN	Inputs_TOPUPAMOUNT
2233	70	1	2233	55

OPCODE	FLAG	TOPUP_ID*	MSISDN	AMOUNT
P	N	1	2233	25
p	N	2	2234	30
p	N	3	2234	15
p	N	4	2233	30

Figure 4. Example of Processing Event

The procedure for building the model in Figure 4 can be very simple if you use the drag and drop functionality from the set of available objects in SAS Event Stream Processing Studio. As shown in Figure 5, in the studio, the Project, Continuous Query, and Functional windows are all basic objects in the user interface.

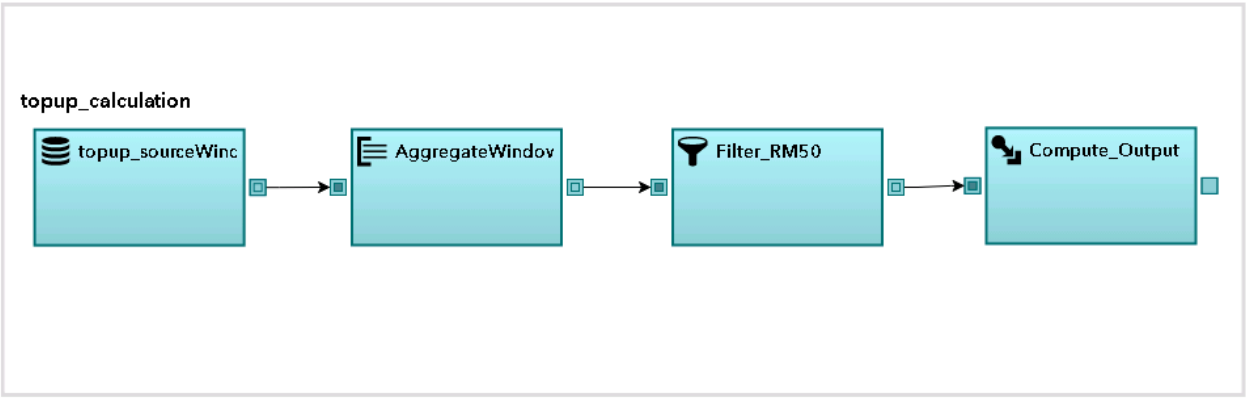


Figure 5. Example of Event Stream Processing Model

PROJECT

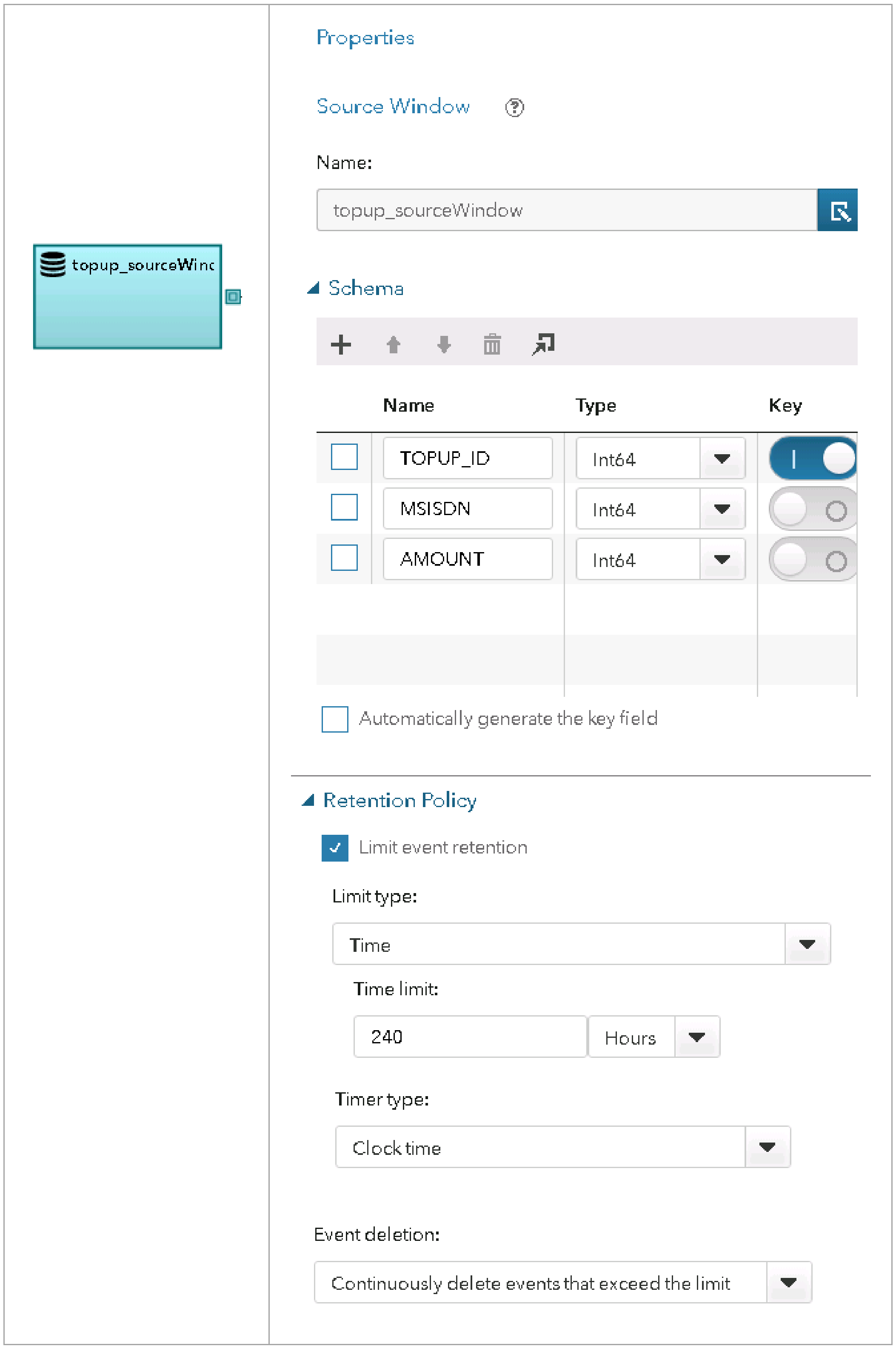
The container, “telco_topup_sample”, is the project in this example that holds the continuous query “topup_calculation”. The properties of the project appear in the right pane when the project is selected in the main workspace. You can modify the properties of a project to set its name, thread pool size, and subscribe mode.

CONTINUOUS QUERY

Within a continuous query, you can define a data flow model using all of the available window types. The “topup_calculation” is the name of the continuous query in this example. It contains a directed graphic of windows. The properties of the continuous query appear in the right pane when the query is selected in the main workspace. You can set its name and index type in the properties. To implement our example, the continuous query contains a source window, “topup_sourceWindow”, and four derived windows: “CopyWindow”, “AggregateWindow”, “Filter_RM50”, “Compute_Output”, as shown in Figure 5.

SOURCE WINDOW

The “topup_sourceWindow” source window is the main entry point of event data into the model. Event data must first be injected into the source window. By selecting the source window, you can view and modify the properties as shown in Figure 6. You can use these properties to set the corresponding schema for the incoming event data. You can also set a retention policy. In this example, the model needs to accumulate the amount spent by customer over 10 days, so you can specify a time-based retention policy for each event to 240 hours shown in Figure 6. Events will then be deleted continuously when they exceed the 10 day time limit.



Properties

Source Window

Name:

topup_sourceWindow

Schema

	Name	Type	Key
☐	TOPUP_ID	Int64	● On
☐	MSISDN	Int64	○ Off
☐	AMOUNT	Int64	○ Off

☐ Automatically generate the key field

Retention Policy

☑ Limit event retention

Limit type: Time

Time limit: 240 Hours

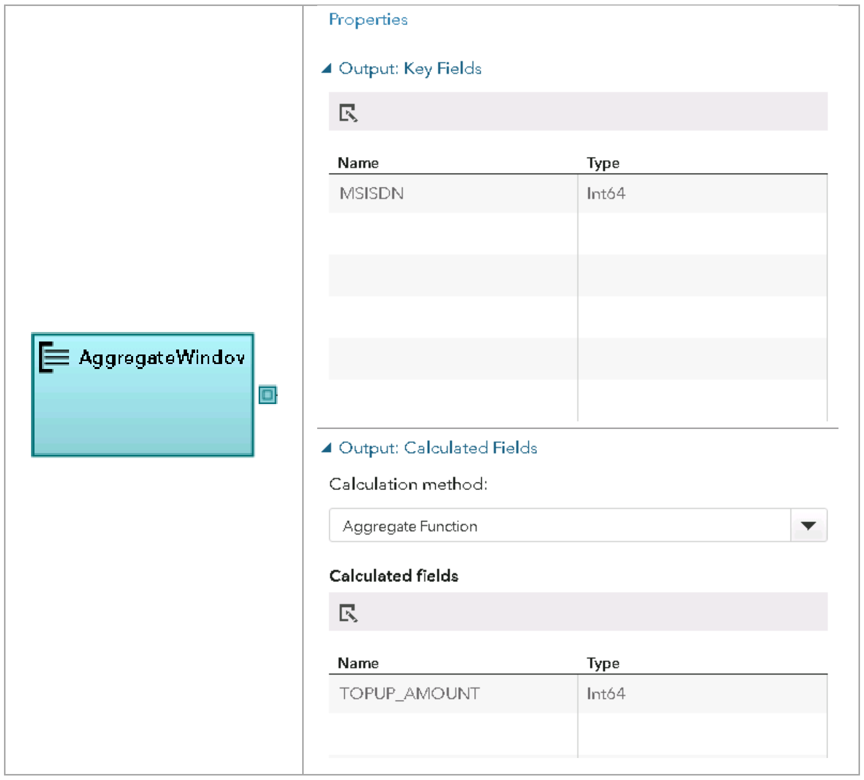
Timer type: Clock time

Event deletion: Continuously delete events that exceed the limit

Figure 6. Properties of the Source Window

AGGREGATE WINDOW

The “AggregateWindow” is an Aggregate window. It uses the key fields for the group-by condition to create the calculated fields as shown in the following figure:



Output: Key Fields

Name	Type
MSISDN	Int64

Output: Calculated Fields

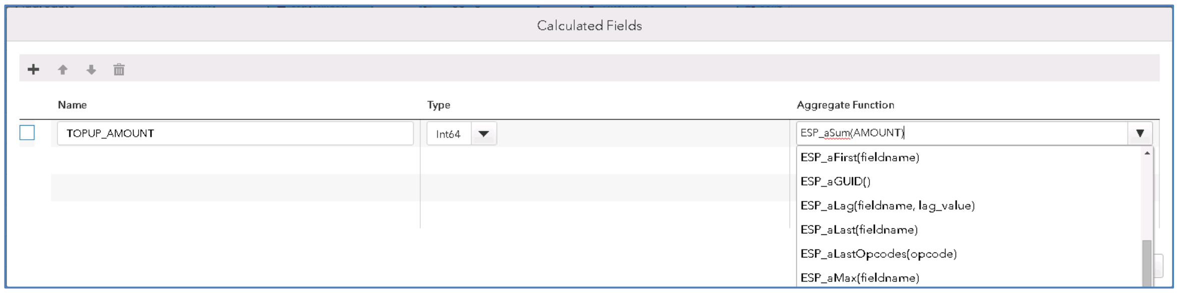
Calculation method: Aggregate Function

Calculated fields

Name	Type
TOPUP_AMOUNT	Int64

Figure 7. Properties of the Aggregate Window

In this example, the Aggregate Window accumulates the amount of top-ups, grouped by the key “MSISDN”, and generates a new calculated field “TOPUP_AMOUNT” by using aggregate function “ESP_aSum(AMOUNT)” is shown as Figure 8.

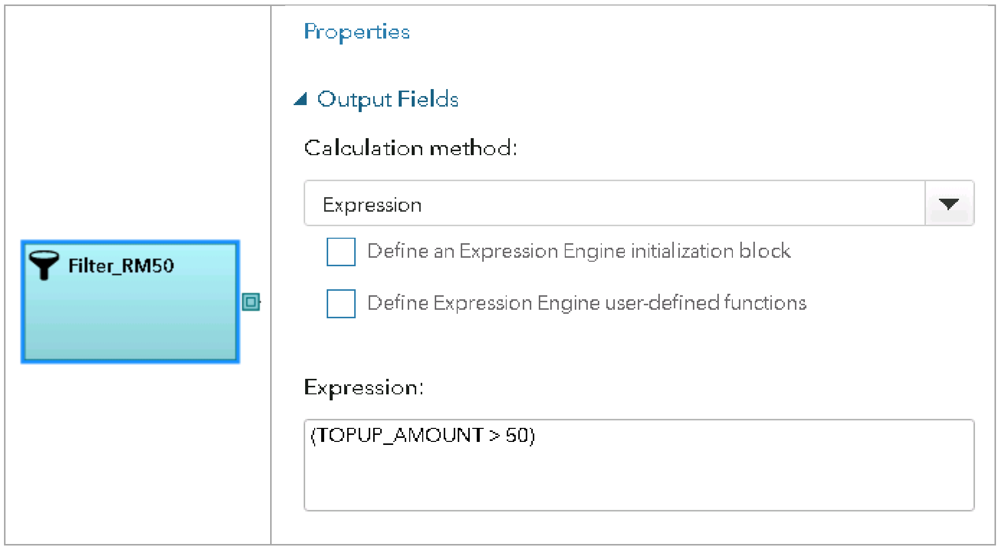


Name	Type	Aggregate Function
TOPUP_AMOUNT	Int64	ESP_aSum(AMOUNT)

Figure 8. Setting an Aggregate Function in Calculated Fields

FILTER WINDOW

The “Filter_RM50” is a Filter window. It contains a Boolean filter expression that determines which events are allowed into the Filter window. In this example, the filter is on the accumulated top-up value passed into it from the Aggregate window. If the aggregate amount, “TOPUP_AMOUNT“, is greater than or equal to $50, results are passed through this window as shown in Figure 9.



Properties

Output Fields

Calculation method:

Expression

- [ ] Define an Expression Engine initialization block
- [ ] Define Expression Engine user-defined functions

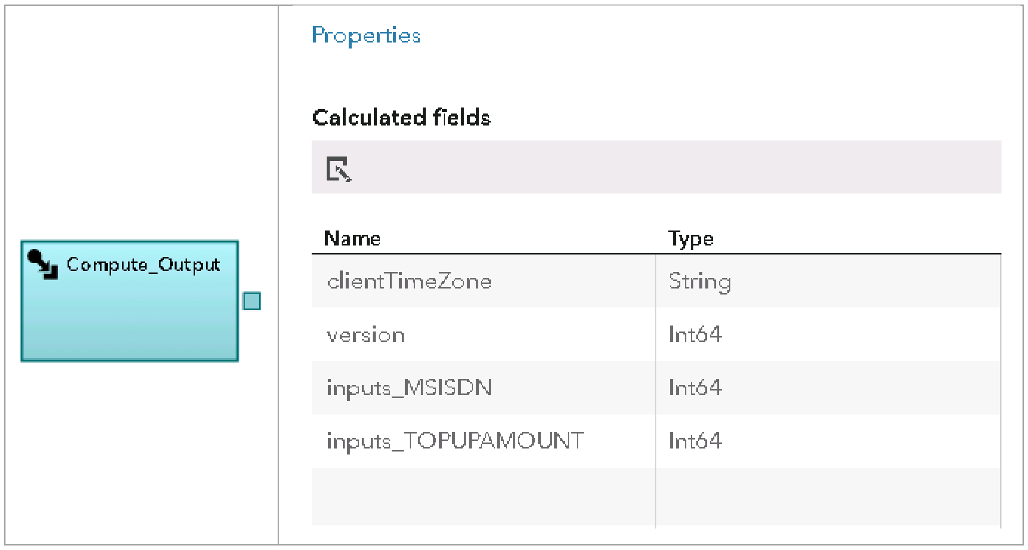
Expression:

```
(TOPUP_AMOUNT > 50)
```

Figure 9. Properties of the Filter Window

COMPUTE WINDOW

“Compute_Output” is a Compute window. It transforms input events into new output events through the computational manipulation of the event’s fields, as shown in Figure 10.



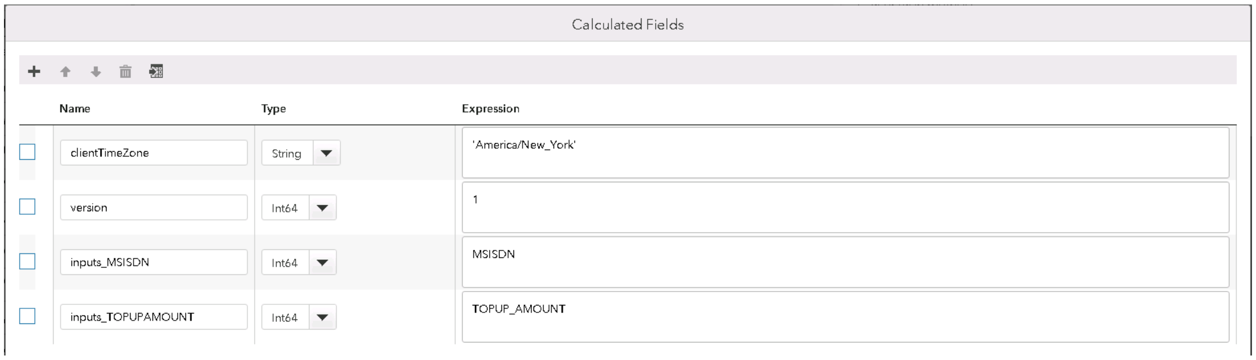
Properties

Calculated fields

Name	Type
clientTimeZone	String
version	Int64
inputs_MSISDN	Int64
inputs_TOPUPAMOUNT	Int64

Figure 10. Calculated Field Properties of the Compute Window

In this example, the Compute window creates several calculated fields through expressions that use the field names of input events to compute values for the derived event fields shown in Figure 12.


Calculated Fields

Name	Type	Expression
clientTimeZone	String	'America/New_York'
version	Int64	1
inputs_MSISDN	Int64	MSISDN
inputs_TOPUPAMOUNT	Int64	TOPUP_AMOUNT

Figure 11. Expressions in Calculated Fields of the Compute Window

EXAMPLE OF INTEGRATING WITH REST SERVICE VIA REST ADAPTERS

The publish-and-subscribe architecture makes SAS Event Stream Processing easy to integrate with other downstream systems. Downstream applications are able to subscribe to receive the stream processing results by using specific adapters. The following example shows how SAS Event Stream Processing can be integrated with SAS Real-Time Decision Manager through the use of a REST adapter.

SAS Real-Time Decision Manager is a SAS solution that enables you to capitalize on customer interactions. It uses SAS® Analytics to manage campaigns by using a combination of inbound business logic and contact strategies to deliver relevant offers in real time to interactive customer channels. The campaigns can be deployed as REST Services in operational environments using Real-Time Decision Manager's run-time servers.

In this example, the model in the SAS Event Stream Processing engine monitors events of interest for customers if the top-up total amount is more than $50 over 10 days.

In the back end, the SAS Real-Time Decision Manager subscribes to the events using a REST Subscriber Adapter. When events of interest are output from SAS Event Stream Processing, the subscribed adapter is notified and then forwards JSON through an HTTP POST to pass these qualified customers into a campaign in Real-Time Decision Manager. The campaign finds each customer's age, loads offer messages from a database, and pushes a personalized offer in a text message to each customer that is based on the customer's age. The HTTP response can be continually forwarded to an unrelated source window.

To successfully integrate with the downstream web service, the event fields in the output window must partially match the REST service and the source window receiving the response from web services, as shown in Figure 12.

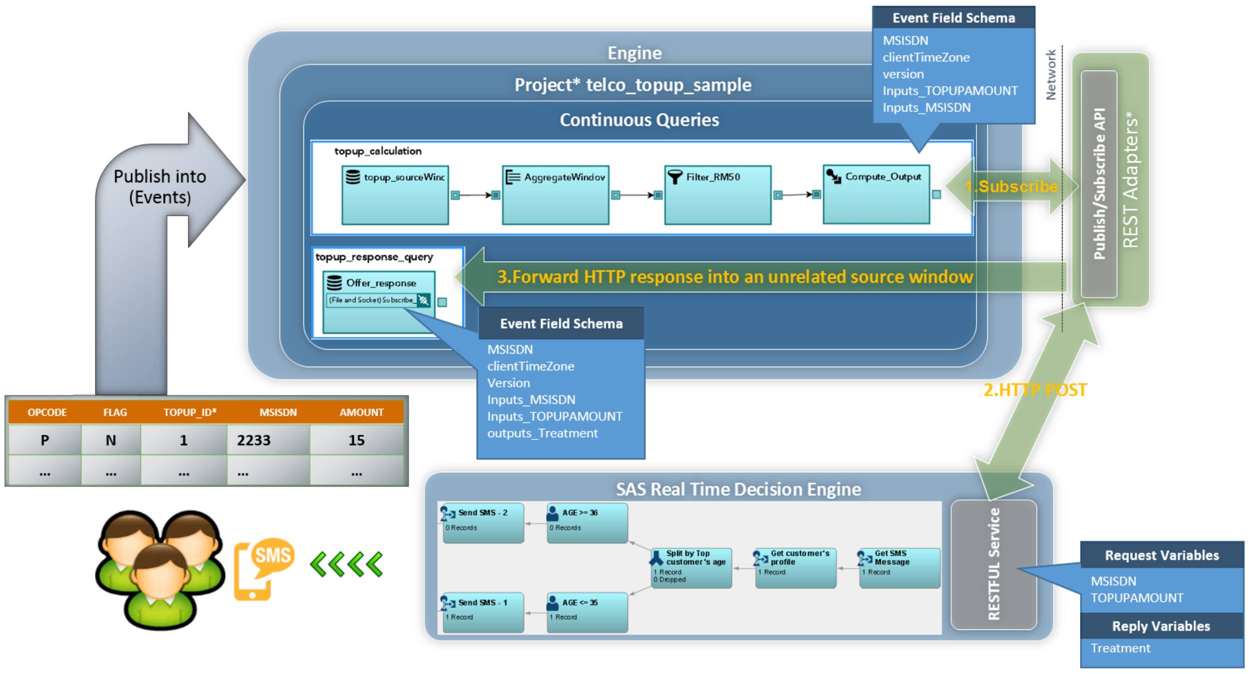


Figure 12. Overview of SAS Event Stream Processing's Integration with SAS Real-Time Decision Manager

EVENT

An Event is a data record that reflects something that has happened like a business event, and technically this is just a data record. The event object is stored in a format and contains many fields and keys. The first and second field represent the operation code (opcode) and a flag. The operation code enables updates, upserts, inserts, and deletes. The flag indicates whether the record has all the fields filled or just the updated ones (in the case of an Update operation code). In the following event shown in Figure 13, the 'p' = upsert and 'n' = normal.

OPCODE	FLAG	TOPUP_ID*	MSISDN	AMOUNT
p	n	1	2233	15

Figure 13. Schema of Event

DEFINING THE OUTPUT FIELD SCHEMA IN THE SUBSCRIBED WINDOW

The REST adapter formats a JSON string that uses all fields of the event and then forwards the JSON through an HTTP POST to the REST service. You must ensure that the schema of output fields in the subscribed window exactly match the fields defined in the REST service of SAS Real-Time Decision Manager. In this example, the Compute Window "Compute_Output" is subscribed to by the REST Adapter, so it generates the 4 calculated fields using the prefix "inputs_" followed by a name field, which is in compliance with the request variables of the SAS Real-time Decision Manager web service. In addition, two other calculated fields are required for the HTTP header element: "clientTimeZone" and "version", as shown in Table 1.

Request variables in REST Service	**Field schema in "Compute_Output" window**
MSISDN TOPUPAMOUNT	Inputs_MSISDN Inputs_TOPUPAMOUNT
clientTimeZone version	clientTimeZone version

Table 1. Field Schema in "RM50_OUTPUT" Subscribed Window and Request Variables in Web Service

DEFINING THE RESPONSE EVENT FIELDS IN A SEPARATE SOURCE WINDOW

The REST Adapter is able to receive the response from the REST Service. The adapter then invokes the Publish API to inject the response into the other Source Window. Thus, the beginning event fields in the Source Window must exactly match the complete schema of the Subscribed window. Also, the response fields must match the reply variable that is returned by the REST Service and follow the beginning fields. Each response field name must contain the prefix "outputs_". In this example, the REST service returns the variable of "Treatment". Thus, the matched field in the Source window "Offer_response" is "outputs_Treatment" shown in Table 2.

Field schema in "Compute_Output" window	**Field schema in "Offer_response" window**
MSISDN Inputs_TOPUPAMOUNT Inputs_MSISDN clientTimeZone version	MSISDN Inputs_TOPUPAMOUNT Inputs_MSISDN clientTimeZone version
Reply variables in REST Service	
Treatment	Outputs_Treatment

Table 2. Field Schema in "Offer_response" Source Window

USAGE OF THE REST SUBSCRIBE ADAPTER

The REST Subscriber Adapter is a stand-alone executable file. After deploying the model into the SAS Event Stream Processing engine, you can execute the command "dfesp_rest_subscriber" to lunch the adapter. An example is shown in Command 1:

The parameter "-u" specifies the dfESP publish and subscribe standard URL in the form of "dfESP://host:port/project/contquery/window".

The parameter "-r" specifies the URL of the target REST service.

The parameter "-t" specifies the value of the Content-Type string that is used in the HTTP post.

The parameter "-e" specifies the Publish and Subscribe standard URL to which responses should be published.

```
dfesp_rest_subscriber -u
"dfESP://ESPHost:55555/telco_topup_sample/topup_calculation/Compute_Output?s
napshot=true" -r "http://
RestHost.race.sas.com/RTDM/rest/runtime/decisions/SGF2016_Event" -t
"application/vnd.sas.decision.request+json" -e "dfESP://
ESPHost:55555/telco_topup_sample/topup_response_query/Offer_response"
```

Command 1. Example of dfesp_rest_subscriber Command

Once the “dfesp_rest_subscriber” command is launched:

1. The adapter keeps waiting until the qualified event reaches the subscribed window. In the preceding example, the subscribed window is “Compute_Output”.
2. The adapter generates the HTTP POST request when qualified events reach the subscribed window. As shown in “Sending request” Output 1, the adapter formats two JSON strings and forwards the JSON through an HTTP POST to the REST service when two events with qualified “TOPUPAMOUNT > 50” reach the subscribed window.
3. The adapter accepts the result from the REST service, and writes the response back to the other source window. As shown in “Server Request” in Output1, the REST service returns the “Treatment” field formatted in the JSON string to the adapter. The adapter then injects the JSON to the “Offer_response” source window that belongs to the other model.

```
 Sending request --->
{"MSISDN":2234,"clientTimeZone":"America/New_York","version":1,"inputs":{"MS
ISDN":2234,"TOPUPAMOUNT":60}}
Sending request --->
{"MSISDN":2233,"clientTimeZone":"America/New_York","version":1,"inputs":{"MS
ISDN":2233,"TOPUPAMOUNT":70}}
Server Response <----
{"version":1,"startTimestamp":"2015-11-18T22:45:44.780-
05:00","endTimestamp":"2015-11-18T22:45:45.359-
05:00","outputs":{"Treatment":"Hi Mary, Why not switch to a contract to get
unlimited talk."}}
Server Response <----
{"version":1,"startTimestamp":"2015-11-18T22:45:44.795-
05:00","endTimestamp":"2015-11-18T22:45:46.925-
05:00","outputs":{"Treatment":"Hi peter, Get one month of 8GB data plus
unlimited standard national talk and text for just $25. Hurry, this online
only offer ends 1 November 2015!"}}
```

Output 1. Output After Running the “dfesp_rest_subscriber” Statement

CONCLUSION

SAS Event Stream Processor is a powerful way to quickly process and analyze continuous events on a real-time basis. Its numerous adapters enable you to subscribe to and publish events from a wide variety of applications.

SAS Event Stream Processing Studio, its web-based visual interface, provides a set of comprehensive objects to build, test, model, and monitor streaming activity to and from these adapters, which improves the efficiency and effectiveness for designing models and processing events.

The preceding example details one way in which you can create models in SAS Event Stream Processing Studio and integrate the results of that processing with a downstream REST service via the REST adapter.

ACKNOWLEDGEMENTS

Thanks to Steve Sparano, Kansun Xia, Tao Zhang, Todd Wright, Frédéric Combaneyre, and Toshi Tsboi for perceptive review of this paper and valuable suggestion on ESP technique.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Name: Lei Xiao and Fang Meng
Enterprise: SAS Institute Inc.
Email: lei.xiao@sas.com and Fang.Meng@sas.com