

## Edge Streaming Creator

System Manual

<u>Overview</u>	<b>1</b>
<u>Using Edge Streaming Creator</u>	<b>2</b>
<u>Creating and Using Windows</u>	<b>3</b>
<u>Using Streaming Analytics</u>	<b>4</b>
<u>Accessibility Features</u>	<b>5</b>

## Legal information

### Warning notice system

This manual contains notices you have to observe in order to ensure your personal safety, as well as to prevent damage to property. The notices referring to your personal safety are highlighted in the manual by a safety alert symbol, notices referring only to property damage have no safety alert symbol. These notices shown below are graded according to the degree of danger.

#### **DANGER**

indicates that death or severe personal injury **will** result if proper precautions are not taken.

#### **WARNING**

indicates that death or severe personal injury **may** result if proper precautions are not taken.

#### **CAUTION**

indicates that minor personal injury can result if proper precautions are not taken.

#### **NOTICE**

indicates that property damage can result if proper precautions are not taken.

If more than one degree of danger is present, the warning notice representing the highest degree of danger will be used. A notice warning of injury to persons with a safety alert symbol may also include a warning relating to property damage.

### Qualified Personnel

The product/system described in this documentation may be operated only by **personnel qualified** for the specific task in accordance with the relevant documentation, in particular its warning notices and safety instructions. Qualified personnel are those who, based on their training and experience, are capable of identifying risks and avoiding potential hazards when working with these products/systems.

### Proper use of Siemens products

Note the following:

#### **WARNING**

Siemens products may only be used for the applications described in the catalog and in the relevant technical documentation. If products and components from other manufacturers are used, these must be recommended or approved by Siemens. Proper transport, storage, installation, assembly, commissioning, operation and maintenance are required to ensure that the products operate safely and without any problems. The permissible ambient conditions must be complied with. The information in the relevant documentation must be observed.

### Trademarks

All names identified by ® are registered trademarks of Siemens AG. The remaining trademarks in this publication may be trademarks whose use by third parties for their own purposes could violate the rights of the owner.

### Disclaimer of Liability

We have reviewed the contents of this publication to ensure consistency with the hardware and software described. Since variance cannot be precluded entirely, we cannot guarantee full consistency. However, the information in this publication is reviewed regularly and any necessary corrections are included in subsequent editions.

# Table of contents

<b>1</b>	<b>Overview.....</b>	<b>13</b>
1.1	Product Overview.....	13
1.2	Planning Your Event Stream Processing Application.....	13
1.3	What is an Event Stream Processing Model? .....	14
1.4	Understanding Events.....	15
1.4.1	What is an Event?.....	15
1.4.2	Data Types in Events.....	16
1.4.3	Converting CSV Events into Binary Code .....	16
1.5	Understanding Event Blocks.....	17
1.6	Implementing Engines .....	18
1.7	Understanding Projects.....	19
1.8	Understanding Continuous Queries.....	19
1.9	Understanding Windows .....	20
1.10	Streaming Events through a Continuous Query .....	21
1.10.1	Overview .....	21
1.10.2	Processing the First Event.....	23
1.10.3	Processing the Second Event.....	23
1.10.4	Processing the Third Event.....	24
1.10.5	Processing the Fourth Event .....	25
1.10.6	Processing the Fifth Event.....	26
1.10.7	Code for the Example .....	26
1.11	Developing a Streaming Analytics Application .....	26
<b>2</b>	<b>Using Edge Streaming Creator.....</b>	<b>29</b>
2.1	Edge Streaming Creator Overview .....	29
2.1.1	What is Edge Streaming Creator .....	29
2.1.2	Requirements for Solution Access and Use .....	29
2.2	Understanding the User Interface.....	29
2.2.1	Pages.....	29
2.2.2	Panes.....	30
2.2.3	Tiles.....	31
2.2.4	Windows .....	31
2.2.5	Toolbars .....	32
2.2.6	Sorting and Filtering.....	33
2.2.7	Edge Streaming Creator Modeler .....	33
2.3	Working with Projects .....	33
2.3.1	Overview .....	33
2.3.2	Create a Project .....	35
2.3.3	Upload a Project .....	37

2.3.4	Delete a Project.....	38
2.3.5	Download a Project.....	38
2.3.6	Project Metadata.....	38
2.4	Working with Engines.....	39
2.4.1	Engine Definition Overview.....	39
2.4.2	Create a New Engine Definition.....	41
2.4.3	Upload an Engine Definition.....	42
2.4.4	Download an Engine Definition.....	43
2.4.5	Delete an Engine Definition.....	43
2.5	Project and Engine Definition Locking.....	43
2.5.1	Overview to Locking.....	43
2.5.2	Project Locking.....	43
2.5.3	Engine Definition Locking.....	44
2.6	Using Edge Streaming Creator Modeler.....	45
2.6.1	Using Edge Streaming Creator Modeler.....	45
2.6.2	Configure a Model's Streaming Server.....	46
2.6.3	Add a Window.....	46
2.6.4	Connecting Windows.....	48
2.6.5	Edge Display Types.....	48
2.6.6	Edge Roles.....	49
2.6.7	Delete a Window or an Edge.....	50
2.6.8	Window Icons.....	50
2.6.9	Configure the Properties of a Window.....	51
2.7	Using XML Editor.....	52
2.7.1	Using the XML Editor.....	52
2.7.2	Using Editing Tools and Keyboard Shortcuts.....	53
2.7.3	Validation.....	54
2.7.4	Efficiency Tips.....	55
2.8	Continuous queries.....	55
2.8.1	Configuring Continuous Queries in Edge Streaming Creator.....	55
2.8.2	Configure the Properties of a Continuous Query.....	55
2.9	Testing Models in Edge Streaming Creator.....	56
2.9.1	Testing Models in Edge Streaming Creator.....	56
2.9.2	Running a Test.....	56
2.9.3	Viewing a Model's Test Logs in Edge Streaming Creator.....	60
2.10	Managing Streaming Servers in Edge Streaming Creator.....	61
2.10.1	Managing Streaming Servers in Edge Streaming Creator.....	61
2.10.2	Create a streaming Server.....	63
2.10.3	Edit a streaming Server's Properties.....	64
2.10.4	Delete a streaming Server.....	64
2.10.5	Refresh the Main Table of Streaming Servers on the Manage Streaming Servers Page.....	64
2.11	Publishing Project Versions.....	65
2.11.1	Publishing Project Versions.....	65
2.11.2	Publish a Version.....	65
2.11.3	View a Published Version.....	66
2.11.4	Revert to a Previous Version.....	67
2.11.5	Download a Version.....	67
2.12	Example: Processing Trades.....	68

2.12.1	Overview .....	68
2.12.2	Project Details .....	68
2.12.3	Example Steps .....	70
2.13	Example: Streaming Analytics with Scoring and Training .....	78
2.13.1	Overview .....	78
2.13.2	Project Details .....	78
2.13.3	Example Steps .....	79
2.14	Example: Geofence .....	85
2.14.1	Overview .....	85
2.14.2	Project Details .....	85
2.14.3	Example Steps .....	86
2.15	Example: Unifying Multiple Input Streams .....	96
2.15.1	Overview .....	96
2.15.2	Project Details .....	97
2.15.3	Example Steps .....	98
2.16	Example: Splitting Generated Events across Output Slots .....	104
2.16.1	Overview .....	104
2.16.2	Project Details .....	105
2.16.3	Example Steps .....	106
2.17	Example: Identifying Trading Patters in a Stock Market.....	113
2.17.1	Overview .....	113
2.17.2	Project Details .....	114
2.17.3	Example Steps .....	115
2.18	Example: Transitioning a Model from Stateful to Stateless .....	121
2.18.1	Transition a Model from Stateful to Stateless Overview .....	121
2.18.2	Project Details .....	122
2.18.3	Example Steps .....	122
2.19	Example: Transposing Data from an Aircraft.....	126
2.19.1	Transpose Aircraft Information Example Overview .....	126
2.19.2	Project Details .....	127
2.19.3	Wide Mode Example Steps .....	128
2.19.4	Long Mode Example Steps.....	134
2.20	Working with SAS Micro Analytic Service Modules in Edge Streaming Creator .....	139
2.20.1	Overview .....	139
2.20.2	Create a SAS Micro Analytic Service Module .....	139
2.20.3	Upload a SAS Micro Analytic Service Module .....	140
2.20.4	Delete a SAS Micro Analytic Service Module .....	141
2.21	Working with Input Handlers.....	141
2.21.1	Overview .....	141
2.21.2	Creating Input Handler Functions in Calculate Windows .....	141
2.21.3	Creating Input Handler Functions in Procedural Windows .....	142
<b>3</b>	<b>Creating and Using Windows .....</b>	<b>145</b>
3.1	Window Types .....	145
3.1.1	Window Types .....	145
3.2	Creating a Continuous Query with Windows and Edges.....	147
3.3	Using Expressions .....	152

3.3.1	Overview to Expressions .....	152
3.3.2	Understanding Data Type Mappings .....	153
3.3.3	Using Event Metadata in Expressions .....	154
3.3.4	Using Expression Language Global Functions .....	154
3.3.5	Using SAS Data Quality Functions .....	155
3.4	Using Source Windows .....	155
3.4.1	Overview to Source Windows .....	155
3.4.2	Defining Event Index Types .....	156
3.4.3	Retention Policies in Source Windows .....	156
3.4.4	Propagation of Insert-Only Processing .....	156
3.4.5	Automatically Generating Key Values .....	157
3.4.6	The Publisher Connector .....	157
3.4.7	Enabling Metering Windows .....	157
3.4.8	Using Singletons .....	158
3.5	Using Aggregate Windows.....	158
3.5.1	Overview to Aggregate Windows.....	158
3.5.2	Flow of Operations.....	159
3.5.3	Using Aggregate Functions.....	159
3.6	Using Calculate Windows .....	164
3.6.1	Overview to Calculate Windows .....	164
3.6.2	Migrating from a Procedural Window to a Calculate Window.....	164
3.7	Using Compute Windows.....	165
3.7.1	Overview to Compute Windows.....	165
3.7.2	Using Compute Functions.....	165
3.8	Using Copy Windows .....	165
3.8.1	Overview to Copy Windows .....	165
3.8.2	Retention Policies in Copy Windows .....	166
3.9	Using Counter Windows .....	166
3.10	Using Filter Windows .....	167
3.11	Using Functional Windows.....	167
3.11.1	Overview to Functional Windows.....	167
3.11.2	Using Event Loops.....	168
3.11.3	Understanding and Using Function Context.....	168
3.12	Using Geofence Windows.....	172
3.12.1	Overview to Geofence Windows.....	172
3.12.2	Geometries.....	173
3.12.3	Positions Window Schema .....	177
3.12.4	Output Schema .....	177
3.12.5	Mesh Index.....	178
3.12.6	Example .....	179
3.13	Using Join Windows.....	180
3.13.1	Overview to Join Windows.....	180
3.13.2	Understanding Streaming Joins.....	181
3.13.3	Creating Empty Index Joins .....	184
3.13.4	Examples of Join Windows .....	185
3.14	Using Model Reader Windows.....	186
3.14.1	Using Model Reader Windows.....	186

3.15	Using Model Supervisor Windows .....	187
3.15.1	Using Model Supervisor Windows .....	187
3.16	Using Notification Windows .....	188
3.16.1	Overview to Notification Windows .....	188
3.16.2	Notification Window Delivery Channels .....	189
3.16.3	Using the Function-Context Element .....	192
3.17	Using Object Tracking Windows .....	196
3.17.1	Overview to Object Tracking Windows .....	196
3.17.2	The Intersection Over Union (IOU) Method of Tracking- By-Detection .....	196
3.17.3	Input Schema for Object Tracking Windows .....	197
3.17.4	Output Schema for Object Tracking Windows .....	198
3.17.5	Reference .....	200
3.18	Using Pattern Windows .....	200
3.18.1	Overview of Pattern Windows .....	200
3.18.2	Creating Patterns .....	201
3.18.3	State Definitions For Operator Trees .....	204
3.18.4	Restrictions on Patterns .....	206
3.18.5	Using Stateless Pattern Windows .....	208
3.18.6	Enabling Pattern Compression .....	208
3.18.7	Enabling the Heartbeat Interval .....	208
3.18.8	Using Index Generation Functions .....	209
3.19	Using Procedural Windows .....	209
3.19.1	Overview .....	209
3.19.2	Using C++ Window Handlers .....	210
3.19.3	Using DATA Step Window Handlers .....	213
3.19.4	Converting DS2 Table Server Code to Run in SAS Micro Analytic Service Mode .....	215
3.20	Using Remove State Windows .....	217
3.21	Using Score Windows .....	219
3.22	Using Text Category Windows .....	220
3.23	Using Text Context Windows .....	220
3.24	Using Text Sentiment Windows .....	220
3.25	Using Text Topic Windows .....	222
3.25.1	Overview .....	222
3.25.2	Example of Text Topic Window .....	222
3.26	Using Train Windows .....	223
3.27	Using Transpose Windows .....	224
3.28	Using Union Windows .....	230
3.29	Understanding Retention .....	230
3.29.1	Introduction .....	230
3.30	Understanding Primary and Specialized Indexes .....	233
3.30.1	Overview .....	233
3.30.2	Fully Stateful Primary Indexes .....	234
3.30.3	Using pi_HLEVELDB and pi_HLEVELDB_NC Primary Indexes .....	235
3.30.4	Non-Stateful Primary Index .....	240
3.30.5	Using a Stateful Local Join Index to Resolve the State .....	241

3.31	Restrictions on a Window's Primary Index and Input Windows.....	246
3.31.1	Introduction .....	246
3.32	Understanding Design Patterns .....	249
3.32.1	Overview to Design Patterns .....	249
3.32.2	Design Pattern That Links a Stateless Model with a Stateful Model .....	249
3.32.3	Controlling Pattern Window Matches.....	251
3.32.4	Augmenting Incoming Events with Rolling Statistics .....	251
3.33	Advanced Window Operations.....	253
3.33.1	Writing Aggregate Functions to Embed in Applications.....	253
3.33.2	Writing Additive Aggregate Functions.....	255
3.33.3	Writing Non-Additive Aggregate Functions .....	257
3.33.4	Implementing Periodic (or Pulsed) Window Output .....	259
3.33.5	Splitting Generated Events across Output Slots.....	259
3.33.6	Marking Events as Partial-Update on Publish .....	264
3.33.7	Implementing Persist and Restore Operations .....	266
3.33.8	Gathering and Saving Latency Measurements.....	268
3.33.9	Enabling Finalized Callback.....	271
3.34	Functional Window and Notification Window Support Functions .....	272
3.34.1	Functions for Event Stream Processing .....	272
3.34.2	CONTQUERYNAME.....	272
3.34.3	ENGINEMETADATA.....	272
3.34.4	ESPCONFIGVALUE .....	272
3.34.5	EVENTCOUNTER .....	273
3.34.6	EVENTSPROCESSED .....	273
3.34.7	EVENTTIMESTAMP .....	273
3.34.8	INPUT .....	273
3.34.9	ISLASTEVENTINBLOCK.....	274
3.34.10	ISNOTRETENTION .....	274
3.34.11	ISRETENTION .....	274
3.34.12	OPCODE.....	274
3.34.13	OUTPUT .....	274
3.34.14	PROJECTNAME .....	275
3.34.15	General Functions.....	275
3.34.16	ABS.....	275
3.34.17	AND.....	275
3.34.18	BASE64DECODE .....	276
3.34.19	BASE64DECODEBINARY.....	277
3.34.20	BASE64ENCODE .....	277
3.34.21	BASE64ENCODEBINARY.....	277
3.34.22	BETWEEN .....	278
3.34.23	BOOLEAN.....	278
3.34.24	CEILING.....	279
3.34.25	COMPARE .....	279
3.34.26	CONCAT .....	280
3.34.27	CONCATDELIM.....	281
3.34.28	CONTAINS.....	281
3.34.29	DECREMENT .....	282
3.34.30	DIFF .....	282
3.34.31	EQUALS.....	283
3.34.32	FALSE.....	284
3.34.33	FLOOR.....	284

3.34.34	GT .....	285
3.34.35	GTE .....	285
3.34.36	GUID .....	286
3.34.37	INCREMENT .....	286
3.34.38	IF .....	286
3.34.39	IFNEXT .....	287
3.34.40	IN .....	288
3.34.41	INDEX .....	288
3.34.42	INDEXOF .....	289
3.34.43	INTEGER .....	289
3.34.44	INTERVAL .....	290
3.34.45	ISNULL .....	291
3.34.46	ISSET .....	291
3.34.47	JSON .....	292
3.34.48	LASTINDEXOF .....	292
3.34.49	LISTITEM .....	293
3.34.50	LISTSIZE .....	293
3.34.51	LONG .....	293
3.34.52	LT .....	294
3.34.53	LTE .....	295
3.34.54	MAPVALUE .....	296
3.34.55	MAPVALUES .....	297
3.34.56	MAX .....	297
3.34.57	MEAN .....	298
3.34.58	MIN .....	298
3.34.59	MOD .....	299
3.34.60	NEG .....	299
3.34.61	NORMALIZESPACE .....	300
3.34.62	NEQUALS .....	300
3.34.63	NOT .....	301
3.34.64	NUMBER .....	302
3.34.65	OR .....	302
3.34.66	OUTSTR .....	303
3.34.67	PRECISION .....	304
3.34.68	PRODUCT .....	304
3.34.69	QUOTIENT .....	304
3.34.70	RANDOM .....	305
3.34.71	RGX .....	305
3.34.72	RGXINDEX .....	305
3.34.73	RGXLASTTOKEN .....	306
3.34.74	RGXMATCH .....	306
3.34.75	RGXREPLACE .....	307
3.34.76	RGXREPLACEALL .....	307
3.34.77	RGXTOKEN .....	308
3.34.78	RGXV .....	308
3.34.79	ROUND .....	309
3.34.80	SETCONTAINS .....	310
3.34.81	STARTSWITH .....	310
3.34.82	STRING .....	311
3.34.83	STRINGLENGTH .....	311
3.34.84	STRIP .....	312
3.34.85	SUBSTRING .....	312

3.34.86	SUBSTRINGAFTER .....	312
3.34.87	SUBSTRINGBEFORE .....	313
3.34.88	SUM .....	313
3.34.89	SWITCH .....	313
3.34.90	SYSTEMMICRO .....	314
3.34.91	SYSTEMMILLI .....	314
3.34.92	TIMECURRENT .....	314
3.34.93	TIMEDAYOFMONTH .....	315
3.34.94	TIMEDAYOFWEEK .....	315
3.34.95	TIMEDAYOFYEAR .....	315
3.34.96	TIMEGMTTOLOCAL .....	316
3.34.97	TIMEGMTSTRING .....	316
3.34.98	TIMEHOUR .....	316
3.34.99	TIMEMICRO .....	317
3.34.100	TIMEMILLI .....	317
3.34.101	TIMEMINUTE .....	318
3.34.102	TIMEMINUTEOFDAY .....	318
3.34.103	TIMEPARSE .....	318
3.34.104	TIMESECOND .....	319
3.34.105	TIMESTAMP .....	319
3.34.106	TIMESTRING .....	319
3.34.107	TIMESTRING .....	320
3.34.108	TIMESECONDOFDAY .....	320
3.34.109	TIMETODAY .....	320
3.34.110	TIMEYEAR .....	320
3.34.111	TOLOWER .....	321
3.34.112	TOUPPER .....	321
3.34.113	TRANSLATE .....	321
3.34.114	TRUE .....	322
3.34.115	URLDECODE .....	322
3.34.116	URLENCODE .....	323
3.34.117	XPATH .....	323
<b>4</b>	<b>Using Streaming Analytics .....</b>	<b>325</b>
4.1	Overview .....	325
4.2	Streaming Analytics Window Types .....	325
4.2.1	Overview .....	325
4.2.2	Edge Roles .....	326
4.2.3	Determining Algorithm Availability and Properties .....	327
4.3	Using Score Windows .....	330
4.4	Using Train Windows .....	330
4.5	Using Calculate Windows .....	331
4.5.1	Overview to Calculate Windows .....	331
4.5.2	Migrating from a Procedural Window to a Calculate Window .....	331
4.6	Using Model Reader Windows .....	332
4.7	Using Model Supervisor Windows .....	333
4.8	Understanding Event Types .....	334
4.8.1	Data Events .....	334
4.8.2	Model Events .....	334

4.8.3	Request Events.....	335
4.8.4	Overview .....	335
4.8.5	List Requests .....	336
4.8.6	Send Requests .....	336
4.8.7	Remove Requests .....	336
4.8.8	Load Requests.....	336
4.8.9	Processing Loaded Analytic Store Files with GPUs .....	337
4.8.10	Reconfig Requests.....	338
4.8.11	Example .....	339
4.9	Using Online Models.....	340
4.9.1	Overview .....	340
4.9.2	Training and Scoring with K-means Clustering .....	343
4.9.3	Training and Scoring with DBSCAN Clustering.....	349
4.9.4	Training and Scoring with Streaming Linear Regression .....	353
4.9.5	Training and Scoring Streaming Logistic Regression.....	359
4.9.6	Training and Scoring with Support Vector Machines.....	366
4.9.7	Training and Scoring with t-Distributed Stochastic Neighbor Embedding .....	372
4.9.8	Calculating Streaming Summary Statistics for One Variable .....	378
4.9.9	Calculating Streaming Pearson's Correlation .....	382
4.9.10	Calculating Segmented Correlation .....	384
4.9.11	Calculating Short-Time Fourier Transforms.....	387
4.9.12	Streaming Distribution Fitting.....	393
4.9.13	Computing Fit Statistics for Scored Results .....	397
4.9.14	Computing Receiver Operating Characteristic (ROC) Information.....	400
4.9.15	Streaming Numerical Data to Create a Histogram .....	405
4.9.16	Streaming Text Tokenization .....	408
4.9.17	Streaming Text Vectorization.....	410
4.9.18	Processing Image Data.....	413
4.9.19	Computing the Moving Relative Range .....	416
4.9.20	Using Term Frequency — Inverse Document Frequency (TFIDF) .....	419
4.9.21	Lag Monitoring .....	422
4.9.22	Applying a Cepstrum Transform .....	426
4.9.23	Video Encoding.....	430
4.9.24	Subspace Tracking (SST).....	432
4.9.25	Converting Speech to Text .....	438
4.9.26	Change Detection .....	456
4.10	Using Recommender Systems .....	459
4.10.1	Overview .....	459
4.10.2	Offline Recommender Models .....	460
4.10.3	Specifying Model Parameters.....	464
4.10.4	Online Recommender Models .....	467
4.10.5	Training an Online Recommender Model.....	468
4.10.6	Scoring Data with Recommender Models .....	474
<b>5</b>	<b>Accessibility Features .....</b>	<b>477</b>
5.1	Accessibility Features of Edge Streaming Analytics.....	477
5.1.1	Overview .....	477
5.1.2	User Interface Layout.....	477
5.1.3	Keyboard Shortcuts .....	477



# Overview

## 1.1 Product Overview

Edge Streaming Analytics enables you to quickly process and analyze a large number of continuously flowing events. Events are delivered through event streams, which are high throughput, low latency data flows. You can write event stream processing applications in XML, Python, or C++. Event streams are published in applications that use the following:

- Connector classes or adapter executables
- The C, JAVA, or Python publish/subscribe APIs
- Edge Streaming Viewer (available in future version)
- Edge Streaming Creator

You can embed event stream processing engines with dedicated thread pools within new or existing applications. You can use the Edge Streaming Server to feed event stream processing engine definitions (called projects) into a streaming Server.

Event stream processing applications can perform real-time analytics on streams of events. Typical use cases for event stream processing include but are not limited to the following:

- Sensor data monitoring and management
- Operational systems monitoring and management
- Cyber security analytics
- Capital markets trading systems
- Fraud detection and prevention
- Personalized marketing

## 1.2 Planning Your Event Stream Processing Application

Conceptually, an event is something that happens at a determinable time that can be recorded as a collection of fields. As you plan your event stream processing application, answer the following questions:

- What specific event streams are published into an application, and with what protocol and format?
- What happens to the data? That is, how are event streams transformed and analyzed?
- What are the resulting event streams of interest? What applications subscribe to these event streams, and in what format and protocol?

Your answers to these questions enable you to plan the structure of your event stream processing model.

## 1.3 What is an Event Stream Processing Model?

An event stream processing model specifies how input event streams from publishers are transformed and analyzed into meaningful resulting event streams consumed by subscribers. The following figure depicts the model hierarchy.

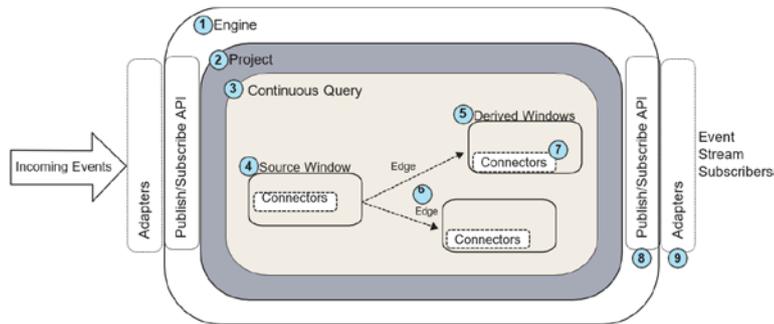


Figure 1-1 The Event Stream Processing Model Hierarchy

1. At the top of the model hierarchy is the *engine*. Each model contains only one engine instance with a unique name. The Streaming Server is an engine instance.
2. The engine contains one or more *projects*, each uniquely named. Projects run in a dedicated *thread pool* whose size is defined as a project attribute. You can specify a port so that projects can be spread across network interfaces for throughout scalability. Using a pool of threads in a project enables the event stream processing engine to use multiple processor cores for more efficient parallel processing.
3. A project contains one or more *continuous queries*. A continuous query is represented by a directed graph. This graph is a set of connected nodes that follow a direction down one or more parallel paths. Continuous queries are data flows, which are data transformations and analysis of incoming event streams.
4. Each query has a unique name and begins with one or more source *windows*.
5. Source windows are typically connected to one or more *derived windows*. Derived windows can detect patterns in the data, transform the data, aggregate the data, analyze the data, or perform computations based on the data. They can be connected to other derived windows.
6. Windows are connected by *edges*, which have an associated direction. In this context, edges are a program element that specifies connectivity between two or more windows.
7. *Connectors* publish or subscribe event streams to and from an engine. Connectors are in-process to the engine.
8. The *publish/subscribe API* can be used to subscribe to an event stream window either from the same machine or from another machine on the network. Similarly, the publish/subscribe API can be used to publish event streams into a running event stream processor project Source window.
9. *Adapters* are stand-alone executable programs that can be networked. Adapters use the publish/subscribe API to publish event streams to do the following:
  - Publish event streams to Source windows
  - Subscribe to event streams from any window

Several objects in the modeling layers measure time intervals in microseconds. The following intervals are measured in milliseconds:

- time-out period for patterns
- retention period in time-based retention
- pulse intervals for periodic window output

Most non-real-time operating systems have an interrupt granularity of approximately 10 milliseconds. Thus, specifying time intervals smaller than 10 milliseconds can lead to unpredictable results.

---

**Note**

In practice, the smallest value for these intervals should be 100 milliseconds. Larger values give more predictable results.

---

## 1.4 Understanding Events

### 1.4.1 What is an Event?

An event is an individual record of an event stream. It is the fundamental building block of event stream processing. An event consists of metadata and field data.

An event's metadata consists of the following:

- An operation code (opcode)
- A set of flags (indicating whether the event is a normal, partial-update, or a retention-generated event from retention policy management)
- A set of four microsecond timestamps that can be used for latency measurements

Table 1-1 Opcodes Supported by Edge Streaming Analytics

Opcode	Description
Delete (D)	Removes event data from a window
Insert (I)	Adds event data to a window
Update (U)	Changes event data in a window
Upsert (P)	Updates event data if the key field already exists. Otherwise, it adds event data to a window
Safe Delete (SD)	Removes event data from a window without generating an error if the event does not exist

One or more fields of an event must be designated as a primary key. Key fields enable the support of opcodes.

Data in an event object is stored in an internal format as described in the schema object. All key values are contiguous and packed at the front of the event. An event object maintains internal hash values based on the key with which it was built. In addition, there are functions

in the `dfESPEventcomp` namespace for a quick comparison of events that were created using the same underlying schema.

When publishing, when you do not know whether an event needs an Update or Insert opcode, use Upsert. The Source window where the event is injected determines whether it is handled as an Insert or an Update. The Source window then propagates the correct event and opcode to the next set of connected windows in the model or to subscribers

## 1.4.2 Data Types in Events

Events support the following data types:

- INT32
- INT64
- DOUBLE
- STRING
- DATE (granularity to the second)
- STAMP (granularity to the microsecond)
- MONEY (192-bit fixed decimal)
- BINARY (binary large object or blob)
- RUTF8STR (reference-counted string or rstring)
- ARRAY (32-bit integers, 64-bit integers, double)

With the base data types (INT32, INT64, DOUBLE, STRING, DATE, STAMP, and MONEY), data is stored inline in the event. Inline storage enables fast indexing and serialization.

The BINARY, RUTF8STR, and ARRAY data types are not stored inline. Instead, they are referenced in an event, which means that the event holds a pointer to the actual data. These data types cannot be used as key fields for an event. They are reference-counted at the object level. This enables the same object to be referenced in multiple events, which reduces memory usage and the time it takes to create a new object and copy the data.

For non-key fields, the RUTF8STR data type might be more economical than STRING. RUTF8STR can be passed in and out of all windows. It is internally referenced as a standard string whenever you use it. These characteristics can lead to considerable savings in memory. Remember that a STRING data type is stored inline in an event. When a 16K string is propagated through a chain of windows, a copy of the string is included in the events in each of the windows. When you use RUTF8STR instead, the first use creates a `dfESPrstring` object that holds the string. Each event contains an 8-byte pointer to that object.

## 1.4.3 Converting CSV Events into Binary Code

You can convert a file or stream of CSV events into a file or stream of binary events. This file or stream can be published into a project and processed at higher rates than the CSV file or stream.

For CSV conversion to binary, refer to the example application "csv2bin" under the **examples/cxx** directory of the Edge Streaming Analytics installation. The `readme.examples.txt` file in `$DFESP_HOME/examples` explains how to use this example in order to convert CSV files to event stream processor binary files. The example shows you how to perform the conversion in C++ using methods of the C client API. You can also convert using the Java or Python client API.

CSV conversion to binary is very CPU intensive, so it is recommended to convert files one time or convert streams at the source. In actual production applications, the data frequently arrives in some type of binary form and needs only reshuffling to be used in Edge Streaming Analytics. Otherwise, the data comes as text that must be converted to binary.

To properly represent string fields in an event, the corresponding CSV string field must follow these rules:

- When a string field includes leading or trailing white space, you must enclose the entire string field in double quotation marks
- When a string field includes the CSV delimiter character (which is ',' by default), you must enclose the entire string field in double quotation marks
- You must prefix literal double quotation mark (") characters in a string field with a leading escape character (\")
- You must prefix literal escape (\) characters in a string field with a leading escape character (\)
- The multi-byte "Byte-Order Mark" (BOM) sequence is unsupported. If you include it, it prevents proper parsing of a CSV string
- The new line (\n) sequence is not supported. CSV is a line-oriented format and new line is reserved as the line delimiter

## 1.5 Understanding Event Blocks

Event blocks contain zero or more binary events. Publish/subscribe clients send and receive event blocks to or from the engine. Because publish/subscribe operations carry overhead, working with event blocks that contain multiple events (for example, 512 events per event block) improves throughput performance with minimal impact on latency.

Table 1-2 Event Block Types

Event Block	Description
Transactional	Processing through the project is atomic. If one event in the event block fails (for example, deleting a non-existing event), then all of the events in the event block fail. Events that fail are logged and placed in an optional bad records file, which can be processed further.
Normal	Processing through the project is not atomic. Events are packaged together for efficiency but are individually treated once they are injected into a source window.

A unique transaction ID is propagated through transformed event blocks as the blocks work their way through an engine model. This persistence enables event stream subscribers to

correlate a group of subscribed events back to a specific group of published events through the ID.

## 1.6 Implementing Engines

An engine is the top level container in the event stream processing model hierarchy. Each model contains only one engine instance with a unique name. Engines can be instantiated as stand-alone executables or embedded within an application.

Edge Streaming Analytics provides two ways to implement engines:

- The Streaming Server enables you to define single engine definitions and to define an engine with dynamic project creations and deletions. For more information, see **Managing Streaming Servers in Edge Streaming Creator**.
- The C++ Modeling API enables you to embed an event stream processing engine inside an application process space. It also provides low-level functions that enable an application's main thread to interact directly with the engine. For more information, open `$DFESP_HOME/doc/html/index.html` (UNIX deployments) or `%DFESP_HOME%\doc\html\index.html` (Windows deployments) in a web browser. This provides access to the complete class and method documentation.

Deciding whether to implement multiple projects or multiple continuous queries depends on your processing needs. For the Streaming Server, multiple projects can be dynamically introduced, destroyed, stopped, or started because the layer is being used as a service. For all modeling layers, multiple projects can be used for different use cases or to obtain different threading models in a single engine instance. You can use:

- A single-threaded model for a higher level of determinism
- A multi-threaded model for a higher level of parallelism

Because you can use continuous queries as a mechanism of modularity, the number of queries that you implement depends on how compartmentalized your windows are. Within a continuous query, you can instantiate and define as many windows as you need. Any given window can flow data to one or more windows.

Loop-back conditions are not permitted within continuous queries. You can loop back across continuous queries using the project connector or adapter.

Event streams must be published or injected into Source windows through one of the following:

- The publish/subscribe API
- Connectors
- Adapters
- HTTP clients
- Edge Streaming Creator
- Edge Streaming Viewer
- The continuous-query-inject method in the C++ Modeling API

Within a continuous query, you can define a data flow model using all the available window types. Procedural windows enable you to write event stream input handlers using C++.

## 1.7 Understanding Projects

A project specifies a container that holds one or more continuous queries and is backed by a thread pool of user-defined size. The level of determinism for a project's incremental computations is, by default, full concurrency. You can change this with the use-tagged-token attribute of the project element, which enables a project to use tagged token data flow semantics. You can also specify an optional port for publish/subscribe scalability.

The data flow model is always computationally deterministic. When a project is multi-threaded, intermediate calculations can occur at different times across different project runs. Therefore, when a project watches every incremental computation, the increments could vary across runs even though the unification of the incremental computation is always the same.

---

### Note

Regardless of the determinism level or of the number of threads used in the engine, each window always processes all data in order. Therefore, data received by a window is never rearranged and processed out of order.

---

## 1.8 Understanding Continuous Queries

A continuous query specifies a container that holds one or more directed graphs of windows and that enables you to specify the connectivity between windows. The windows within a continuous query can transform or analyze data, detect patterns, or perform computations. Query containers provide functional modularity for large projects. Typically, each container holds a single directed graph.

Continuous query processing follows these steps:

1. An event block (with or without atomic properties) that contains one or more events is injected into a Source window.
2. The event block flows to any derived window that is directly connected to the Source window. If transactional properties are set, then the event block of one or more events is handled atomically as it makes its way to each connected derived window. That is, all events must be performed in their entirety.

If any event in the event block with transactional properties fails, then all of the events in the event block fail. Failed events are logged. They are written to a bad records file for you to review, fix, and republish when you enable this feature.

3. Derived windows transform events into zero or more new events that are based on the properties of each derived window. After new events are computed by derived windows,

they flow farther down the model to the next level of connected derived windows, where new events are potentially computed.

4. This process ends for each active path down the model for a given event block when either of the following occurs:
  - There are no more connected derived windows to which generated events can be passed.
  - A derived window along the path has produced zero resulting events for that event block. Therefore, it has nothing to pass to the next set of connected derived windows.

## 1.9 Understanding Windows

A continuous query contains a Source window and one or more derived windows. Windows are connected by edges, which have an associated direction.

Edge Streaming Analytics supports the following window types:

Table 1- 3 Window Types

Window Type	Description
Source window	All event streams must enter continuous queries by being published or injected into a source window. Event streams cannot be published or injected into any other window type.
Compute window	Enables a one-to-one transformation of input events into output events through the computational manipulation of the input event stream fields.
Aggregate window	Similar to a Compute window in that non-key fields are computed. An Aggregate window uses the key field or fields for the group-by condition. All unique key field combinations form their own group within the Aggregate window. All events with the same key combination are part of the same group.
Copy window	Makes a copy of the parent window. Making a copy can be useful to set new event state retention policies. Retention policies can be set only in source and Copy windows.  You can set event state retention for a Copy window only when the window is not specified to be Insert-only and when the window index is not set to pi_EMPTY. All subsequent sibling windows are affected by retention management. Events are deleted when they exceed the windows retention policy.
Counter window	Enables you to see how many events are streaming through your model and the rate at which they are being processed.
Filter window	Uses a registered Boolean filter function or expression. This function or expression determines what input events are allowed into the Filter window.
Functional window	Enables you to use different types of functions to manipulate or transform the data in events. Fields in a Functional window can be hierarchical, which can be useful for applications such as web analytics.
Geofence window	Enables you to create a window to determine whether the location of an event stream is inside or near an area of interest.
Join window	Takes two input windows and a join type. Supports equijoins that are one to many, many to one, or many to many. Both inner and outer joins are supported.

Window Type	Description
Notification window	Enables you to send notifications through email, text, or multimedia message. You can create any number of delivery channels to send the notifications. A Notification window uses the same underlying language and functions as the Functional window.
Object Tracking window	Enables you to perform multi-object tracking (MOT) in real time.
Pattern window	Enables the detection of events of interest (EOI). A pattern defined in this window type is an expression that logically connects declared events of interest.  To define a pattern window, you need to define events of interests and then connect these events of interest using operators. The supported operators are "AND", "OR", "FBY", "NOT", "NOTOCCUR", and "IS". The operators can accept optional temporal conditions.
Procedural window	Enables the specification of an arbitrary number of input windows and input-handler functions for each input window (that is, event stream).
Remove State window	Facilitates the transition of a stateful part of a model to a stateless part of a model.
Transpose window	Enables you to interchange an event's rows as columns, or columns as rows.
Union window	Combines multiple event streams with the same schema into a single stream, similar to an SQL union operation.

Edge Streaming Analytics provides an additional set of window types. For more information, see **Streaming Analytics Window Types**.

## 1.10 Streaming Events through a Continuous Query

### 1.10.1 Overview

The following example demonstrates what happens when events stream through the windows in a continuous query:

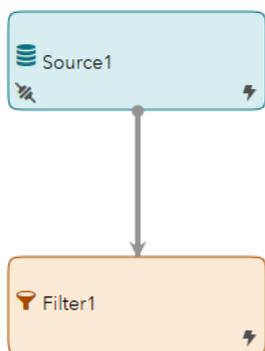


Figure 1-2 Continuous Query with a Source Window and a Filter Window

In this query, a Source window publishes events to a single Filter window. The Source window contains the following schema:

```
ID*: int32, symbol: string, quantity: int32, price: double
```

This schema consists of four fields:

Table 1-4 Source Window Schema

Field	Type
ID	32-bit integer
symbol	string
quantity	32-bit integer
price	double precision floating-point

Key fields in a schema identify an event for operations such as Insert, Update, Delete, or Upsert. Key fields must be unique. If you think of the event stream as a database, then you can think of the key fields as lookup keys.

In the schema, the ID field has the \* designator to indicate that this field is part of the key. No other field in the schema has this designator, so the ID field completely forms the key.

Here is XML code for the Source window:

```
<window-source name="Source1" pubsub="true">
  <schema>
    <fields>
      <field type="int32" name="ID" key="true" />
      <field type="string" name="symbol" />
      <field type="int32" name="quantity" />
      <field type="double" name="price" />
    </fields>
  </schema>
  <connectors>
    <connector class="fs" name="New_Connector_1">
      <properties>
        <property name="type">pub</property>
        <property name="fsname">events.csv</property>
        <property name="fstype">csv</property>
        <property name="transactional">>true</property>
        <property name="blocksize">1</property>
      </properties>
    </connector>
  </connectors>
</window-source>
```

A file and socket connector publishes a CSV file in the current directory that contains the events to be processed into the Source window.

The Filter window applies the expression `quantity > 1000`. Thus, events are passed only when the **Quantity** field in the event exceeds the value of 1000.

Here is XML code for the Filter window:

```
<window-filter name="Filter1" pubsub="true">
  <expression>quantity > 1000</expression>
</window-filter>
```

Here is the XML for the edge that connects the Source window to the Filter window:

```
<edges>
```

```
<edge target="Filter1" source="Source1" />
</edges>
```

For more information about Source windows, see [Using Source Windows](#). For more information about available filter conditions, see [Using Expressions](#).

The following sections provide detailed information about what happens when five events stream through this model.

## 1.10.2 Processing the First Event

Suppose that the first event streaming through the query is as follows:

```
e1: [i,n,10,IBM,2000,164.1]
```

1. The Source window receives  $e_1$  as an Input event. It stores the event and passes it to the Filter window.
2. The Filter window receives  $e_1$  as an Input event, as designated by the "i" in the first field. The second field in this and all subsequent events designates "normal".
3. The **Quantity** field has a value of 2000. Because the filter expression is `quantity > 1000`, the Filter window stores the input. Typically, a Filter window would pass  $e_1$  forward. However, because the Filter window has no dependent windows, there is no additional data flow for the event.

The window contents are now as follows:

Table 1- 5 Contents of Source Window After First Event

ID	Symbol	Qty	Price
10	IBM	2000	164.10

Table 1- 6 Contents of Filter Window After First Event

ID	Symbol	Qty	Price
10	IBM	2000	164.10

## 1.10.3 Processing the Second Event

The second event is as follows:

```
e2: [p,n,20,MSFT,1000,114.22]
```

1. The Source window receives  $e_2$  as an Upsert event. It checks whether the window has a stored event with a key (ID) of 20.
2. An ID of 20 is not stored, so the Source window creates a new event  $e_{2a}$ : `[I, 20, "MSFT", 1000, 114.22]`. It stores this new event and passes it to the Filter window.

1.10 Streaming Events through a Continuous Query

3. The Filter window receives  $e_{2a}$  as an Input event.
4. The value in the **Quantity** field of  $e_2$  equals 1000, which does not meet the condition set by the filter expression in the schema. Thus, this event is not stored or passed to any dependent windows.

The window contents are now as follows:

Table 1- 7 Contents of Source Window After Second Event

ID	Symbol	Qty	Price
10	IBM	2000	164.10
20	MSFT	1000	114.22

Table 1- 8 Contents of Filter Window After Second Event

ID	Symbol	Qty	Price
10	IBM	2000	164.10

### 1.10.4 Processing the Third Event

The third event is as follows:

$e_3$ : [d,n,10, , , , ]

---

**Note**

For a Delete event, you need only specify key fields. Remember that in this example, only the ID field is key.

---

1. The Source window receives  $e_3$  as a Delete event.
2. The Source window looks up the event that is stored with the same key. The Delete opcode removes the event from the Source window.
3. The Source window passes the found record to the Filter window with the Delete opcode specified. In this case, the record that is passed to the Filter window is as follows:  
 $e_{3a}$ : [d,n,10,IBM,2000,164.1]
4. The Filter window receives  $e_{3a}$  as an Input event.
5. The value in the **Quantity** field of  $e_{3a}$  equals 2000. This old event that was previously stored makes it through the filter, so it is removed.

The window contents are now as follows:

Table 1- 9 Contents of Source Window After Third Event

ID	Symbol	Qty	Price
20	MSFT	1000	114.22

The Filter window is empty.

### 1.10.5 Processing the Fourth Event

The fourth event is as follows:

```
e4: [u,n,20,MSFT,3000,114.25]
```

1. The Source window receives  $e_4$  as an Update event.
2. The Source window looks up the event stored with the same key and modifies it.
3. The Source window constructs an update block that consists of the new record with updated values marked as an update block followed by the old record that was updated.
4. The block is marked as a Delete event. The new event Update block that is passed to the Filter window looks like this:

```
e4a: [ub,n,20,MSFT,3000,114.22] , [d,n,20,MSFT,1000,114.25]
```

---

#### Note

Both the old and new records are supplied because derived windows often require the current and previous state of an event. They need these states in order to compute any incremental change caused by an Update.

---

5. The Filter window receives  $e_{4a}$  as an Input event.
  6. The value in the **Quantity** field of  $e_{4a} > 1000$ , but previously it was  $\leq 1000$ . The input did not pass the previous filter condition, but now it does pass. Because the input is not present in the filter window, the Filter window generates an Insert event of the following form:
- ```
e4b: [i,n,20,MSFT,3000,114.25]
```
7. The Insert event is stored. The Filter window would pass  $e_{4b}$ . However, because there are no dependent windows, this input does not pass. There is no further data flow for this event.

The window contents are now as follows:

Table 1- 10 Contents of Source Window After Fourth Event

| ID | Symbol | Qty  | Price  |
|----|--------|------|--------|
| 20 | MSFT   | 3000 | 114.25 |

Table 1- 11 Contents of Filter Window After Fourth Event

| ID | Symbol | Qty  | Price  |
|----|--------|------|--------|
| 20 | MSFT   | 3000 | 114.25 |

### 1.10.6 Processing the Fifth Event

The fifth event is as follows:

```
e5: [i,n,30,APPL,2000,225.06]
```

1. The Source window receives `e5` as an Insert event, stores it, and passes `e1` to the Filter window.
2. The Filter window receives `e5` as an Input event. Because the value in the **Quantity** field > 1000, the Filter window stores the input. Because the filter window has no dependent windows, there is no further data flow.

The window contents are now as follows:

Table 1- 12 Contents of Source Window After Fifth Event

| ID | Symbol | Qty  | Price  |
|----|--------|------|--------|
| 20 | MSFT   | 3000 | 114.25 |
| 30 | APPL   | 2000 | 225.06 |

Table 1- 13 Contents of Filter Window After Fifth Event

| ID | Symbol | Qty  | Price  |
|----|--------|------|--------|
| 20 | MSFT   | 3000 | 114.25 |
| 30 | APPL   | 2000 | 225.06 |

### 1.10.7 Code for the Example

The complete XML code that implements this example is available in `$DFESP_HOME/examples/xml/filter_exp.xml`. The directory also contains the CSV file that contains data to stream these five events.

## 1.11 Developing a Streaming Analytics Application

1. Design, test, and validate an event stream processing model using Edge Streaming Creator. For more information, see Using Edge Streaming Creator.
2. Run the model in the Streaming Server or within a stand-alone event stream processing application. For more information about the Streaming Server, see Managing Streaming Servers in Edge Streaming Creator.  
 For more information about the C++ modeling objects to use to write a stand-alone event stream processing application, open `$DFESP_HOME/doc/html/index.html` (UNIX deployments) or `%DFESP_HOME%doc\html\index.html` (Windows deployments) in a web browser. This provides access to the complete class and method documentation.

3. Publish one or more event streams into the engine one of the following ways:
  - Through connectors (in-process classes) or adapters (networked executables)
  - Through the Java, C, or Python publish/subscribe API
  - Using the `dfESPcontquery::injectEventBlock()` method for C++ models
4. Subscribe to relevant window event streams within continuous queries using connectors, adapters, the publish/subscribe API, Edge Streaming Creator, Edge Straming Viewer (future product), or by using the `dfESPwindow::addSubscriberCallback()` method for C++ models.

You can use Edge Streaming Manager to do the following:

- Deploy projects into test environments or production environments or both
- View the component parts of each deployment
- Monitor the health of your deployments
- Administer your deployments and manage change
- Monitor your Edge Streaming Analytics metering servers

For more information, see **EXTERNAL LINK TO ESM USER GUIDE**.

When you start a stand-alone C++ event stream processing application with `-b filename`, the application writes the events that are not processed because of computational failures to the named log file. When you do not specify this option, the same data is written to `stderr`. It is recommended to create logs of bad events so that you can monitor them for new insertions.



# Using Edge Streaming Creator

## 2.1 Edge Streaming Creator Overview

### 2.1.1 What is Edge Streaming Creator

Edge Streaming Creator is a web-based client that enables you to create, edit, upload, publish, and test event stream processing models using Edge Streaming Creator Modeler. Edge Streaming Creator Modeler displays a model as a data flow diagram, enabling you to see and control how windows relate and flow into one another.

### 2.1.2 Requirements for Solution Access and Use

Here are the requirements for accessing and using Edge Streaming Creator:

- A supported web browser has been installed
- Your screen has a minimum resolution of 1,280 x 1,024
- JavaScript has been enabled in your browser

## 2.2 Understanding the User Interface

### 2.2.1 Pages

A *page* is the highest level container in the user interface. All other user interface elements are contained within a page.

Edge Streaming Creator contains the following main pages:

- The **Projects** page enables you to create, edit, upload, download, or delete the projects that contain your models
- The **Engine Definitions** page enables you to create, edit, upload, download, or delete engine definitions
- The **Streaming Servers** page enables you to create, edit, upload, download, or delete

Streaming Servers When you first access Edge Streaming Creator, the **Projects** page appears.

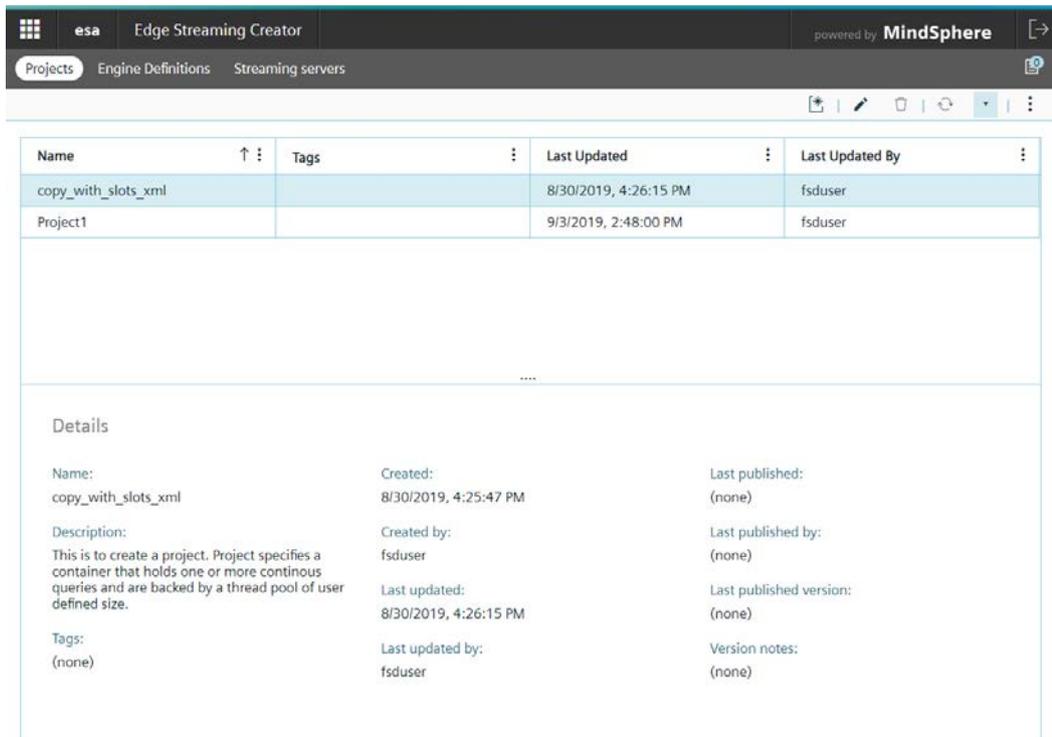


Figure 2-1 The Projects Page Displaying Active Test Projects

### 2.2.2 Panes

*Panes* enable you to view different types of information within the same page. The following figure displays a bottom pane on the **Engine Definitions** page. In this example, the pane contains further information about the engine definition selected.

To resize a pane, drag a border in the appropriate direction. To resize a horizontal pane, drag a border upward or downward. To resize a vertical pane, drag the border left or right.

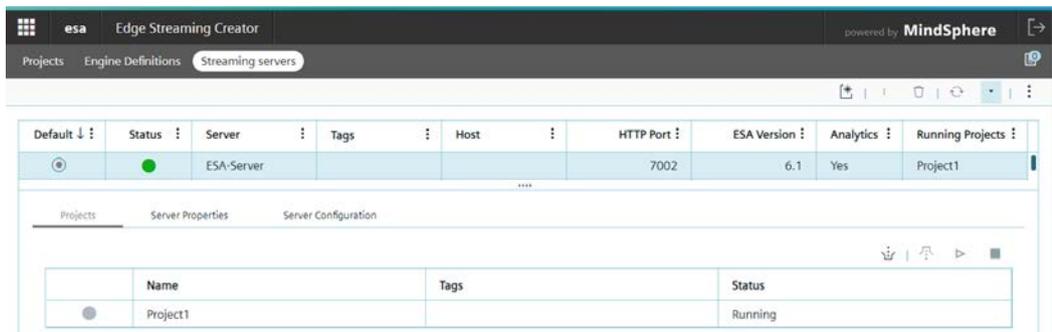


Figure 2-2 Example of a Horizontal Pane

### 2.2.3 Tiles

A *tile* is a self-contained block of information that resides within a pane or sometimes directly on a page.



Figure 2-3 Example of a Single Tile inside a Horizontal Pane

### 2.2.4 Windows

A *window* is a floating user interface element that often appears as a result of a user action. Windows generally provide a means by which to perform an action and can be closed to return you to the page from which the window was launched. The following figure shows a window that is used to upload an engine definition to Edge Streaming Creator.

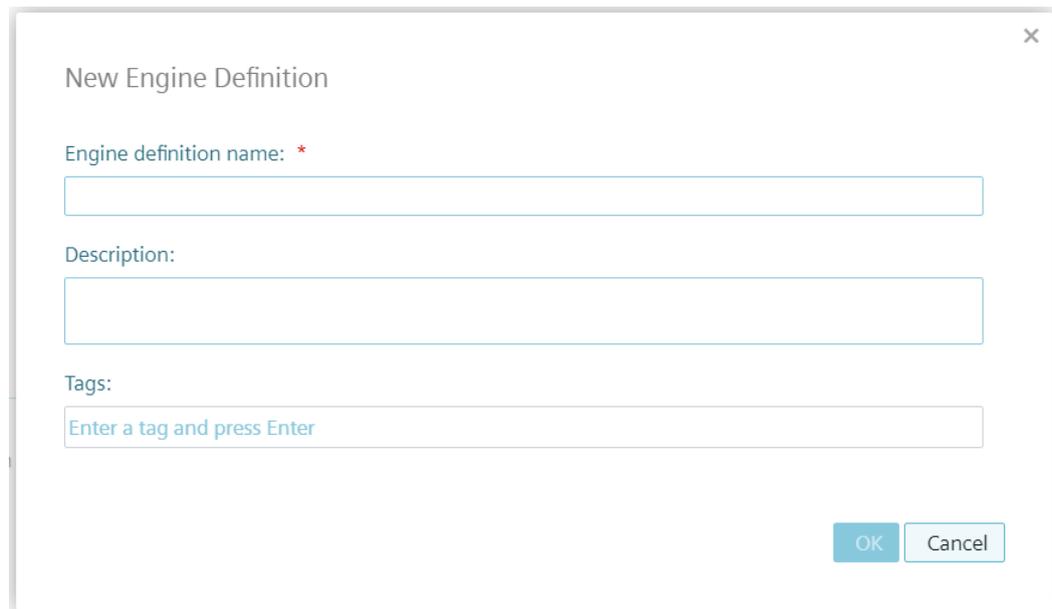


Figure 2-4 The Upload Engine Definition Window

**Note**

The user interface element *window* in Edge Streaming Creator does not have the same meaning as a Edge Streaming Analytics window. In Edge Streaming Analytics, windows are components of a continuous query. A continuous query contains a source window and one or more derived windows. Edge Streaming Analytics windows are connected by edges, which have an associated direction. Edge Streaming Creator contains both user interface element windows and Edge Streaming Analytics windows.

**2.2.5 Toolbars**

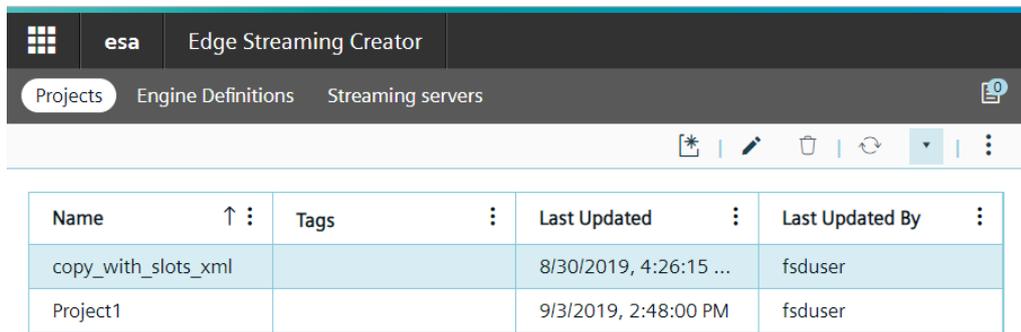


Figure 2-5 The Application Toolbars

There are three main toolbars in Edge Streaming Creator, as shown in the following table:

| Item Number | Name            | Description                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|-------------|-----------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1           | Application bar | <ul style="list-style-type: none"> <li>Displays the product name</li> <li>Displays the first character of your display name or user ID. Clicking on this character does the following:                             <ul style="list-style-type: none"> <li>Shows your display name or your user ID in full. If you have not set a display name, your user ID is displayed by default. If Edge Streaming Creator has been configured so that you do not need to log on with a user name and password, your user ID is not displayed.</li> <li>Provides access to Help and product information</li> <li>Enables you to sign out of Edge Streaming Creator (if applicable)</li> </ul> </li> </ul> |
| 2           | Menu bar        | <ul style="list-style-type: none"> <li>Provides access to the main Edge Streaming Creator pages: <b>Projects</b>, <b>Engine Definitions</b> and <b>Streaming Servers</b></li> <li>Provides access to each project, project version, model test, or engine definition that is currently open. The navigation overflow menu button displays the total number of these pages that are currently open, for example, </li> </ul>                                                                                                                                                                                                                                                                   |
| 3           | Toolbar         | <ul style="list-style-type: none"> <li>Includes buttons or tabs associated with the open item</li> <li>Enables you to perform actions associated with a project. For example, you can download a project to your computer by clicking  on the toolbar and selecting <b>Download</b></li> </ul>                                                                                                                                                                                                                                                                                                                                                                                                |

## 2.2.6 Sorting and Filtering

To make it easier to work with a large amount of information, you can sort and filter items displayed in tables. You can also show, hide, and reorder columns.

You can sort lists of items by ascending or descending order. To sort in ascending order, click the heading of the column that you want to sort. To sort in descending order, click the column again. To remove sorting, click the column a third time.

You can create filter criteria by which to display only a subset of information for a column. To create filter criteria, click  for the column that you want to apply filter criteria to, select **Filter**, and enter your filter criteria. You can configure the columns that you want to display. To do this, click  in any column, select **Columns**, and deselect the columns that you do not want to appear.

You can re-order columns. To do this, click and hold the column heading, and drag it to a different location.

## 2.2.7 Edge Streaming Creator Modeler

When you create a new project or open an existing project, a separate page that contains the project content appears. This project page displays Edge Streaming Creator Modeler, which enables you to design models in a visual way and to test them. Edge Streaming Creator Modeler also includes the XML Editor, which you can use as an alternative way to construct your model.

For more information about the modeler, see "[Using Edge Streaming Creator Modeler](#)". For more information about the XML Editor, see "[Using the XML Editor](#)".

## 2.3 Working with Projects

### 2.3.1 Overview

A project consists of one or more continuous queries. You can use Edge Streaming Creator to create, upload, download, and delete projects. You can associate your project with a defined engine. For more information, see "[Engine Definition Overview](#)". The **Projects** page enables you to view the projects in your deployment, along with their identification details and associated engines.

The following figure shows an example:

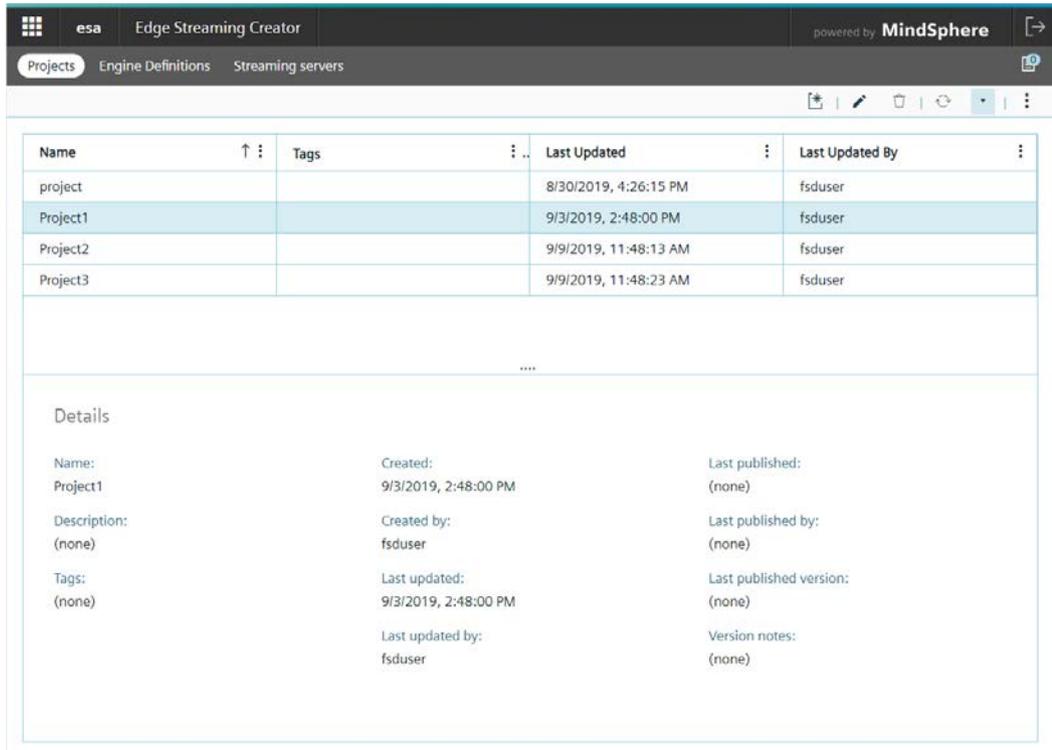


Figure 2-6 The Projects Page

**Note**

Projects that are being edited in Edge Streaming Creator are automatically locked. This ensures that changes to a project cannot be unintentionally overwritten by another user. For more information, see "Project Locking" on page 48.

The Projects page displays the following information for each Streaming Server:

- The project’s name.
- Identifying tags assigned to the project.
- The date and time that the project was last updated.
- The username of the user who last updated a project.

To refresh the main table, click .

To open a project for editing, select the project from the main table and click . Alternatively, you can open a project for editing by double-clicking the project on the main table.

To view a project’s additional information, select the relevant project on the **Projects** page.

A tile appears that contains this information, as shown here:

| Details      |                      |                         |
|--------------|----------------------|-------------------------|
| Name:        | Created:             | Last published:         |
| Project1     | 9/3/2019, 2:48:00 PM | (none)                  |
| Description: | Created by:          | Last published by:      |
| (none)       | fsduser              | (none)                  |
| Tags:        | Last updated:        | Last published version: |
| (none)       | 9/3/2019, 2:48:00 PM | (none)                  |
|              | Last updated by:     | Version notes:          |
|              | fsduser              | (none)                  |

Figure 2-7 Additional Project Details

To hide this information, click .

## 2.3.2 Create a Project

To create a new project:

1. On the **Projects** page, click . The New Project window appears.
2. In the New Project window, do the following:
  - In the **Name** field, enter a unique name for the project.

---

### Note

You must enter a unique project name. Duplicate project names are not supported. If a project has been published and then subsequently deleted, you cannot reuse the deleted project's name.

---

- If required, in the **Description** field, enter a description for the project.
- If required, in the **Tags** field, enter any identifying keywords that describe the project.
- Click **OK**.  
If you do not currently have any Streaming Servers configured, you are prompted to decide whether you want to configure a streaming Server now.
- Click **Yes** to configure a streaming Server now or click **No** to configure a streaming Server later.

3. If you chose to configure a streaming Server now, do the following:
  - In the **Name** field, enter a name to identify the new Streaming Server that you want to create.
  - In the **Host** field, enter the host name or IP address of the server that contains the new Streaming Server.
  - In the **HTTP port** field, enter the new Streaming Server's administration port number.
  - If required, in the **Description** field, enter a description of the new Streaming Server.
  - If required, in the **Tags** field, enter any keywords that describe the Streaming Server and then press Enter.
  - If required, click **Edit** to change the setting for the Authentication field:
    - None:** This is the default option.
    - Kerberos:** This option is relevant only if the Streaming Server is configured to require authentication using Kerberos.
    - OAuth token:** This option is relevant only if the Streaming Server is configured to require authentication using an OAuth token. If you select this option, an additional field appears where you must enter the OAuth token.
    - Username and password:** This option is relevant only if the Streaming Server is configured to require authentication using a username and password (SASLogon Services). If you select this option, additional fields appear where you must enter the username and password.
  - If required, select the **Connect using SSL** check box. Selecting this option is relevant only if the Streaming Server is configured to require SSL encryption.
  - If required, select the **Enable server logging** check box to enable logging on the Streaming Server.
  - If required, in the **Number of messages to retain** field, change the default number of messages that are retained by the Streaming Server log. The default is 10,000 messages.
  - Click **OK**.

---

**Note**

You can register additional Streaming Servers and view the details of existing Streaming Servers on the **Streaming Servers** page. For more information, see "**Managing Streaming Servers in Edge Streaming Creator**".

---

4. Click **OK**.  
Edge Streaming Creator Modeler appears.  
Your project is created with a set of default properties. Before you start creating your

model, configure your project's properties.

To configure your new project's properties:

- Review the default project properties in the right pane and modify them if necessary.
- You can also add or modify additional project properties, such as SAS Micro Analytic Service modules, user-defined properties, and connector orchestration.

5. Click .

---

**Note**

To create a copy of the project with a different filename, click  and select **Save project as**. Enter the relevant information into the Save As window and click **OK**.

---

### 2.3.3 Upload a Project

---

**Note**

Project XML files uploaded to Edge Streaming Creator must be encoded in UTF-8 format. Uploading project XML files that are not encoded in UTF-8 format can cause invalid characters to be displayed in Edge Streaming Creator.

---

To upload a project:

1. On the **Projects** page, click  and select **Upload projects**. The Upload Projects window appears.
2. Click .
3. Navigate to the file that contains the project that you want to upload and click **Open**.

---

**Note**

If you want to upload multiple projects that are located in the same folder, you can select the relevant projects to upload simultaneously. To do this, press and hold Ctrl, select the projects that you want to upload, and click **Open**. If your projects are located in different folders, click  again, select the relevant project, and click **Open**.

---

4. Click **Upload**.  
An icon appears. This indicates whether the project was successfully uploaded. Successfully uploaded projects are indicated by the icon . Projects that failed to upload are indicated by the icon .
5. Click **OK**.  
The projects that you uploaded appear on the **Projects** page.

---

**Note**

You cannot upload a project that has the same name as a project version that has previously been published. This also applies to a project that has the same name as a project version that has been subsequently deleted from Edge Streaming Creator.

---

### 2.3.4 Delete a Project

To delete a project, select the project that you want to delete from the table on the **Projects** page and click . Click **Yes** to confirm the deletion.

The project is permanently deleted from Edge Streaming Creator.

---

**Note**

Only the working version of a project is deleted. Published versions can still be accessed by other applications through SAS Files and Folders Services.

---

### 2.3.5 Download a Project

To download a project, select the project that you want to download from the table on the **Projects** page, click , and select **Download project**.

The project downloads to your computer.

---

**Note**

The location of the project that you downloaded might vary depending on your browser's configuration.

---

### 2.3.6 Project Metadata

When you create a project, the following unique information that identifies a project is created automatically:

- The user ID of the user who created the project
- The user ID of the user who last modified the project
- The date on which the project was created
- The date on which the project was last modified

The project information is displayed within the `<metadata>` element in your project's XML code, but it is stored in the Edge Streaming Creator database.

Metadata is also created if you perform one of the following actions:

- Apply a tag to the project
- Make changes to your model in the workspace

---

**Note**

When you make a change to your model in the workspace, a `<meta id="layout">` element is added. This element specifies the name of your model's continuous query, the names of the windows in your model, and each window's X and Y coordinates in the workspace.

---

When you publish a version of a project, the following unique information that identifies the version is created automatically:

---

**Note**

The following elements are not displayed in the XML code of a working model. However, these elements are displayed in the model's XML code if you have published the model and you are viewing the model in Read- Only mode.

---

1. The project version's unique ID
2. The project version's major version number
3. The project version's minor version number

Here is an example of a published project version's metadata:

```
<metadata>
<meta id="studioUploaded">2/26/2018</meta>
<meta id="studioUploadedBy">espuser</meta>
<meta id="studioModified">2/26/2018</meta>
<meta id="studioModifiedBy">espuser</meta>
<meta id="studioTags">Tag1</meta>
<meta id="layout">{"cq1":{"Source1":{"x":-310,"y":-315}}}</meta>
<meta id="studioVersionMajor">1</meta>
<meta id="studioVersionMinor">0</meta>
</metadata>
```

In this example, the project's version number is 1.0.

## 2.4 Working with Engines

### 2.4.1 Engine Definition Overview

An *engine* is the top-level container in the model hierarchy. Each model contains only one engine instance with a unique name. You can use Edge Streaming Creator to create, upload, download, and delete engine definitions. You can associate each project that you produce or upload with an engine definition in Edge Streaming Creator.

The **Engine Definitions** page enables you to view all operational engines in your deployment. The following figure shows an example:

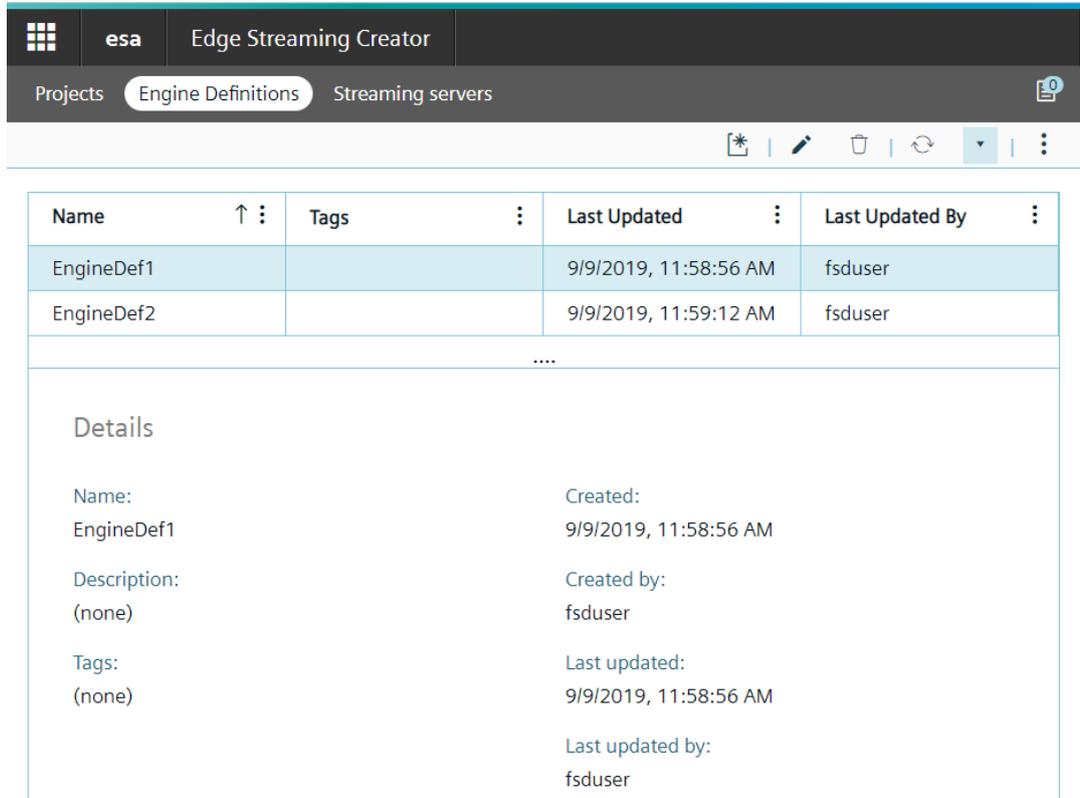


Figure 2-8 The Engine Definition Page

**Note**

Engine definitions that are being edited in Edge Streaming Creator are automatically locked. This ensures that changes to an engine definition cannot be unintentionally overwritten by another user. For more information, see **"Overview to Locking"**.

The **Engine Definition** page displays the following information about each engine definition:

- The engine definition’s name.
- Identifying tags assigned to the engine definition.
- The date and time at which the engine definition was last updated.
- The username of the user who last updated the engine definition.

To refresh the main table, click .

To view additional information for an engine definition, select the relevant engine definition in the main table.

To open an engine definition for editing, select the engine definition from main table and click .

A tile appears that contains this information, as shown here:

Details	
Name:	Created:
EngineDef1	9/9/2019, 11:58:56 AM
Description:	Created by:
(none)	fsduser
Tags:	Last updated:
(none)	9/9/2019, 11:58:56 AM
	Last updated by:
	fsduser

Figure 2-9 Additional Engine Definition Details

To hide this information, click .

Double-clicking an engine definition displays an engine definition page. This page contains:

- A **Name and Description** section – enables you to change the engine definition's name, description, and tags.
- A **Settings** section – enables you to define the engine definition's engine port, HTTP port, and fatal error handling settings. In addition, you can enable execution limits for SAS Micro Analytic Service modules that are written in Python code.
- A **Projects** tile – enables you to associate the engine definition with one or more projects. Associating an engine definition with one or more projects is useful when executing multiple projects as a single action by grouping the projects within an engine. This enables you to reuse projects without having to re-create individual projects within each new engine

## 2.4.2 Create a New Engine Definition

To create a new engine definition:

1. On the **Engine Definitions** page, click . The New Engine Definition window appears.
2. In the **Engine definition name** field, enter a name for the engine definition that you are creating.
3. Click **OK**. Your engine definition is created, and an **Engine Definition** page appears.

4. Review the information:
  - The **Name** field contains the engine definition’s name
  - In the **Description** field, enter a description for the engine definition that you are creating
  - In the **Tags** field, enter any keywords that describe the engine definition that you are creating
5. Associate projects with your engine definition:
  - In the **Projects** pane, click **+**.  
The Add Project window appears.
  - In the Available projects table, select the project that you want your engine definition to be associated with and click **+>**.

---

**Note**

To associate all available projects with your engine definition, click **+>>**.

---

The projects that you have selected appear in the Selected projects table.

- Click **Save**.  
The newly associated projects appear in the **Projects** tile.
6. If you changed any of fields on the **Engine Definition** page, click .

### 2.4.3 Upload an Engine Definition

To upload an engine definition:

1. On the **Engine Definition** page, click  and select **Upload**.  
The Upload Engine Definition window appears.
2. In the **File** field, click **Browse**.
3. Navigate to the file that contains the engine definition that you want to upload and click **Open**.
4. In the **Engine definition name** field, if necessary, adjust the name of the engine definition that you are uploading.
5. In the **Description** field, enter a description for the engine definition that you are uploading.
6. In the **Tags** field, enter any keywords that describe the engine definition that you are uploading.
7. Click **OK**.

## 2.4.4 Download an Engine Definition

To download an engine definition, select the engine definition that you want to download from the table on the **Engine Definitions** page, click , and select **Download**.

---

### Note

The location of the engine definition that you downloaded might vary depending on your browser's configuration.

---

## 2.4.5 Delete an Engine Definition

To delete an engine definition, select the engine definition that you want to delete from the table on the **Engine Definitions** page and click . Click **Yes** to confirm the deletion. The engine definition is deleted from Edge Streaming Creator.

## 2.5 Project and Engine Definition Locking

### 2.5.1 Overview to Locking

Projects and engine definitions that are being edited in Edge Streaming Creator are automatically locked. This ensures that changes to a project or to an engine definition cannot be unintentionally overwritten by another user. If you open a project or an engine definition, it is assumed that you intend to edit it. If a project or an engine definition has not been locked by another user, the project or the engine definition is locked immediately after you open it. If you are editing a project or an engine definition, the lock is released automatically if you close the tab that contains the project in the application. A lock is also released approximately two minutes after you close the application. For example, if you turn off your computer or close the browser, other users must wait two minutes until they can lock a project or lock an engine definition.

### 2.5.2 Project Locking

If you attempt to open a project that is locked by another user, you are informed that another user is editing the project. You are prompted to decide whether you want to continue. If you click **Yes**, the project opens in Edge Streaming Creator Modeler in Read-Only mode. If you click **No**, you return to the **Projects** page.

Read-Only mode enables you to view the model and, if necessary, rearrange the position of the model's windows. However, you cannot save your changes.

---

**Note**

Projects are locked against your username. If you are editing a project, it is not recommended that you open the project in another browser tab or window.

---

A banner similar to the following example appears when you open a read-only copy of a project:

 Project Locked

User ID "fsduser" is editing this project. A read-only copy will be opened.  
Do you want to continue?

Figure 2-10 Viewing a Project in Read-Only Mode

The **Windows** pane and magnification icons are unavailable in Read-Only mode. Therefore, you cannot add windows to your model using the **Windows** pane or adjust your model's magnification.

### 2.5.3 Engine Definition Locking

If you attempt to open an engine definition that is locked by another user, you are informed that another user is editing the engine definition. You are prompted to decide whether you want to continue. If you click **Yes**, the engine definition opens in Read-Only mode. If you click **No**, you return to the **Engine Definitions** page.

Read-Only mode enables you to view the engine definition's properties and to download the engine definition to your computer. However, you cannot modify the engine definition's properties or save your changes.

A banner similar to the following example appears when you open a read-only copy of an engine definition:

 Engine Locked

User ID "fsduser" is editing this engine. A read-only copy will be opened.  
Do you want to continue?

Figure 2-11 Viewing an Engine Definition in Read-Only Mode

## 2.6 Using Edge Streaming Creator Modeler

### 2.6.1 Using Edge Streaming Creator Modeler

Edge Streaming Creator Modeler enables you to construct, change, and test Edge Streaming Analytics models. A *model* specifies how an engine analyzes and then transforms input event streams into meaningful results.

Edge Streaming Creator Modeler appears when you create a new project or open an existing project.

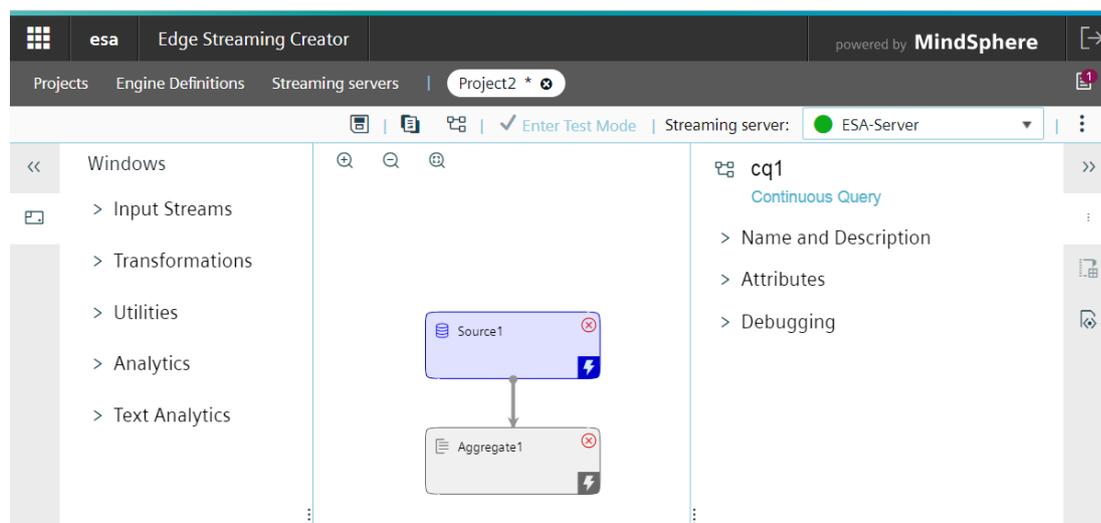


Figure 2-12 Edge Streaming Creator Modeler

#### Note

You can increase or decrease the magnification of your model by using the zoom buttons. Click  to increase the magnification and click  to decrease the magnification. To adjust the magnification of your model so that the entire model appears in the workspace, click .

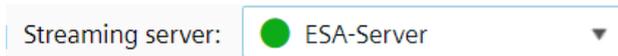
The modeler displays one continuous query at a time. When you create a new model, a continuous query named `cq1` is created by default. To construct your model, you must configure at least one continuous query. For more information about configuring continuous queries, see **Edge Streaming Analytics**.

#### Note

To pan your model, click anywhere in the workspace and then drag the cursor in the appropriate direction.

### 2.6.2 Configure a Model's Streaming Server

When you create a model, if you have not configured any Streaming Servers, you are prompted to decide whether you want to configure a streaming Server. If you decide to create a streaming Server, the Streaming Server that you create becomes the model's default Streaming Server and appears in Edge Streaming Creator Modeler:



If you want to test your model in test mode, you must create a streaming Server and assign it to your model. However, creating a streaming Server is optional when viewing or editing a model. For more information about testing your model, see "**Running a Test**".

---

#### Note

Functionality is limited if you open a model that does not have an assigned Streaming Server. For example, connector properties are unavailable to models for which Streaming Servers are not assigned.

---

You can manage your Streaming Servers on the **Streaming Servers** page. If you have configured multiple Streaming Servers, you can use this page to change the default Streaming Server that is associated with your model.

When you associate a streaming Server with your project, the association is saved in your browser's local storage. The default Streaming Server selected on the **Streaming Servers** page applies only to projects that do not have an associated Streaming Server within your browser's local storage. For example, if you have not opened the project before, the default Streaming Server is associated with the project. For more information about managing Streaming Servers, see "**Managing Streaming Servers in Edge Streaming Creator**".

### 2.6.3 Add a Window

To add windows to the continuous query that is currently displayed, drag a window from the **Windows** pane on the left to the workspace.

---

#### Note

Alternatively, you can add a window to the continuous query by double-clicking the relevant window in the **Windows** pane.

---

The windows are grouped into the following categories:

- Input Streams
- Transformations
- Utilities
- Analytics
- Text Analytics

You must ensure that you enter valid properties for each window. Entering invalid window properties causes the window to display an error icon: . Here is an example of a model where the Join window contains invalid properties:

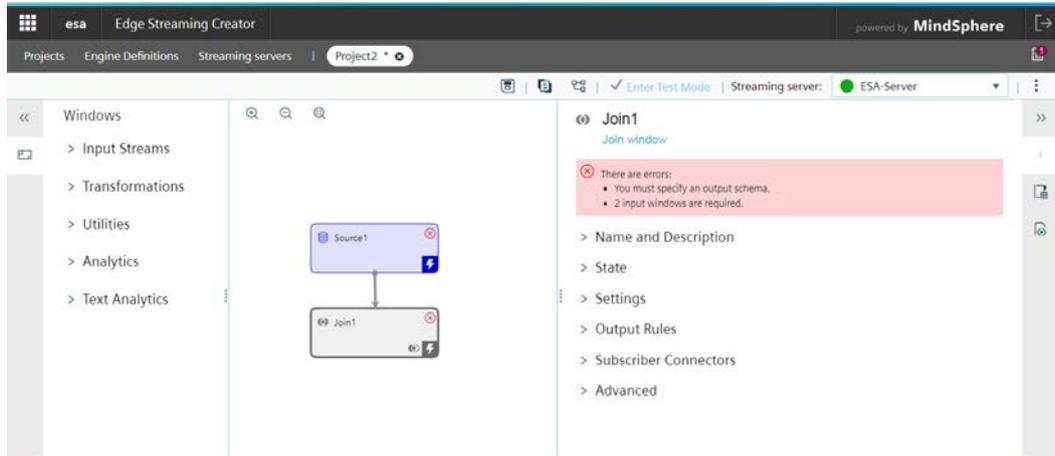


Figure 2-13 A Window Validation Error Icon and Corresponding Message

In addition, if a window requires further changes for the model to run successfully, the window displays a warning icon: . The right pane that displays the window's properties shows the corresponding warning message.

Here is an example of a model where the Source window requires further changes for the window to be in a valid state:

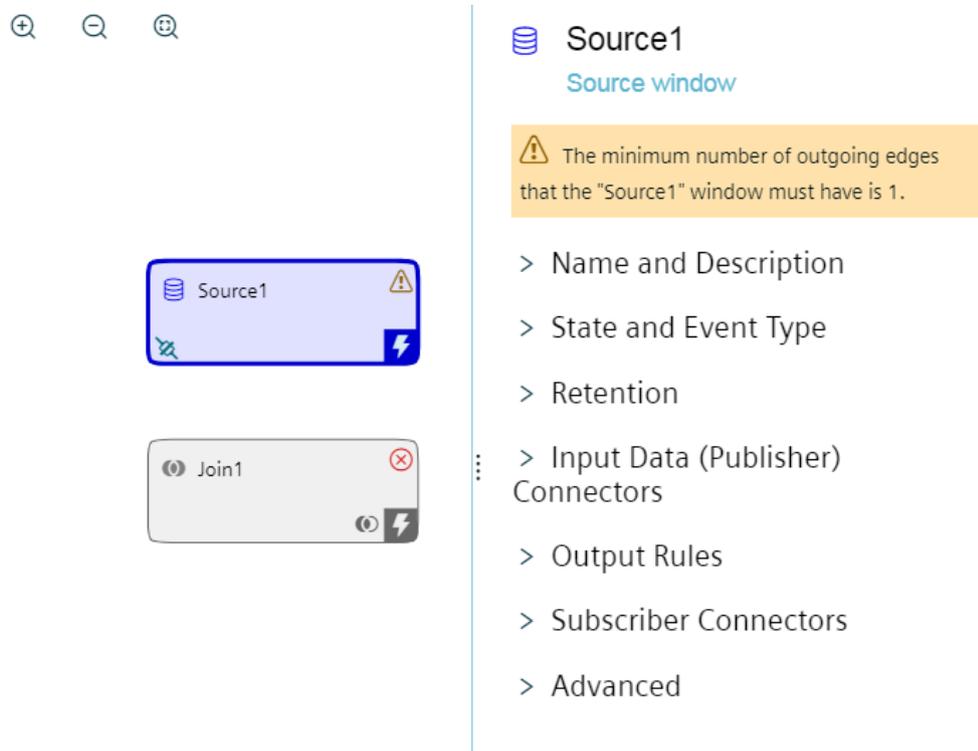


Figure 2-14 A Window Validation Warning Icon and Corresponding Message

### 2.6.4 Connecting Windows

To connect a window to another window with an edge:

1. Position the cursor over the anchor point at the bottom of the window so that the anchor point color changes to white:

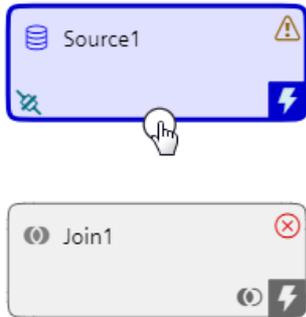


Figure 2-15 Cursor over the Anchor Point

2. Click the white anchor point, hold the left mouse button down, and draw a line to the anchor point of another window:

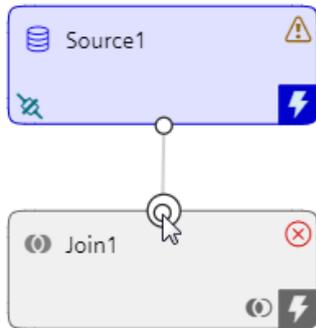


Figure 2-16 An Edge Connecting Two Windows

The edge automatically connects to the window.

---

**Note**

You cannot change a connection by moving an edge from one window to another. Instead, you must establish a new connection by creating a new edge. If you have created a connection between two windows in error, you must delete the edge. To do this, select the edge that you want to delete and press **Delete**.

---

### 2.6.5 Edge Display Types

Connecting edges can appear differently depending on the type of connection between windows.

For information about edge display types, see **the following table**:

Edge Display Type	Connecting Edge	Details
Event stream		Event stream edges connect input windows that contain event stream data.
Supporting data		Supporting data edges connect input windows that contain geometric data (that is, where the edge role is set to <code>Geometry</code> ). They can also connect secondary input windows to a Join window.
Non-data edges		Non-data edges connect windows that do not contain event stream data (that is, where the edge role is set to <code>Model</code> or <code>Request</code> ).

#### Note

An invalid edge appears as a red dashed line in Edge Streaming Creator Modeler.

## 2.6.6 Edge Roles

You can use Edge Streaming Creator Modeler to configure the edge roles of connecting edges in your model. Edge roles must be specified for edges that connect streaming analytics windows and for edges that connect Geofence and Join windows. Each edge is assigned a role by default.

To change an edge's default role:

1. In the workspace, select the edge whose role you want to change.
2. In the right pane, in the **Role** field, change the default selection.

For information about edge roles, see the following table:

Window Name	Default Edge Role	Available Edge Roles	Dependencies
Join	Left table	Left table or Right table	If you assign an edge role to a Join window's connecting edge, the remaining connecting edge is automatically assigned the alternative edge role.
Geofence	Position	Position or Geometry	If you assign an edge role to a Geofence window's connecting edge, the remaining connecting edge is automatically assigned the alternative edge role.

Window Name	Default Edge Role	Available Edge Roles	Dependencies
Model Reader	Request	Request	Not applicable
Model Supervisor	Request	Request or Model	Not applicable
Calculate	Data	Data or Request	Not applicable
Train	Data	Data or Request	Not applicable
Score	Data	Data or Model	If you assign an edge role to a connecting edge, the remaining connecting edge is automatically assigned the alternative edge role.

### 2.6.7 Delete a Window or an Edge

To remove a selected window or an edge from the model, press **Delete**.

---

**Note**

Deleting a window from a model automatically deletes all its connecting edges.

---

### 2.6.8 Window Icons

Each window in your model can display icons that represent its current state. For example, a Source window that contains a publisher connector displays . For information about each icon, see the following table:

Icon	Description
	Indicates that the window contains an error message that will cause the model to fail to run.
	Indicates that the window requires further changes for it to be in a valid state.
	Indicates that the window contains a fully stateful index that is stored in memory. <b>Note:</b> If the window contains a non-stateful index, this icon is not displayed. This color of this icon changes depending on the window that contains it. This example shows an icon on a Source window.
	Indicates that the window contains a fully stateful index that is stored in disk. <b>Note:</b> If the window contains a non-stateful index, this icon is not displayed. This color of this icon changes depending on the window that contains it. This example shows an icon on a Source window.
	Indicates that the Source window contains either a publisher connector or a subscriber connector. <b>Note:</b> Connector icons that appear on the left of a window indicate that the window contains a publisher connector. Connector icons that appear on the right of a window indicate that the window contains a subscriber connector.

Icon	Description
	Indicates that the Join window contains an inner join type.
	Indicates that the Join window contains a full outer join type.
	Indicates that the Join window contains a right outer join type.
	Indicates that the Join window contains a left outer join type.

## 2.6.9 Configure the Properties of a Window

To configure the properties of a window, click the window in the workspace. The right pane displays the properties for that window. Edit the properties as required.

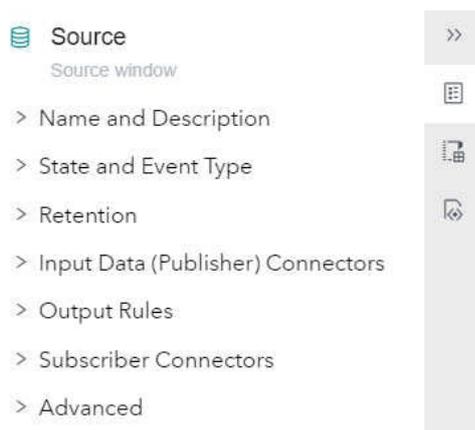


Figure 2-17 Properties of the Source Window

---

### Note

If the right pane displays XML code or an output schema, you can view the window's properties by clicking  in the right pane to display the properties instead.

---

## 2.7 Using XML Editor

### 2.7.1 Using the XML Editor

Edge Streaming Creator Modeler includes the XML Editor. You can use it as an alternative way of creating models, compared to the visual modeling capabilities in Edge Streaming Creator Modeler. The workspace displays a snapshot of your model's XML code. You can use the XML Editor to rename a window. To do this, select the window that you want to rename in the workspace and then change the window's name in the XML Editor.



**CAUTION**  
Manually editing your model's XML code using the XML Editor can result in an invalid model.

Using Edge Streaming Creator Modeler to construct your model limits the possibility of your model containing invalid XML code. You must correct any invalid XML in the XML Editor before you can switch back to the Properties pane. Changes that you make manually in the XML Editor are not always reflected in the workspace. Using the XML Editor to rename a window, without first selecting it, results in the window being replaced by a new window in a default position on the workspace. This invalidates your model. Any connections to or from the redundant window must then be deleted and then re-created in the workspace. Alternatively, you can manually edit the edges in the XML Editor.

To open the XML Editor, open a project and click  on the right toolbar.

The right pane displays the XML Editor.

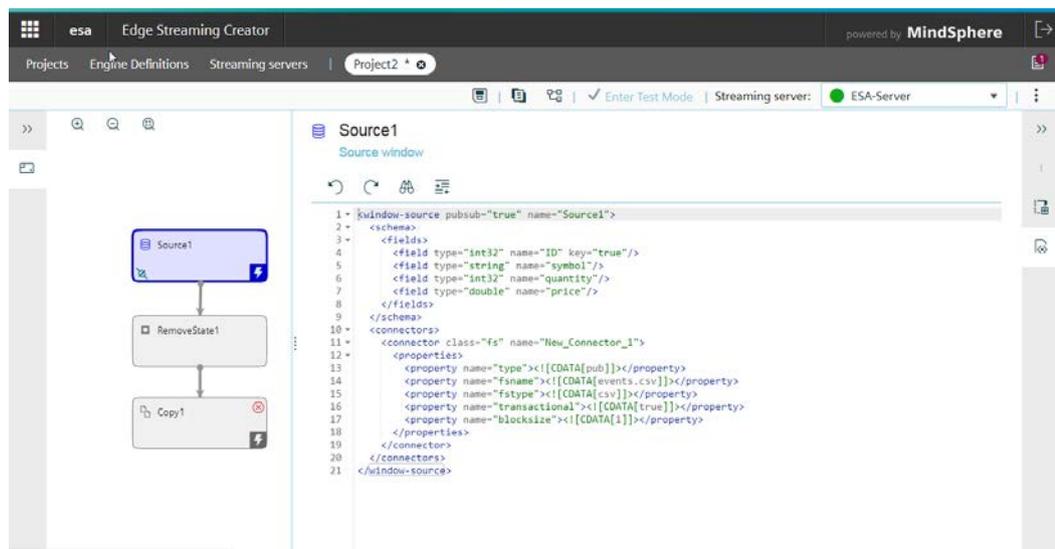


Figure 2-18 The XML Editor

Selecting a specific element in your workspace reloads the XML Editor to display only the corresponding section of XML code. To display the entire project's XML again, click . If

you have associated a project with an engine, the XML code that specifies the engine is not included in the project's XML code. This information is instead included as metadata in the Edge Streaming Creator database.

If you selected a specific ESP window, clicking an area of white space in your workspace reloads the XML Editor to display the XML relating to your model's continuous query.

If you add a comment to your XML code that is not enclosed within its relevant XML element, the comment is automatically moved within the XML element. This occurs when the XML code is reordered. For example, the XML code is reordered if you save your model or if you move away from the XML editor.

When a model is created, optional attributes are not included in its XML code. Models also contain settings that are not directly specified in the model's XML code, but they are represented in the user interface. For example, if a Source window has a default window state of **(inherit from query) pi\_HASH**, the implied setting is not displayed in the XML code. See the example shown here:



Figure 2-19 The XML Editor Displaying a Source Window's XML Code without the Window's Default State

Changing the window's state from its default value includes the attribute in the model's XML code, as shown here:



Figure 2-20 The XML Editor Displaying a Source Window's XML Code with the Window's Default State

Unique identifying project information is displayed within the `<metadata>` element in your project's XML code. Although the metadata is displayed in your project's XML code, it is located in the Edge Streaming Creator database. For more information, see "**Project Metadata**".

## 2.7.2 Using Editing Tools and Keyboard Shortcuts

The XML Editor includes a toolbar that contains editing tools. These tools are also accessible using keyboard shortcuts.

Icon	Action	Keyboard Shortcut
	Reverts your previous change	Ctrl + Z
	Reverts the effects of the undo action	Ctrl + Y
	Prompts you to search for specific text. Pressing Ctrl + F again prompts you to replace the text that you have searched for.	Ctrl + F

Icon	Action	Keyboard Shortcut
	Formats the XML code that you have manually entered	Not available
No icon	Removes the code that you have selected from its original position	Ctrl + X
No icon	Copies the code that you have selected to the clipboard	Ctrl + C
No icon	Pastes the code on the clipboard at the cursor's position	Ctrl + V
No icon	Selects all of the code in the XML editor.	Ctrl + A

### 2.7.3 Validation

The XML Editor automatically validates the syntax of the code that you enter. If you enter invalid code, the XML Editor displays the following error message:

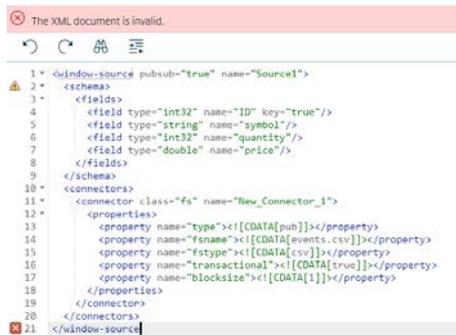


Figure 2-21 Invalid XML Warning

The XML error message also indicates the error's location with a breadcrumb trail. Each section of the breadcrumb trail represents an XML tag in your XML code. The top-level XML tag in your XML code is not included in the breadcrumb trail.

Position the cursor over the warning icon to view a generic description of the error in your XML code. The icon also displays the location of the error in your XML code.

For a more detailed description of the error, position the cursor over the icon .

## 2.7.4 Efficiency Tips

For some window types, you can copy schema fields between windows if there are windows in your model that use the same fields. In the Output Schema window, click  to open the Copy Fields from Input Schema window. Select the schema fields that you want to copy and click **OK**. Alternatively, you can use the XML Editor to copy and paste the fields between the windows.

---

### Note

This functionality is not available for window types where it is not appropriate for schema fields to be copied from another window. For example, you cannot copy schema fields to or from windows that contain schemas that are implied or have been internally generated.

---

## 2.8 Continuous queries

### 2.8.1 Configuring Continuous Queries in Edge Streaming Creator

Continuous queries allow engines to analyze and manipulate data. *Continuous queries* are queries that run automatically and periodically on data in real time.

Models must contain at least one continuous query. Edge Streaming Creator Modeler creates a continuous query `cq1` by default. You can then add and configure windows within this continuous query. Your model can contain many continuous queries.

Your continuous query must contain at least one Source window. Source windows connect to one or more derived windows (for example, a Pattern or Join window). After you have created a Source window, you can then add derived windows to your model.

### 2.8.2 Configure the Properties of a Continuous Query

To configure the properties of a continuous query:

1. On the **Projects** page, right-click the project that contains the continuous query that you want to configure, and select **Open Project**.  
Edge Streaming Creator Modeler appears. The right pane displays the project's properties.
2. Click  on the toolbar.  
The right pane displays the properties of the continuous query that you selected when the project was last saved.
3. Configure the fields as required.

To add a new continuous query to your model, click  on the toolbar and select **Add continuous query**. You can also delete continuous queries from your model by selecting **Delete continuous query**.

To switch between each continuous query in your model, select the continuous query that you want to view from the **Continuous Query** drop-down list on the toolbar.



Figure 2-22 The Continuous Query Drop-down List on the Toolbar

## 2.9 Testing Models in Edge Streaming Creator

### 2.9.1 Testing Models in Edge Streaming Creator

You can use Edge Streaming Creator to verify that your model operates as intended. You can analyze how incoming data is transformed into meaningful event streams that can be consumed by subscribers.

---

#### Note

An *engine* is the top-level container in the model hierarchy and can contain one or more projects. A project can contain one or more continuous queries. When you are running a model in test mode, only the project is tested. Test mode does not test engines.

---

For information about how to run a test, see "**Running a Test**".

You can also analyze your model's performance in real-time by viewing its log. For more information, see "**Viewing a Model's Test Logs in Edge Streaming Creator**".

### 2.9.2 Running a Test

To run a test, do the following:

1. On the **Projects** page, right-click the project that contains the model that you want to test.
2. Select **Open project** from the menu.  
The project appears in a new page.

---

#### Note

Ensure you have saved any changes that you have made to the project. Projects that contain unsaved changes cannot be opened in test mode.

---

3. Click  **Enter Test Mode** .  
A page appears, enabling you to test your model.

4. In the **Streaming Server** drop-down list, verify that a streaming Server has been selected to perform the test on.

---

**Note**

If Edge Streaming Creator does not contain any registered Streaming Servers, you must register one before you can continue testing your model. You can register a new Streaming Server on the **Streaming Servers** page.

---

5. To configure your test's settings, click  and select **Output data settings**. The Output Data Settings window appears.
6. To return events to test mode in real time, select **Return events from server as they happen**. Alternatively, to return events to test mode in pages, select **Return events from server in pages**.

---

**Note**

This version of Edge Streaming Creator uses the WebSocket protocol to subscribe to windows. Models that are executed might display events in a different sequence than models that were executed in previous versions. The order of the events delivered to the WebSocket subscriber will not match the order of the events received from the engine. If you are using the WebSocket subscriber, the event key and event state take precedence over the sequence of events received from the engine.

---

If you selected **Return events from the server in pages**, do the following:

- In the **Maximum page size (events)** field, enter the maximum number of events to be displayed in a results page.

---

**Note**

To prevent an excessive number of results being simultaneously returned from the Streaming Server, you can display results in paged format. This can improve your browser's performance. However, if the total number of results is greater than maximum page size, some results are not displayed.

---

- In the **Interval (ms)**, enter an interval at which each page is to be returned from the Streaming Server (in milliseconds).
7. Click **OK**.
  8. Each window's results appear in their corresponding tab. Only windows whose event stream you have subscribed to can display data. If you subscribed to view the results of six windows or fewer, you can choose to view your test results in windowed format. To do this , and select **Tile**.
  9. From the list of windows on the left of the screen, select the windows whose results you want to view.
  10. To run the test, click  **Run Test**.  
You can group information by column. The results table contains a horizontal bar at the top of the table, with the text **Drag a column header here to group by that column**. To

group information by column, drag a column heading to the bar. If required, you can drag additional columns to the bar.

Alternatively, you can view your test results by opening the output file that you specified in your subscriber connector properties.

The Status indicator informs you of your test's current status. Your test can have the following statuses:

Status	Description
Initial	The test is in an initial state and has not started.
Starting	The test is starting.
Started	The test has been started and is running.
Stopping	The test is stopping.
Stopped	The test has been stopped.

You can use the **Show formatted fields** check box to choose whether data appears exactly as it was received from the Streaming Server or with additional formatting. Here are some examples of additional formatting that is applied when the check box is selected:

- Dates are shown as Coordinated Universal Time (UTC) in ISO 8601 format, for example, 2018-11-30T13:33:47.000Z. If you clear the check box, dates appear in

UNIX Epoch time, as this is the format in which the data is received from the Streaming Server.

- A dot is used as a separator in certain types of numerical data, rather than another separator, such as a comma. If you clear the check box and your locale is set to a locale that uses another separator, that separator is displayed instead of a dot.
- Opcodes are displayed using their localized names if your locale is not set to an English-language locale. If you clear the check box, opcodes are always shown in English, as this is how the data is received from the Streaming Server.

### Note

If a window in your model contains more than 1,000 results, only the last 1,000 results are displayed in test mode.

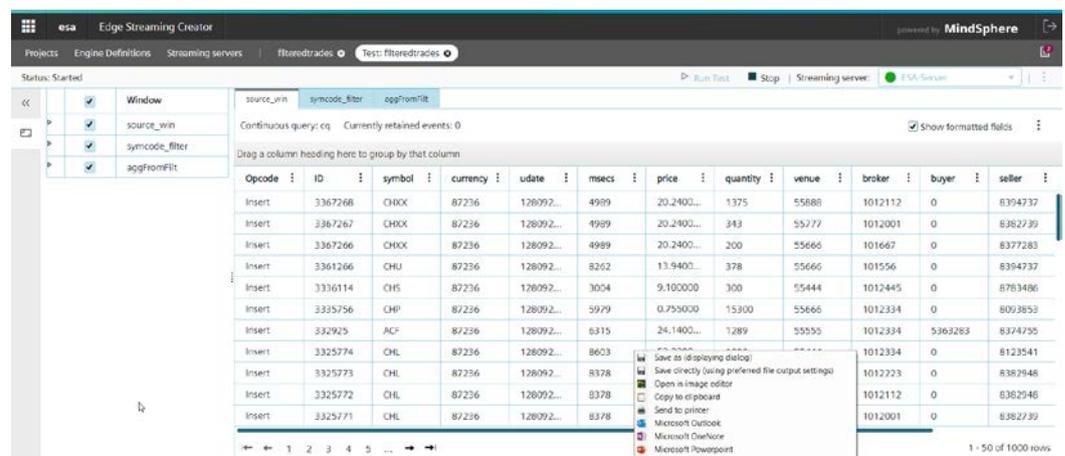


Figure 2-23 Test Mode

All output schema fields appear by default for each window in your model. However, you can filter these fields to ensure that only specific fields appear. To do this, in the left pane, click in the row for the window whose fields you want to filter. Deselect the fields that you do not want to appear. To deselect all fields in the window's output schema, click . If a window in your model contains more than 15 fields in its schema, only the first 15 fields that you have selected are shown in test mode.

11. To stop the test, click **Stop**.

12. When you have finished testing your model, close the page.

### Note

If the test fails and the resulting error message does not explain what caused the failure, you can troubleshoot the problem by checking the test logs in the Log pane. For more information, see **"Viewing a Model's Test Logs in Edge Streaming Creator"**.

### 2.9.3 Viewing a Model's Test Logs in Edge Streaming Creator

You can monitor your model in real-time by viewing the model's logs in test mode. Log messages appear in the Log pane, a horizontal pane located at the bottom of test mode. Test mode logging is disabled by default on each Streaming Server. To enable test mode logging on the Streaming Server that you are currently using, click **Enable** in the Log pane.

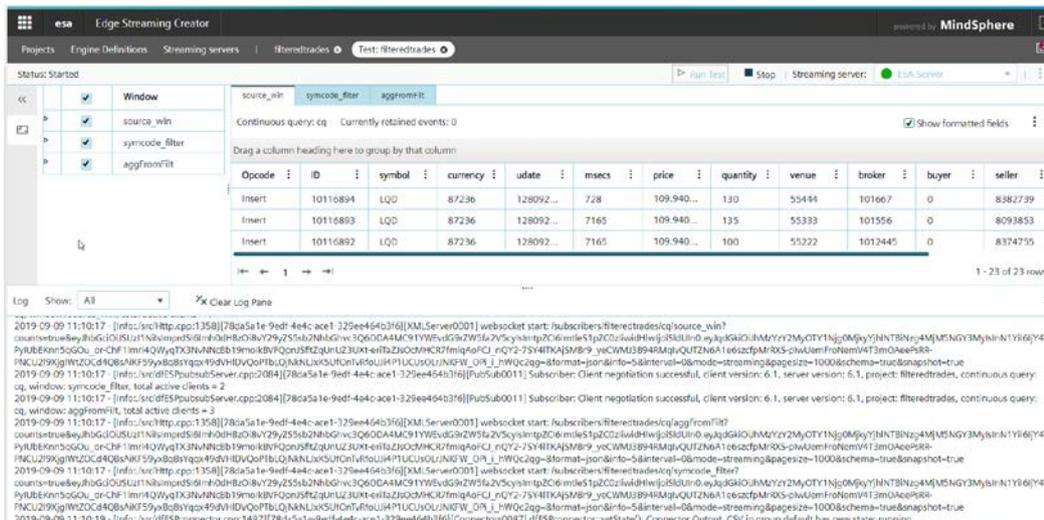


Figure 2-24 Test Mode with Logging Enabled

To filter log messages by message type, select one of the following options from the **Show** drop-down list:

- All – Shows all log message types.
- Informational – Shows generic log messages that are not warnings or errors. For example, in Figure 24, an informational message shows the date and time for when a specific websocket started.
- Warnings – Shows only warning messages.
- Fatal and Errors – Shows only fatal errors and normal error messages. For example, in Figure 24, an error message shows that XML schema validation has failed.

To clear the contents of the Log pane, click **Clear Log Pane**.

To close the Log pane, click **X**. To reopen the Log pane, click **:** and select **Log** from the drop-down list.

## 2.10 Managing Streaming Servers in Edge Streaming Creator

### 2.10.1 Managing Streaming Servers in Edge Streaming Creator

You can use the **Streaming Servers** page to view details of existing Streaming Servers in your deployment. You can also use the controls on this page to create, edit, and delete Streaming Servers.

The following figure shows an example:

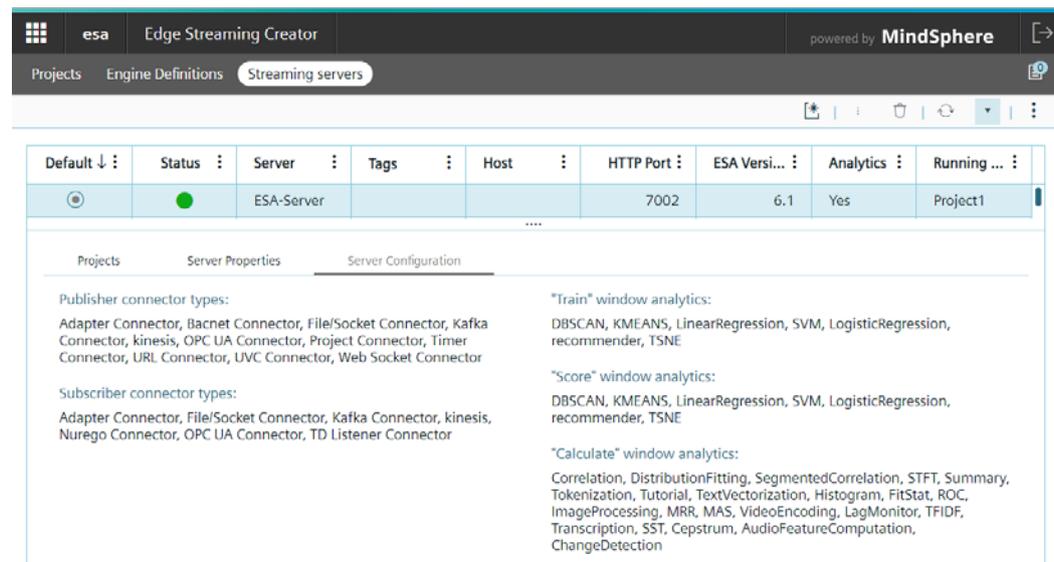


Figure 2-25 The Streaming Servers Page

The **Streaming Servers** page displays the following information for each Streaming Server:

- Whether it is the default Streaming Server.
- The Streaming Server's status.
- The Streaming Server's name.
- Identifying tags assigned to the Streaming Server.
- The host on which the Streaming Server is running.
- The port that is used for HTTP administration requests.
- The Edge Streaming Analytics version installed on the host on which the Streaming Server is running.
- Whether streaming analytics is available on the Streaming Server.
- The number of running projects on the Streaming Server.

The Status column displays an icon summarizing the Streaming Server's condition. The condition of the Streaming Server is determined by the server's connectivity and availability. This information helps you focus on the Streaming Servers that have problems. The following icons can appear in the Status column:

Icon	Description
Available ●	The Streaming Server is available and operating normally.
Errors Reported ●	The Streaming Server is not available.

You can group information by column. To enable grouping, click and select **Group columns**.

A horizontal bar at the top of the table appears. The bar contains the text **Drag a column header here to group by that column**. To group information by column, drag a column heading to the bar. If required, you can drag additional columns to the bar.

To view additional information for a streaming Server, select the relevant Streaming Server on the **Streaming Servers** page.

The **Server Properties** tab in the bottom pane contains information about the Streaming Server's properties, such as the authentication method used on the Streaming Server. For example, a streaming Server can use Kerberos, OAuth, or SASLogon services authentication. This tab also displays information such as the Streaming Server's host name, HTTP port, and whether Transport Layer Security (TLS) is enabled on the Streaming Server.

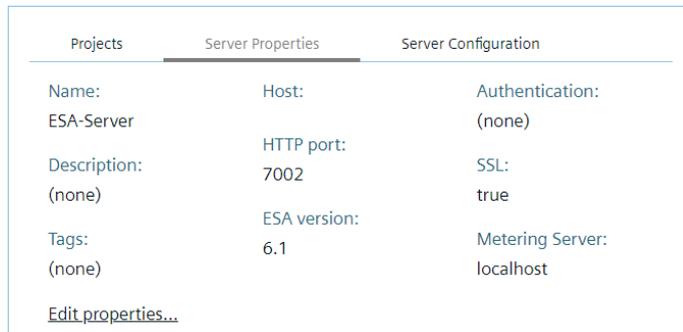


Figure 2-26 The Server Properties Tab

The **Projects** tab appears by default and specifies the projects that the Streaming Server contains. You can use the **Projects** tab to load, unload, start, and stop a project on the Streaming Server selected.

To load a project onto the Streaming Server, click . In the Load Project window, select the project that you want to load from the table and click **OK**. This process uploads a working copy of the project to the Streaming Server.

**Note**

Uploading projects that have already been published is not permitted

After you have loaded the project onto the Streaming Server, you can run the project by clicking . The **Running Projects** field on the corresponding row in the table on the **Streaming Servers** page is updated with the running project's name. To stop the running project, click . To unload the project from its Streaming Server, click .

The **Server Properties** tab contains identifying information about the Streaming Server. To edit the Streaming Server's properties, click **Edit properties**. The Edit Streaming Server

Properties window appears, enabling you to do this. For more information, see "**Edit a streaming Server's Properties**".

The **Server Configuration** tab contains information about connector types, streaming analytics, and whether Edge Streaming Analytics has been enabled to meter the number of events that are processed on the Streaming Server.

To hide this additional information, click .

## 2.10.2 Create a streaming Server

To create a new Streaming Server:

1. On the **Streaming Servers** page, click .  
The Streaming Server Properties window appears.
2. In the **Name** field, enter a name to identify the new Streaming Server.
3. In the **Host** field, enter the new Streaming Server's host name or IP address.
4. In the **HTTP port** field, enter the new Streaming Server's administration port number.
5. If required, in the **Description** field, enter a description of the new Streaming Server.
6. If required, in the **Tags** field, enter any keywords that describe the Streaming Server and then press Enter.
7. If required, click **Edit** to change the setting for the **Authentication** field:
  - **None**: This is the default option.
  - **Kerberos**: This option is relevant only if the Streaming Server is configured to require authentication using Kerberos.
  - **OAuth token**: This option is relevant only if the Streaming Server is configured to require authentication using an OAuth token. If you select this option, an additional field appears where you must enter the OAuth token.
  - **Username and password**: This option is relevant only if the Streaming Server is configured to require authentication using a username and password (SASLogon Services). If you select this option, additional fields appear where you must enter or confirm the username and password.
8. If required, select the **Connect using SSL** check box. Selecting this option is relevant only if the Streaming Server is configured to require SSL encryption.
9. If required, select the **Enable server logging** check box to enable logging on the Streaming Server.
10. If required, in the **Number of messages to retain** field, change the default number of messages that are retained by the Streaming Server log. The default is 10,000 messages.
11. Click **OK**.

### 2.10.3 Edit a streaming Server's Properties

1. On the **Streaming Servers** page, select the Streaming Server that you want to open and click . The Edit Streaming Server Properties window appears.
2. Edit the information in the **Name**, **Host**, and **HTTP Port** fields as required.
3. If required, click Edit to change the setting for the Authentication field:
  - **None**: This is the default option.
  - **Kerberos**: This option is relevant only if the Streaming Server is configured to require authentication using Kerberos.
  - **OAuth token**: This option is relevant only if the Streaming Server is configured to require authentication using an OAuth token. If you select this option, an additional field appears where you must enter the OAuth token.
  - **Username and password**: This option is relevant only if the Streaming Server is configured to require authentication using a username and password (SASLogon Services). If you select this option, additional fields appear where you must enter or confirm the username and password.
4. If required, select the **Connect using SSL** check box. Selecting this option is relevant only if the Streaming Server is configured to require SSL encryption.
5. If required, select the **Enable server logging** check box to enable logging on the Streaming Server.
6. If required, in the **Number of messages to retain** field, change the number of messages that are retained by the Streaming Server log. The default is 10,000 messages.
7. Click **OK**.

### 2.10.4 Delete a streaming Server

Deleting a streaming Server removes it from the table on the **Streaming Servers** page. Deleting Streaming Servers can be useful if the table contains Streaming Servers that are no longer used. You can delete a streaming Server that is still running. To delete a specific Streaming Server from the table:

1. On the **Streaming Servers** page, select the Streaming Server that you want to delete and click . The Remove Streaming Server window appears.
2. Click **Yes** to confirm the deletion of the Streaming Server.

### 2.10.5 Refresh the Main Table of Streaming Servers on the Manage Streaming Servers Page

To refresh the main table of Streaming Servers on the **Streaming Servers** page, click .

## 2.11 Publishing Project Versions

### 2.11.1 Publishing Project Versions

You can use Edge Streaming Creator to create and manage multiple versions of a project. Publishing a project version from Edge Streaming Creator makes the project version available to Edge Streaming Manager.

You cannot edit project versions after they have been published. A project that you can edit is designated as the working copy of the project. It does not become a project version until you publish it. The working copy of the project enables you to make changes to the project without affecting any project versions that you previously published.

When you publish a project version, the version's XML code is updated to display its unique ID number. Project versions that are published for the first time are assigned a version number of 1.0. The number to the left of the decimal point is the project's major version number. The number to the right of the decimal point is the project's minor version number. Publishing subsequent versions of a project increments the major version number. In Edge Streaming Creator, you can publish major project versions, but you cannot publish minor project versions. Minor versions are created when you make a change to the project version in Edge Streaming Manager.

You can view a project's major versions and minor versions in the version hierarchy on the **Versioning** page.

### 2.11.2 Publish a Version

To publish a version of a project:

1. On the **Projects** page, right-click the relevant project and select **Open Project**. Edge Streaming Creator Modeler appears.

---

**Note**

To create a new version of a project, the version must exist in a valid state.

---

2. Click .  
The **Versioning** page appears. This page contains a version hierarchy that displays the versions of the project that you are working on.

3. Click .  
The Publish — Version window appears.
  - In the **Version notes** field, enter any notes that relate to the version of the project that you want to publish. This enables you to maintain a record of a project's version history.

---

**Note**

You cannot modify the version notes of a project version that has already been published.

---

- Click **OK**.  
The **Versioning** page displays your published project version in the version hierarchy.
4. To view information relating to the project version that you published, select the relevant version in the version hierarchy on the left.

When you publish a project version, the version's XML code is updated to display its unique ID number. The project's ID number, major version number, and minor version number are specified in the version's XML code as metadata. For more information, see "**Overview**".

### 2.11.3 View a Published Version

To view a published version of a project:

1. On the **Projects** page, right-click the relevant project and select **Open Project**.  
Edge Streaming Creator Modeler appears.
2. Click .  
The **Versioning** page appears. This page displays a version hierarchy containing the current and previous versions of the project.
3. In the version hierarchy on the left, select the version of the project that you want to view in Edge Streaming Creator Modeler.
4. Click .  
The published version is displayed in Read-Only mode

---

**Note**

You cannot make changes to a version of a project that has already been published. If you want to make more changes to a project, a new working copy is made available for you to edit in Edge Streaming Creator.

---

## 2.11.4 Revert to a Previous Version

To revert to a previous version:

1. On the **Projects** page, right-click the relevant project and select **Open Project**.  
Edge Streaming Creator Modeler appears.
2. Click .  
The **Versioning** page appears. This page displays a version hierarchy containing the current and previous versions of the project.
3. In the version hierarchy on the left, select the version of the project that you want to revert to.
4. Click .  
The Revert to Version window appears.
5. Click **Yes** to confirm the reversion.

The working version of the project reverts to the published version that you selected in the version hierarchy. You cannot undo this operation.

## 2.11.5 Download a Version

To download a previously published version:

1. On the **Projects** page, right-click the relevant project and select **Open Project**.  
Edge Streaming Creator Modeler appears.
2. Click .  
The **Versioning** page appears. This page displays a version hierarchy containing the current and previous versions of the project.
3. In the version hierarchy on the left, select the version of the project that you want to download.
4. Click .  
The project version downloads to your computer.

---

### Note

The location of the project version that you downloaded might vary depending on your browser's configuration.

---

## 2.12 Example: Processing Trades

### 2.12.1 Overview

This example creates a model that processes stock market trades. The model identifies large securities transactions and the traders who were involved in those trades. The model performs the following actions:

- Events about securities transactions are streamed to a Source window called Trades
- Receives information about traders using a Source window called Traders
- Identifies large trades using a Filter window called LargeTrades matches the large trades with the traders who made those trades using a Join window called AddTraderName
- Computes the total cost of the large trades using a Compute window called TotalCost
- Aggregates the large trades by security using an Aggregate window called BySecurity

This example uses three files listed below:

- The XML file (trades.xml) associated with this example.
- trades.csv is an input file. This file contains events relating to securities transactions.
- traders.csv is an input file. This file contains events relating to the traders involved in the securities transactions.

### 2.12.2 Project Details

This project contains six windows:

- Trades is a Source window. This is where the securities from the trades.csv file enter the model.
- Traders is a Source window. This is where the trader names from the traders.csv file enter the model.
- LargeTrades is a Filter window. This window filters out all trades not in the specified range.
- AddTraderName is a Join window. This window combines the large trades with their corresponding trader names.
- TotalCost is a Compute window. This window shows the total cost of each transaction. You can use this information to identify high-value transactions.
- BySecurity is an Aggregate window. This window shows all the inserts, deletes, and update blocks for the large trades.

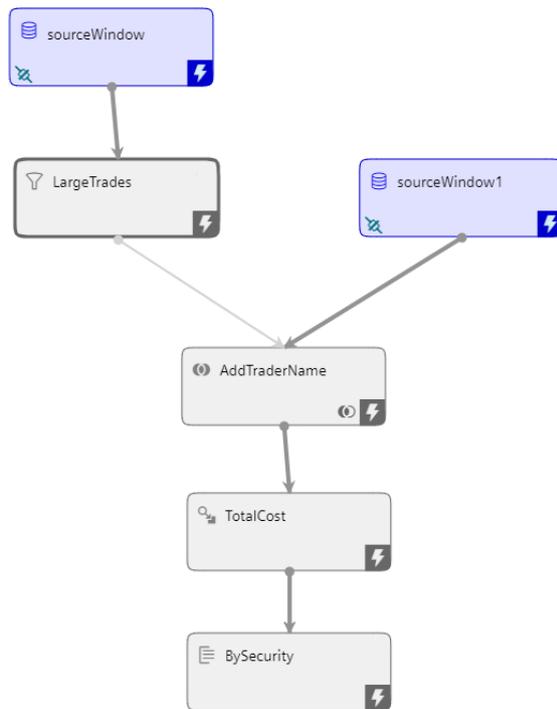


Figure 2-27 Diagram of the Processing Trades Project

---

### Note

The comma-separated value (CSV) data and model XML code that are used in this example are available within your installation, typically in the following location: `/opt/sas/viya/home/SASEventStreamProcessingEngine/<release>/examples/xml/trades_xml`. Replace `<release>` with the release number in your installation directory path.

---

### 2.12.3 Example Steps

To complete this example, follow these steps:

1. On the Projects page, click .  
The New Project window appears.
2. In the New Project window, do the following:
  - In the **Name** field, enter `Trades`.
  - In the **Description** field, enter a description. Here is an example: `This model can be used to identify large securities transactions and the traders who were involved in those trades.`
  - Click **OK**.

If you do not currently have any Streaming Servers configured, you are prompted to decide whether you want to configure a streaming Server now.

---

#### Note

It is assumed that you do not have any Streaming Servers configured. If you already have Streaming Servers configured, go to step 5.

---

3. Click **Yes** to configure a streaming Server now. The Streaming Server Properties window appears.
4. Configure a streaming Server:
  - In the **Name** field, enter a name to identify the new Streaming Server that you want to create.
  - In the **Host** field, enter the host name or IP address of the new Streaming Server.
  - In the **HTTP port** field, enter the new Streaming Server's HTTP publish/subscribe port.
  - If required, in the **Description** field, enter a description of the new Streaming Server.
  - If required, in the **Tags** field, enter any keywords that describe the Streaming Server and then press Enter.
  - If required, click **Edit** to change the setting for the **Authentication** field:
    - None**: This is the default option.
  - If required, select the **Connect using SSL** check box. Selecting this option is relevant only if the Streaming Server is configured to require SSL encryption.
  - If required, select the **Enable server logging** check box to enable logging on the Streaming Server.
  - If required, in the **Number of messages to retain** field, change the default number of messages that are retained by the Streaming Server log. The default is 10,000 messages.
  - Click **OK**.

Your project is created with a set of default properties.

5. In the right pane, configure your project's properties:
  - Expand **Attributes**.
  - In the **Threads** field, change the thread pool size to **4**.
6. Configure the project's continuous query:
  - Click .
  - In the right pane, in the **Name** field, change the continuous query's default name `cq1` to `trades_cq`.
7. Expand **Input Streams** on the **Windows** pane on the left and drag a Source window to the workspace. The right pane displays the Source window's properties.
 

This window receives events about securities transactions.
8. Specify a name for the Source window: In the right pane, in the **Name** field, change the default name to `Trades`.
9. Specify an output schema for the Trades window:
  - In the right pane, click .
  - Click .
  - The Output Schema window appears.
  - Click  to add a row to the schema table. After you add a row, click  again to add the next row.

Enter the following values in the rows:

Key	Field Name	Type
Y	tradeID	String
N	security	String
N	quantity	Int32
N	price	Double
N	traderID	Int64
N	time	Timestamp

- Click **OK**.
10. Configure the Trades window to stream events from a file called `trades.csv` that contains securities transactions. You can find this example CSV file in the `trades_xml` folder in the `examples` directory. To add a connector to this CSV file:
    - In the right pane, click .
    - Expand **Input Data (Publisher) Connectors**.
    - Click .
    - The Connector Configuration window appears.
    - In the **Name** field, replace the default value with `TradesConnector`.
    - In the **Fsname** field, enter the path to the CSV file. For example, you might enter `/opt/sas/viya/home/SASEventStreamProcessingEngine/<release>/examples/xml/trades_xml/`

trades.csv. Replace *<release>* with the release number in your installation directory path.

- In the **Fstype** drop-down list, select **csv**.
- Configure the TradesConnector connector's properties:
- Click **All properties**.

The All Properties window appears.

- Enter %d/%b/%Y:%H:%M:%S in the **dateformat** property's **Value** field.
- Click **OK**.
- Click **OK**.

11. Collapse **Input Data (Publisher) Connectors**.

12. Specify a state and event type for the Trades window:

- Expand **State and Event Type**.
- In the **Window state and index** drop-down list, select **Stateless (pi\_EMPTY)**.
- Select the **Accept only "Insert" events** check box.

13. Expand **Input Streams** on the **Windows** pane on the left and drag another Source window to the workspace. The right pane displays the Source window's properties. Configure this window to receive information about stock market traders.

14. Specify a name for the Source window: In the right pane, in the **Name** field, change the default name to `Traders`.

15. Specify an output schema for the Traders window:

- In the right pane, click .
- Click .
- The Output Schema window appears.
- Click  to add a row to the schema table. After you add a row, click  again to add the next row.

Enter the following values:

Key	Name	Type
Y	ID	Int64
N	name	String

- Click **OK**.

16. Configure the Traders window to receive information from a file called `traders.csv` that contains details of stock market traders. You can find this example CSV file in the `trades_xml` folder in the `examples` directory. To add a connector to this CSV file:
    - In the right pane, click .
    - Expand **Input Data (Publisher) Connectors**.
    - Click .
    - The Connector Configuration window appears.
    - In the **Name** field, replace the default value with `TradersConnector`.
    - In the **Fsname** field, enter the path to the CSV file. For example, you might enter `/opt/sas/viya/home/SASEventStreamProcessingEngine/<release>/examples/xml/trades_xml/traders.csv`. Replace `<release>` with the release number in your installation directory path.
    - In the **Fstype** drop-down list, select `csv`.
    - Click **OK**.
  17. Specify a state and event type for the Traders window:
    - Expand **State and Event Type**.
    - In the **Window state and index** drop-down list, select **Stateless (pi\_EMPTY)**.
    - Select the **Accept only "Insert" events** check box.
  18. Expand **Transformations** on the **Windows** pane on the left and drag a Filter window to the workspace. Configure this window to identify large trades. In this example, a trade is regarded as a large trade if the quantity of stock traded equals or exceeds 100.
  19. Click the newly created Filter window on the workspace. The right pane displays the Filter window's properties.
  20. Specify a name for the Filter window: In the **Name** field, change the default name to `LargeTrades`.
  21. Specify a filter expression for the LargeTrades window:
    - Expand **Filter**.
    - In the **Expression** field, enter `quantity >= 100`
    - Collapse **Filter**.
  22. Configure the LargeTrades window's state:
    - Expand **State**.
    - In the **Window state and index field**, select **Stateless (pi\_EMPTY)** from the drop-down list.
  23. Connect the Trades window to the LargeTrades window with an edge:
    - Position the cursor over the anchor point at the bottom of the Trades window so that the anchor point color changes to white.
    - Click the white anchor point, hold the mouse button down, and draw a line to the anchor point in the LargeTrades window.
- The LargeTrades window now accepts trades from the Trades window.

24. Expand **Transformations** on the **Windows** pane on the left and drag a Join window to the workspace. Configure this window to match large trades with the traders who made those trades.
25. Specify a name and description for the Join window: In the right pane, in the **Name** field, change the default name to `AddTraderName`.
26. Connect the `LargeTrades` window to the `AddTraderName` window with an edge.  
The `AddTraderName` window now accepts trades from the `LargeTrades` window.
27. Connect the `Traders` window to the `AddTraderName` window with an edge.  
The `AddTraderName` window now accepts trader names from the `Traders` window.
28. Click the `AddTraderName` window on the workspace.  
The right pane displays the `AddTraderName` window's properties.
29. Confirm the `AddTraderName` window's configuration settings:
  - In the right pane, expand **Settings** and notice that the `LargeTrades` window is regarded as the left window and the `Traders` window is regarded as the right window. This is due to the order in which you added the edges.
  - In the **Output field calculation method** field, confirm that **Select fields** is selected from the drop-down list.  
Selecting the **Select fields** option ensures that, as new input events arrive, join non-key fields are calculated using a join selection string. This selection string is a one-to-one mapping of input fields to join fields.
  - Collapse **Settings**.
30. Configure the `AddTraderName` window's join conditions:
  - Expand **Join Conditions**.
  - In the **Join Conditions** section, click  to add a row to the table.
  - Click the cell in the Left: `LargeTrades` column, and select **traderID**.
  - Click the cell in the Right: `Traders`: column, and select **ID**.
  - Collapse **Join Conditions**.
31. Confirm the `AddTraderName` window's join criteria:
  - Expand **Join Criteria** if it is not already expanded.
  - Confirm that the **Join Type** drop-down list has a default value of **LeftOuter**.
  - Select the **Set the "no-regenerates" option** check box.
  - Collapse **Join Criteria**.
32. Configure the `AddTraderName` window's state:
  - Expand **State**.
  - In the **Window state and index** field, select **Stateless (pi\_EMPTY)** from the drop-down list.

33. Specify a schema for the AddTraderName window:

- In the right pane, click .
- Click .

The Edit Output Schema window appears. Use this window to configure the fields as shown in the following table. The schema fields that are required have already been defined previously. Click  to open the Copy Fields from Input Schema window. Select the following schema fields and click **OK**.

Window	Field	Type
LargeTrades	security	String
LargeTrades	quantity	Int32
LargeTrades	price	Double
LargeTrades	traderID	Int64
LargeTrades	time	Timestamp
Traders	name	String

The Edit Output Schema window displays the fields that you selected.

- Click **OK**.  
You are returned to the Edit Output Schema – Non-Key Fields window.
- Click **OK**.

34. In the right pane, click .

35. Expand **Transformations** on the **Windows** pane on the left and drag a Compute window to the workspace.

The right pane displays the Compute window's properties.

Configure this window to compute the total cost of the large trades.

36. Specify a name for the Compute window: In the **Name** field, change the default name to `TotalCost`.

37. Connect the AddTraderName window to the TotalCost window with an edge.

The TotalCost window now accepts trades from the AddTraderName window.

38. Click the TotalCost window to display the window's properties in the right pane again.

39. Configure the TotalCost window's state:

- In the right pane, expand **State**.
- In the **Window state and index** field, select **Stateless (pi\_EMPTY)** from the drop-down list.

40. Specify an output schema for the TotalCost window:

- In the right pane, click .
- Click . The Output Schema window appears.
- Click  to add a row to the schema table.

Enter the following values:

**Note**

Alternatively, you can copy the schema fields that you have previously defined. Click  to open the Copy Fields from Input Schema window. Select the schema fields that you want to copy and click **OK**. If you copied schema fields you previously created, you must still manually enter all new fields and their values.

Key	Field Name	Type	Expression
Y	tradeID	String	(not applicable)
N	security	String	security
N	quantity	Int32	quantity
N	Price	Double	Price
N	totalCost	Double	price*quantity
N	traderID	Int64	traderID
N	Time	Timestamp	Time
N	Name	String	Name

- Click **OK**.

41. Expand **Transformations** on the **Windows** pane on the left and drag an Aggregate window to the workspace.

The right pane displays the Aggregate window's properties.

Configure this window to compute the total cost of the large trades.

42. In the right pane, click .

43. Specify a name for the Aggregate window: In the **Name** field, change the default name to `BySecurity`.

44. Connect the TotalCost window to the BySecurity window with an edge.

The BySecurity window now accepts trades from the TotalCost window.

45. Click the BySecurity window to display the Aggregate window's properties in the right pane again.

46. Specify an output schema for the BySecurity window:

- In the right pane, click .
- Click .  
The Output Schema window appears.
- Click  to add a row to the schema table. Enter the following values:

Key	Field Name	Type	Aggregate function	Parameters
Y	Security	String	(not applicable)	(not applicable)
N	quantityTotal	Double	ESP_aSum	quantity
N	costTotal	Double	ESP_aSum	totalCost

- Click **OK**.

47. The model is now complete. Click  to save your model.

48. Click .

A new page called **Test: Trades** appears.

49. In the **Streaming Server** drop-down list, select the Streaming Server on which you want to test the model.

50. Click .

The results for each window appear on separate tabs:

- The **Trades** tab lists the securities transactions
- The **Traders** tab lists the traders
- The **LargeTrades** tab lists the large trades
- The **AddTraderName** tab lists the large trades and includes an additional column that shows trader names
- The **TotalCost** tab includes an additional column that shows the total cost of each transaction. You can use this information to identify high-value transactions.
- The **BySecurity** tab shows all the inserts, deletes, and update blocks for the large trades. The newest event is shown at the top of the table. The total cost of transactions for each security is displayed: 601300 for IBM and 91950 for SAP.

---

#### Note

If the table is empty, check that the publisher connectors for the Trades and Traders windows are set correctly to point to the CSV files.

---

51. To stop the test, click .

The project stops and then unloads from the Streaming Server.

## 2.13 Example: Streaming Analytics with Scoring and Training

### 2.13.1 Overview

This example demonstrates the use of the machine learning algorithm *k-means*, which is often used for cluster analysis in data mining. The algorithm assigns data points to their nearest cluster centroid. Each cluster centroid is then recomputed based on the average of the cluster's data points. In *k-means* clustering, the input event is augmented with a cluster number. This indicates the cluster that the observation falls into.

This example uses two files:

- The XML file (model.xml) associated with this example.
- input.csv is an input file. This file contains the events to be scored.

---

#### Note

The CSV data and model XML code that are used in this example are available within your installation, typically in the following location:  
**/opt/esa/<release>/examples/analytics/analytics\_kmeans**. Replace *<release>* with the release number in your installation directory path.

---

### 2.13.2 Project Details

This project contains three windows:

- w\_source is a Source window. This is where events from the input.csv file enters the model to be scored.
- w\_training is a Train window. This window generates and periodically updates the *k-means* model.
- w\_scoring is a Score window. This is where the events are scored.

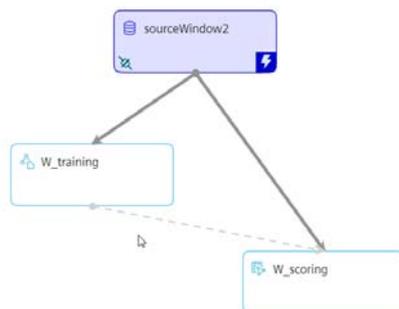


Figure 2-28 Diagram of the Streaming Analytics Model with Scoring and Training

### 2.13.3 Example Steps

To complete this example, follow these steps:

1. On the **Projects** page, click .  
The New Project window appears.
2. In the New Project window, do the following:
  - In the **Name** field, enter `Scoring_and_Training`.
  - In the **Description** field, enter: `This model demonstrates the use of the K-means machine learning algorithm for clustering.`
  - Click **OK**.  
If you do not currently have any Streaming Servers configured, you are prompted to decide whether you want to configure a streaming Server now.

---

#### Note

It is assumed that you do not have any Streaming Servers configured. If you already have Streaming Servers configured, go to step 5.

---

3. Click **Yes** to configure a streaming Server now. The Streaming Server Properties window appears.
4. Configure a streaming Server:
  - In the **Name** field, enter a name to identify the new Streaming Server that you want to create.
  - In the **Host** field, enter the host name of the new Streaming Server.
  - In the **HTTP port** field, enter the new Streaming Server's HTTP publish/subscribe port.
  - If required, in the **Description** field, enter a description of the new Streaming Server.
  - If required, in the **Tags** field, enter any keywords that describe the Streaming Server and then press Enter.
  - If required, click **Edit** to change the setting for the **Authentication** field:  
**None**: This is the default option.
  - If required, select the **Connect using SSL** check box. Selecting this option is relevant only if the Streaming Server is configured to require SSL encryption.
  - If required, select the **Enable server logging** check box to enable logging on the Streaming Server.
  - If required, in the **Number of messages to retain** field, change the default number of messages that are retained by the Streaming Server log. The default is 10,000 messages.
  - Click **OK**.  
Your project is created with a default set of properties.
5. In the right pane, configure your project's properties:
  - Expand **Attributes**.
  - Select the **Compress open patterns** check box.

2.13 Example: Streaming Analytics with Scoring and Training

6. Expand **Input Streams** on the **Windows** pane on the left and drag a Source window to the workspace. The right pane displays the Source window's properties.
7. Enter a name for the Source window: In the right pane, in the **Name** field, change the default name to `W_source`.
8. Configure `W_source` window's event type:
  - In the right pane, expand **State and Event Type**.
  - Select the **Accept only "Insert" events** check box.
9. Specify an output schema for the `W_source` window:
  - In the right pane, click .
  - Click . The Output Schema window appears.
  - Click  to add a row to the schema table. After you add a row, click  again to add the next row. Enter the following values in the rows:

Key	Field Name	Type
Y	id	Int64
N	x_c	Double
N	y_c	Double

- Click **OK**.
10. The `W_source` window will stream events from a file called `input.csv` that contains example data. You can find this CSV file in the `analytics_kmeans` folder in the `/examples/analytics` directory. To add a connector to this CSV file:
    - In the right pane, click .
    - If it is not already selected, click the `W_source` window to select it.
    - Expand **Input Data (Publisher) Connectors** and click . The Publisher Connectors window appears.
    - In the **Name** field, replace the default value with `Source_File`.
    - In the **Fsname** field, enter the path to the CSV file. For example, you might enter `/opt/sas/viya/home/SASEventStreamProcessingEngine/<release>/examples/analytics/`

analytics\_kmeans/input.csv. Replace *<release>* with the release number in your installation directory path.

- In the **Fstype** drop-down list, select **csv**.
  - Configure the Source\_File connector's properties:
  - Click **All properties**.  
The All Properties window appears.
  - Select **true** from the drop-down list in the **Value** field of the **transactional** property.
  - Enter **1** in the **Value** field of the **blocksize** property.
  - Click **OK**.
  - Click **OK**.
  - Collapse **Input Data (Publisher) Connectors**.
11. Configure an output rule for the W\_source window:
- Expand **Output Rules**.
  - Select the **Only output "insert" events** check box.
12. Expand **Analytics** on the **Windows** pane on the left and drag a Train window to the workspace.  
This window uses the *k-means* algorithm to periodically generate a new clustering model. The right pane displays the Train window's properties.
13. Specify a name for the Train window: In the right pane, in the **Name** field, change the default name to `W_training`.
14. Connect the W\_source window to the W\_training window with an edge:
- Position the cursor over the anchor point at the bottom of the W\_source window so that the anchor point color changes to white.
  - Click the white anchor point, hold the mouse button down, and draw a line to the anchor point in the W\_training window.
- The W\_training window now accepts events from the W\_source window.
15. If it is not already selected, click the W\_training window on the workspace to select it.
16. If it is not already expanded, expand **Settings**.
17. In the **Algorithm** drop-down list, select **KMEANS**.

2.13 Example: Streaming Analytics with Scoring and Training

18. Expand **KMEANS**:

- Expand **Parameters**.
- In the **nClusters** field, confirm that the default number of clusters is set to 2.
- In the **initSeed** field, enter 1 to specify the random seed that is used during initialization when each point is assigned to a random cluster.
- In the **dampingFactor** field, confirm that the damping factor's default value for old data points is set to 0.8.
- In the **fadeOutFactor** field, confirm that the default value for determining whether an existing cluster is fading out is set to 0.05.
- In the **disturbFactor** field, confirm that the default value for the disturbance factor when splitting a cluster is set to 0.01.
- In the **nInit** field, confirm that the default value for the number of data events that are used during initialization is set to 50.
- In the **velocity** field, enter 5 to specify the number of events that arrive at a single timestamp.
- In the **commitInterval** field, confirm that the default value for the number of timestamps to elapse before committing a model to downstream scoring is set to 25.
- Collapse **Parameters**.
- Expand **Input Map**.
- In the **Field** column in the table, click the row in the table twice and select **x\_c** and **y\_c** from the drop-down list. These variables are to be used in the clustering.

19. Expand **Analytics** on the **Windows** pane on the left and drag a Score window to the workspace.

The right pane displays the Score window's properties.

This window scores incoming events.

20. Enter a name for the Score window: In the right pane, in the **Name** field, change the default name to `w_scoring`.

21. Specify a schema for the W\_scoring window:

- In the right pane, click .
- Click .  
The Output Schema window appears.
- Click  to add a row to the schema table. After you add a row, click  again to add the next row.

Enter the following values in the rows:

Key	Field Name	Type
Y	id	Int64
N	x_c	Double
N	y_c	Double
N	seg	Int32
N	min_dist	Double
N	model_id	Int64

22. Click **OK**.

23. Connect the W\_source window to the W\_scoring window with an edge.

The W\_scoring window can now score events that originate from the W\_source window.

24. Configure the settings for the `W_scoring` window:

- Click the `W_scoring` window on the workspace.
- Expand **Settings** if it is not already expanded.
- Specify an algorithm to use to score incoming events:
- In the **Configured algorithms** field, click  .  
The Configured Algorithms window appears.
- Select the **KMEANS** check box.
- Click **OK**.

– Expand **KMEANS**.

– Configure an input map:

Expand **Input Map**.

In the **Field** column in the table, click the row in the table twice and select `x_c` and `y_c` from the drop-down list. These variables are to be used in the clustering.

Collapse **Input Map**.

– Configure an output map:

Expand **Output Map**.

Specify the output variable name in the output schema that stores the cluster label. In the **labelOut** row, click the **Name** field twice to display the drop-down list and select **seg**.

Specify the output variable name in the output schema that stores the distance to the nearest cluster. In the **minDistanceOut** row, click the **Name** field twice to display the drop-down list and select **min\_dist**.

Specify the output variable name in the output schema that stores the ID of the model from which the score is computed. In the **modelIdOut** row, click the **Name** field twice to display the drop-down list and select **model\_id**.

25. Connect the `W_training` window to the `W_scoring` window with an edge.

26. Configure the project's continuous query:

- Click .
- In the right pane, in the **Name** field, change the default name to `scoretrain_cq`.
- Expand **Debugging**.
- In the **Enable trace server logging for this query** field, select **W\_scoring** and **W\_training**.

27. Click .

28. Click  **Enter Test Mode** .

A new page called **Test: Scoring\_and\_Training** appears.

29. In the **Test Server** drop-down list, select the Streaming Server on which you want to test the model.

30. Click .

The results for each window appear in separate tabs:

- The **w\_source** tab displays events to be scored
- The **w\_training** tab displays the generated clustering model using the k-means algorithm
- The **w\_scoring** tab displays the scored events

31. To stop the test, click .

## 2.14 Example: Geofence

### 2.14.1 Overview

This example creates a model that displays a list of wanted vehicles found in close proximity of critical infrastructure sites. The model performs the following actions:

- Streams a list of vehicles, including vehicle locations
- Streams a list of vehicles that are included on a vehicle watch list
- Streams a list of critical infrastructure sites, including site locations
- Processes the list of vehicles and attempts to match any wanted vehicles that are in close proximity to critical infrastructure sites
- Produces a list of wanted vehicles found in close proximity to critical infrastructure sites
- Note: The CSV data and model XML code that are used in this example are available within your installation, typically in the following location:  
`/opt/esa/<release>/examples/xml/geofence2.xml`. Replace `<release>` with the release number in your installation directory path.

---

#### Note

The CSV data and model XML code that are used in this example are available within your installation, typically in the following location:

`/opt/esa/<release>/examples/xml/geofence2.xml`. Replace `<release>` with the release number in your installation directory path.

---

### 2.14.2 Project Details

This project contains six windows:

The ANPR window is a Source window. This is where a list of all vehicles within close proximity of critical infrastructure sites from the `anpr.csv` file enter the model.

The VehicleWatchList window is a Source window. This is where a list of all vehicles on the vehicle watch list from the `wantedvehicle.csv` file enter the model.

2.14 Example: Geofence

The CriticalInfrastructure window is a Source window. This is where a list of sites that contain critical infrastructure from the infrastructure.csv file enter the model.

The WantedVehicleMatch window is a Join window. This is where a list of all vehicles found within close proximity of critical infrastructure sites, and a list of all wanted vehicles are merged into one list.

The Geofence window is a Geofence window. This is where geofencing information that relates to the matched vehicles enter the model.

The GeofenceMatches window is a Filter window. This is where the geofencing information is filtered.

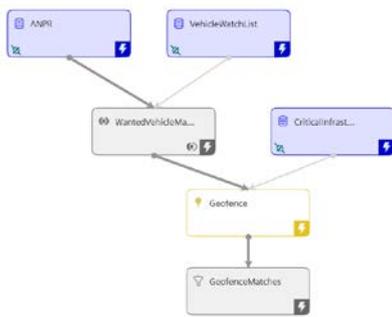


Figure 2-29 Diagram of the Geofence Model

### 2.14.3 Example Steps

To complete this example, follow the steps below:

1. On the **Projects** page, click . The New Project window appears.
2. In the New Project window, do the following:
  - In the **Project name** field, enter `geofence_demo`.
  - In the **Description** field, enter a description. Here is an example: `This model can be used to identify wanted vehicles found in close proximity to critical infrastructure sites.`
  - Click OK.

If you do not currently have any Streaming Servers configured, you are prompted to decide whether you want to configure a streaming Server now.

---

**Note**

It is assumed that you do not have any Streaming Servers configured. If you already have Streaming Servers configured, skip to step 5.

---

3. Click **Yes** to configure a streaming Server now. The Streaming Server Properties window appears.

4. Configure a streaming Server:
  - In the **Name** field, enter a name to identify the new test server to create.
  - In the **Host** field, enter the host name or the IP address of the test server.
  - In the **HTTP port** field, enter the test server's HTTP publish/subscribe port.
  - If required, in the **Description** field, enter a description of the new Streaming Server.
  - If required, in the **Tags** field, enter any keywords that describe the Streaming Server and then press Enter.
  - If required, click **Edit** to change the setting for the **Authentication** field:
    - None:** This is the default option.
  - If required, select the **Connect using SSL** check box. Selecting this option is relevant only if the test server is configured to require SSL encryption.
  - If required, select the **Enable server logging** check box to enable logging on the Streaming Server.
  - If required, in the **Number of messages to retain** field, change the default number of messages that are retained by the Streaming Server log. The default is 10,000 messages.
  - Click **OK**.  
Your project is created with a set of default properties.
5. Expand **Input Streams** on the **Windows** pane on the left and drag a Source window to the workspace.  
The right pane displays the Source window's properties.
6. Specify a name for the Source window: In the right pane, in the **Name** field, change the default name to `ANPR`.
7. Configure the ANPR window to accept only "Insert" events and to automatically generate the key field:
  - Expand **State and Event Type**.
  - Select the **Accept only "Insert" events** check box.
  - Select the **Automatically generate the key field** check box.

- 8. Specify an output schema for the ANPR window:
  - In the right pane, click .
  - Click . The Output Schema window appears.
  - Click  to add a row to the schema table. After you add a row, click  again to add the next row. Enter the following values:

Key	Field Name	Type
N	vrn	String
N	lat	Double
N	long	Double
N	date	Timestamp
Y	pkey	String

- Click **OK**.
- 9. The ANPR window streams a list of vehicles from a file called anpr.csv that contains example data. You can find this CSV file in the `geofence2_xml` folder in the `examples` directory. To add a connector to this CSV file:
  - In the right pane, click .
  - Expand **Input Data (Publisher) Connectors** and click .
  - The Connector Configuration window appears.
  - In the **Name** field, replace the default value with `anpr_csv_read`.
  - In the **Fsname** field, enter the path to the CSV file. For example, you might enter `/opt/sas/viya/home/SASEventStreamProcessingEngine/<release>/examples/xml/geofence2_xml/anpr.csv`. Replace `<release>` with the release number in your installation directory path.
  - In the **Fstype** drop-down list, select `csv`.
  - Configure the `anpr_csv_read` connector’s properties:
    - Click **All properties**. The All Properties window appears.
    - Enter `%Y-%m-%d %H:%M:%S` in the Value field of the **dateformat** property.
    - Enter `1` in the Value field of the **header** property.
    - Select `true` from the drop-down list in the Value field of the **ignorecsvparseerrors** property.
    - Select `true` from the drop-down list in the Value field of the **noautogenfield** property.
    - Click **OK**.
  - Click **OK**.
- 10. Expand **Input Streams** on the **Windows** pane on the left and drag another Source window to the workspace. The right pane displays the Source window’s properties.

11. Specify a name for the Source window: In the right pane, in the **Name** field, change the default name to `VehicleWatchList`.

12. Specify an output schema for the `VehicleWatchList` window:

- In the right pane, click .
- Click .  
The Output Schema window appears.
- Click  to add a row to the schema table. Enter the following values:

Key	Field Name	Type
Y	vrm	String

- Click **OK**.

13. The `VehicleWatchList` window streams a list of wanted vehicles from a file called `wantedvehicle.csv` that contains example data. You can find this CSV file in the `geofence2_xml` folder in the `examples` directory. To add a connector to this CSV file:

- In the right pane, click .
- Expand **Input Data (Publisher) Connectors** and click .
- In the **Name** field, replace the default value with `vehicle_watchlist`.
- In the **Fsname** field, enter the path to the CSV file. For example, you might enter `/opt/sas/viya/home/SASEventStreamProcessingEngine/<release>/examples/xml/geofence2_xml/wantedvehicle.csv`. Replace `<release>` with the release number in your installation directory path.
- In the **Fstype** drop-down list, select **csv**.
- Configure the `vehicle_watchlist` connector's properties:
  - Click **All properties**.  
The All Properties window appears.
  - Enter `1` in the **Value** field of the **header** property.
  - Click **OK**.
- Click **OK**.

14. Expand **Transformations** on the **Windows** pane on the left and drag a Join window to the workspace. The right pane displays the Join window's properties.

15. Specify a name for the Join window: In the right pane, in the **Name** field, change the default name to `WantedVehicleMatch`.

16. Connect the ANPR window to the `WantedVehicleMatch` window with an edge:

- Position the cursor over the anchor point at the bottom of the ANPR window so that the anchor point color changes to white.
- Click the white anchor point, hold the left mouse button down, and draw a line to the anchor point in the `WantedVehicleMatch` window.  
The `WantedVehicleMatch` window now accepts values from the ANPR window.

17. Connect the VehicleWatchlist window to the WantedVehicleMatch window with an edge.  
The WantedVehicleMatch window now accepts values from the VehicleWatchlist window.

---

**Note**

Each window in your model displays specific icons that represent window properties. For example, if a Source window contains a publisher connector, the window displays the corresponding publisher connector icon. For more information about window icons, see **Window Icons**. The WantedVehicleMatch window displays an error icon  indicating that an invalid join type has been set. The occurrence of this error is expected behavior and will be resolved later when you set a valid join type.

---

18. Click the WantedVehicleMatch window in the workspace.  
The right pane displays the Join window's properties.
19. Examine the calculation method for the WantedVehicleMatch window's output fields:
  - In the right pane, expand **Settings**.
  - Inspect the **Left window** and **Right window** fields. Notice that the ANPR window is regarded as the left window and the VehicleWatchList window is regarded as the right window. This is due to the order in which you added the edges.
  - In the **Output field calculation method** field, confirm that **Select fields** is selected from the drop-down list.  
As a result of choosing the **Select fields** option, as new input events arrive, join non-key fields are calculated using a join selection string. This selection string is a one-to-one mapping of input fields to join fields.
  - Collapse **Settings**.
20. Configure the WantedVehicleMatch window's join criteria:
  - If it is not already expanded, expand **Join Criteria**.
  - In the **Join Type** drop-down list, select **Inner**.
  - Collapse **Join Criteria**.
21. Configure the WantedVehicleMatch window's join conditions:
  - Expand **Join Conditions**.
  - In the **Join Conditions** field, click  to add a join condition.
  - Click the cell in the Left: ANPR column twice, and select **vrn** from the drop-down list.
  - Click the cell in the Right: VehicleWatchList column twice, and select **vrn** from the drop-down list.
22. Specify an output schema for the WantedVehicleMatch window:
  - In the right pane, click .
  - Click .The Edit Output Schema window appears.  
Use this window to configure the fields as shown in the following table. As the schema

fields required have already been defined previously, click  to open the Copy Fields from Input Schema window. Select the following schema fields and click **OK**.

Window	Field	Type
ANPR	vrn	String
ANPR	lat	Double
ANPR	long	Double
ANPR	date	TimeStamp

The Edit Output Schema window displays the fields that you selected.

- Click **OK**.

23. Click .

24. Expand **Input Streams** on the **Windows** pane on the left and drag another Source window to the workspace. The right pane displays the Source window's properties.

25. Specify a name and description for the Source window:

- In the right pane, in the **Name** field, change the default name to `CriticalInfrastructure`.

26. Specify an output schema for the CriticalInfrastructure window:

- In the right pane, click .
- Click . The Output Schema window appears.
- Click  to add a row to the schema table. After you add a row, click  again to add the next row. Enter the following values:

Key	Field Name	Type
Y	name	String
N	lat	Double
N	long	Double
N	location	String
N	county	String
N	region	String
N	type	String
N	capacity	String
N	opened	String
N	closed	String
N	demolished	String
N	notes	String

- Click **OK**.

27. The CriticalInfrastructure window will stream a list of sites that contain critical infrastructure from a file called `infrastructure.csv` that contains example data. You can find this CSV file in the `geofence2_xml` folder in the `examples` directory. To add a connector to this CSV file:
  - In the right pane, click .
  - Expand **Input Data (Publisher) Connectors**.
  - Click .
  - The Connector Configuration window appears.
  - In the **Name** field, replace the default value with `infrastructure_csv_reader`.
  - In the **Fsname** field, enter the path to the CSV file. For example, you might enter `/opt/sas/viya/home/SASEventStreamProcessingEngine/<release>/examples/xml/geofence2_xml/infrastructure.csv`. Replace `<release>` with the release number in your installation directory path.
  - In the **Fstype** drop-down list, select `csv`.
  - Configure the `infrastructure_csv_reader` connector's properties:
    - Click **All properties**.
    - The All Properties window appears.
    - Enter `1` in the **Value** field of the **header** property.
    - Select `true` from the drop-down list in the **Value** field of the **ignorecsvparseerrors** property.
    - Click **OK**.
  - Click **OK**.
28. Expand **Utilities** on the **Windows** pane on the left and drag a Geofence window to the workspace.
  - The right pane displays the Geofence window's properties.
29. Specify a name for the Geofence window: In the right pane, in the **Name** field, change the default name to `Geofence`.
30. Connect the `WantedVehicleMatch` window to the Geofence window with an edge.
31. Connect the `CriticalInfrastructure` window to the Geofence window with an edge.
32. Click the Geofence window in the workspace.
  - The right pane displays the Geofence window's properties.
33. Configure the Geofence window's positional settings:
  - Expand **Positions**.
  - In the **X coordinate** field, select `long` from the drop-down list.
  - In the **Y coordinate** field, select `lat` from the drop-down list.
  - In the **Default lookup distance (meters)** field, enter `20`.
  - Collapse **Positions**.

34. Configure the Geofence window's geometric settings:
  - Expand **Geometries**.
  - In the table, in the **X coordinate** row, select **long** from the **Field** drop-down list.
  - In the table, in the **Y coordinate** row, select **lat** from the **Field** drop-down list.
  - In the **Default radius (meters)** field, enter **100**.
  - Collapse **Geometries**.
35. Configure the Geofence window's geofence algorithm properties:
  - Expand **Geofence Algorithm Properties**.
  - Select the **Record invalid geometries in the standard output log** check box.
  - Collapse **Geofence Algorithm Properties**.
36. Configure the Geofence window's output map properties:
  - Expand **Output Map**.
  - In the **Geometry ID** field, enter `geoid`.
  - In the **Event number** field, enter `eventnum`.
  - Collapse **Output Map**.
37. Expand **Transformations** on the **Windows** pane on the left and drag a Filter window to the workspace. The right pane displays the Filter window's properties.
38. Specify a name for the Filter window: In the **Name** field, change the default name to `GeofenceMatches`.
39. Configure a subscribe connector for the GeofenceMatches window:
  - Expand **Subscriber Connectors**.
  - Click .
  - The Connector Configuration window appears.
  - In the **Name** field, enter `sub`.
  - Select the **Snapshot** check box.
  - In the **Fsname** field, enter the path to the output file: **result.out**. For example, you might enter `/opt/esa/<release>/examples/xml/geofence2_xml/result.out`. Replace `<release>` with the release number in your installation directory path.
  - In the **Fstype** drop-down list, select **csv**.
  - Click **OK**.
40. Specify a filter expression for the GeofenceMatches window:
  - Expand **Filter**.
  - In the **Expression** field, enter `geoid != ''`
  - Collapse **Filter**.
41. Connect the Geofence window to the GeofenceMatches window with an edge.

42. Configure your model's connector orchestration:

- Click .
- In the right pane, expand **Connector Orchestration**.
- Click  below the **Connector groups** label.  
The Connector groups window appears.
- In the **Name** field, enter `sub1`.
- Click  below the **Connectors** label.
- In the Connector column, click the newly created row and select `cq1/GeofenceMatches/sub` from the drop-down list.
- In the Target state column, select **Running** from the drop-down list.
- Click **OK**.
- Click  below the **Connector groups** label.  
The Connector groups window appears.
- In the **Name** field, enter `pub1`.
- Click  below the **Connectors** label.
- In the Connector column, click the newly created row and select `cq1/ANPR/anpr_csv_read` from the drop-down list.
- In the Target state column, confirm that **Finished** is selected from the drop-down list.
- Click **OK**.
- Click  below the **Connector groups** label.  
The Connector groups window appears.
- In the **Name** field, enter `pub2`.
- Click  below the **Connectors** label.
- In the Connector column, click the newly created row and select `cq1/CriticalInfrastructure/ infrastructure_csv_reader` from the drop-down list.
- In the Target state column, confirm that **Finished** is selected from the drop-down list.
- Click **OK**.
- Click  below the **Connector groups** label.  
The Connector groups window appears.
- In the **Name** field, enter `pub3`.
- Click  below the **Connectors** label.
- In the Connector column, click the newly created row and select `cq1/Vehiclewatchlist/ vehicle_watchlist` from the drop-down list.
- In the Target state column, confirm that **Finished** is selected from the drop-down list.
- Click **OK**.
- Configure the dependency rules. Click  below the **Dependency rules** label. After you add a row, click  again to add the next row.

Enter the following values in the rows:

Row	Controlling Group	Dependent Group
1	sub1	pub1
2	pub1	pub2
3	pub2	pub3

- In the right pane, click  .  
The XML Editor appears.
- Locate the following line in the XML code: `<edge source="sub1" target="pub1" />`
- Amend this line to the following: `<edge source="sub1" target="pub1 pub2 pub3" />`

43. Configure your model's threading level:

- In the right pane, click  .
- In the right pane, expand **Attributes**.
- In the **Threads** field, enter 8.

44. The model is now complete. Click  to save your model.

45. Click  **Enter Test Mode** .

A new page called **Test: geofence\_demo** appears.

46. In the **Streaming Server** drop-down list, confirm that the Streaming Server on which you want to test the model is selected. If the appropriate Streaming Server is not selected, select it from the drop-down list.

47. Click  **Run Test** .

The results for each window appear on separate tabs:

- The **ANPR** tab lists all vehicles within close proximity of critical infrastructure sites
- The **VehicleWatchlist** tab lists all vehicles on the vehicle watch list
- The **WantedVehicleMatch** tab combines a list of all vehicles found within close proximity of critical infrastructure sites with a list of all wanted vehicles.
- The **Geofence** tab lists the geofencing information that relates to the matched vehicles
- The **CriticalInfrastructure** tab lists sites that contain critical infrastructure
- The **GeofenceMatches** tab shows any wanted vehicles found within close proximity of critical infrastructure sites

---

#### Note

If the table is empty, check that the publisher connectors for the ANPR, VehicleWatchList, and CriticalInfrastructure windows are set correctly to point to the CSV files.

---

48. To stop the test, click  **Stop** .

The project stops and then unloads from the Streaming Server.

## 2.15 Example: Unifying Multiple Input Streams

### 2.15.1 Overview

By following this example, you learn how to create a model that merges one or more streams with the same schema. The model contains two Source windows and one Union window. The two Source windows stream multiple lists of stock prices to the Union window. The Union window is configured with strict enforcement, that is, the key merge from each window must semantically merge cleanly.

This example uses five files listed below:

- The XML file (model.xml) associated with this example.
- input\_sw\_01\_1.csv is an input file. This file contains a list of stock prices.
- input\_sw\_01\_2.csv is an input file. This file contains a list of stock prices.
- input\_sw\_02.csv is an input file. This file contains a list of stock prices.
- output.csv is an output file. When you run the model, the unified list of stock prices is written to the output.csv file.

## 2.15.2 Project Details

This project contains three windows:

- The sourceWindow\_01 window is a Source window. This is where a list of stock prices from the input\_sw\_01\_1.csv file and a list of stock prices from the input\_sw\_01\_2.csv file enter the model.
- The sourceWindow\_02 window is a Source window. This is where a list of stock prices from the input\_sw\_02.csv file enters the model..
- The unionWindow\_strict window is a Union window. This is where the multiple lists of stock prices are merged into one list. This window also writes the results of the merge to the output.csv file.

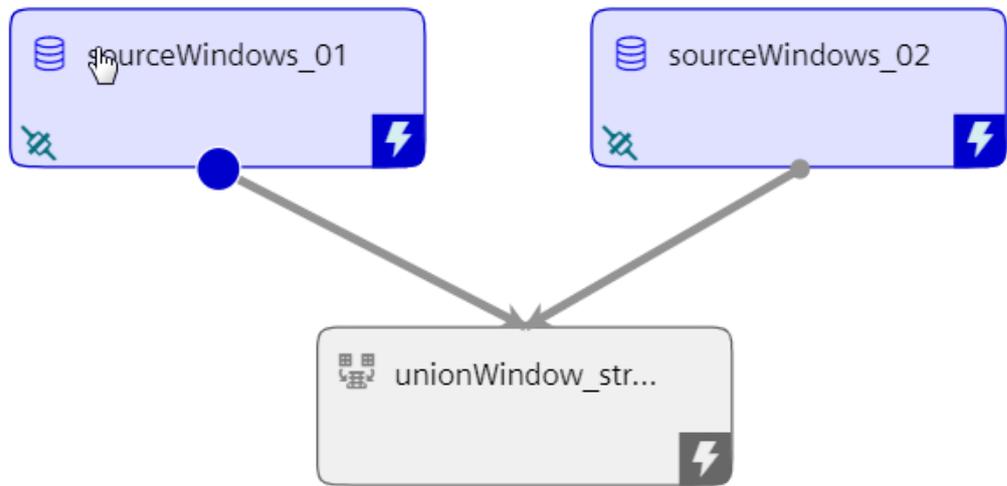


Figure 2-30 Diagram of the Unifying Multiple Input Streams Project

### 2.15.3 Example Steps

Follow these steps:

1. On the **Projects** page, click .  
The New Project window appears.
2. In the New Project window, do the following:
  - In the **Project name** field, enter `union_demo`.
  - In the **Description** field, enter a description. Here is an example: `This model merges three event streams of stock prices using a Union window.`
  - Click **OK**.  
If you do not currently have any Streaming Servers configured, you are prompted to decide whether you want to configure a streaming Server now.

---

#### Note

It is assumed that you do not have any Streaming Servers configured. If you already have Streaming Servers configured, go to step 5.

---

3. Click **Yes** to configure a streaming Server now. The Streaming Server Properties window appears.
4. Configure a streaming Server:
  - In the **Name** field, enter a name to identify the new Streaming Server that you want to create.
  - In the **Host** field, enter the host name or the IP address of the new Streaming Server.
  - In the **HTTP port** field, enter the new Streaming Server's HTTP publish/subscribe port.
  - If required, in the **Description** field, enter a description of the new Streaming Server.
  - If required, in the **Tags** field, enter any keywords that describe the Streaming Server and then press Enter.
  - If required, click **Edit** to change the setting for the **Authentication** field:  
**None**: This is the default option.
  - If required, select the **Connect using SSL** check box. Selecting this option is relevant only if the Streaming Server is configured to require SSL encryption.
  - If required, select the **Enable server logging** check box to enable logging on the Streaming Server.
  - If required, in the **Number of messages to retain** field, change the default number of messages that are retained by the Streaming Server log. The default is 10,000 messages.
  - Click **OK**.  
Your project is created with a set of default properties.

5. Configure your model's threading level:
  - In the right pane, confirm that the project's properties appear. If they do not appear, click .
  - Expand **Attributes**.
  - In the **Threads** field, enter 3.
6. Configure the project's continuous query:
  - Click .
  - In the **Name** field, enter `contquery_01`.
7. Expand **Input Streams** on the **Windows** pane on the left and drag a Source window to the workspace.  
The right pane displays the Source window's properties.
8. Specify a name and description for the Source window:
  - In the right pane, in the **Name** field, change the default name to `SourceWindow_01`.
  - In the **Description** field, enter `This window defines an event stream. All event streams must enter continuous queries by being published or injected into a source window.`
9. Specify an output schema for the `SourceWindow_01` window:
  - On the right toolbar, click .
  - Click .
  - The Output Schema window appears.
  - Click  to add a row to the schema table. After you add a row, click  again to add the next row.  
Enter the following values:

Key	Field Name	Type
Y	ID	Int32
N	symbol	String
N	price	Double

- Click **OK**.
10. The `SourceWindow_01` window streams a list of vehicles from a file called `input_sw_01_1.csv` that contains example data. To add a connector to this CSV file:
    - In the right pane, click .
    - Expand **Input Data (Publisher) Connectors** and click .
    - The Connector Configuration window appears.
    - In the **Name** field, replace the default value with `pub_sw_01_1`.
    - In the **Fsname** field, enter the path to the CSV file. For example, you might enter `/opt/sas/viya/home/SASEventStreamProcessingEngine/<release>/examples/xml/union_`

2.15 Example: Unifying Multiple Input Streams

xml/ input\_sw\_01\_1.csv. Replace *<release>* with the release number in your installation directory path.

- In the **Fstype** drop-down list, select **csv**.
- Configure the pub\_sw\_01\_1 connector's properties:
  - Click **All properties**
  - The All Properties – pub\_sw\_01\_1 window appears.
  - Enter 3 in the **Value** field of the **blocksize** property.
  - Click **OK**.
- Click **OK**.

11. The SourceWindow\_01 window streams a list of vehicles from a file called input\_sw\_01\_2.csv that contains example data. To add a connector to this CSV file:

- Expand **Input Data (Publisher) Connectors** and click .
- The Connector Configuration window appears.
- In the **Name** field, replace the default value with pub\_sw\_01\_2.
- In the **Fsname** field, enter the path to the CSV file. For example, you might enter  
/opt/sas/viya/  
home/SASEventStreamProcessingEngine/<release>/examples/xml/union\_xml/  
input\_sw\_01\_2.csv. Replace *<release>* with the release number in your installation directory path.
- In the **Fstype** drop-down list, select **csv**.
- Configure the pub\_sw\_01\_2 connector's properties:
  - Click **All properties**.
  - The All Properties – pub\_sw\_01\_2 window appears.
  - Enter 3 in the **Value** field of the **blocksize** property.
  - Click **OK**.
- Click **OK**.

12. Expand **Input Streams** on the **Windows** pane on the left and drag a Source window to the workspace. The right pane displays the Source window's properties.

13. Specify a name and description for the Source window:

- In the right pane, in the **Name** field, change the default name to SourceWindow\_02.
- In the **Description** field, enter This window defines an event stream. All event streams must enter continuous queries by being published or injected into a source window.

14. Specify an output schema for the SourceWindow\_02 window:

- On the right toolbar, click .
- Click .  
The Output Schema window appears.
- Click  to add a row to the schema table. After you add a row, click  again to add the next row.

Enter the following values:

Key	Field Name	Type
Y	ID	Int32
N	symbol	String
N	price	Double

- Click **OK**.

15. Configure the SourceWindow\_02 window's state and index:

- On the right toolbar, click .
- Expand **State and Event Type**.
- In the **Window state and index field**, select **Stateful (pi\_HASH)** from the drop-down list.

16. The SourceWindow\_02 window streams a list of vehicles from a file called input\_sw\_02.csv that contains example data. To add a connector to this CSV file:

- Expand **Input Data (Publisher) Connectors** and click .
- The Connector Configuration window appears.
- In the **Name** field, replace the default value with `pub_sw_02`.
- In the **Fsname** field, enter the path to the CSV file. For example, you might enter `/opt/sas/viya/home/SASEventStreamProcessingEngine/<release>/examples/xml/union_xml/input_sw_02.csv`. Replace `<release>` with the release number in your installation directory path.
- In the **Fstype** drop-down list, select **csv**.
- Configure the `pub_sw_02` connector's properties:  
Click **All properties**.  
The All Properties – `pub_sw_02` window appears.  
Enter `3` in the Value field of the `blocksize` property.  
Click **OK**.
- Click **OK**.

17. Expand **Transformations** on the **Windows** pane on the left and drag a Union window to the workspace. The right pane displays the Union window's properties.

18. In the right pane, in the **Name** field, change the default name to `unionWindow_strict`.

2.15 Example: Unifying Multiple Input Streams

19. In the right pane, expand **Union** if necessary.
  - In the **Key merging** field, confirm that **Strict** is selected.
  - Collapse **Union**.
20. Configure the unionWindow\_strict window's state:
  - Expand **State**.
  - In the **Window state and index** field, select **Stateful (pi\_HASH)** from the drop-down list.
21. The unionWindow\_strict window streams data to a CSV file (output.csv) using a subscriber connector. To configure this subscriber connector:
  - Expand **Subscriber Connectors** and click .  
The Connector Configuration window appears.
  - In the **Name** field, replace the default value with sub\_uw. In the **Fsname** field, enter the path to the CSV file. For example, you might enter  
`/opt/sas/viya/home/SASEventStreamProcessingEngine/<release>/examples/xml/union_xml/output.csv`. Replace `<release>` with the release number in your installation directory path.
  - In the **Fstype** drop-down list, select **csv**.
  - Select the **Snapshot** check box.
  - Click **OK**.
22. Connect the sourceWindow\_01 window and the sourceWindow\_02 window to the unionWindow\_strict window with an edge:
  - Position the cursor over the anchor point at the bottom of the window so that the anchor point color changes to white.
  - Click the white anchor point, hold the left mouse button down, and draw a line to the anchor point in the unionWindow\_strict window.

The unionWindow\_strict window now accepts values from the sourceWindow\_01 window and the sourceWindow\_02 window.

23. Configure your model's connector orchestration:

- Click .
- In the right pane, expand **Connector Orchestration**.
- Click  below the Connector groups label.  
The Connector groups window appears.
- In the **Name** field, enter `CG_sub1_1`.
- Click  below the Connectors label.
- In the Connector column, click the newly created row and select **contquery\_01/unionWindow\_strict/ sub\_uw** from the drop-down list.
- In the Target state column, select **Running** from the drop-down list.
- Click **OK**.
- Click  below the Connector groups label.  
The Connector groups window appears.
- In the **Name** field, enter `CG_pub_sw_01_1`.
- Click  below the Connectors label.
- In the Connector column, click the newly created row and select **contquery\_01/sourceWindow\_01/ pub\_sw\_01\_1** from the drop-down list.
- In the Target state column, confirm that **Finished** is selected from the drop-down list.
- Click **OK**.
- Click  below the Connector groups label.  
The Connector groups window appears.
- In the Name field, enter `CG_pub_sw_02`.
- Click  below the **Connectors** label. In the Connector column, click the newly created row and select **contquery\_01/sourceWindow\_02/ pub\_sw\_02** from the drop-down list.
- In the Target state column, confirm that **Finished** is selected from the drop-down list.
- Click **OK**.
- Click  below the Connector groups label.  
The Connector groups window appears.
- In the **Name** field, enter `CG_pub_sw_01_2`.
- Click  below the **Connectors** label.
- In the Connector column, click the newly created row and select **contquery\_01/sourceWindow\_01/ pub\_sw\_01\_2** from the drop-down list.
- In the Target state column, confirm that **Finished** is selected from the drop-down list.
- Click **OK**.
- Configure the dependency rules. Click  below the **Dependency rules** label. After you add a row, click  again to add the next row.  
Enter the following values in the rows:

2.16 Example: Splitting Generated Events across Output Slots

Row	Controlling Group	Dependent Group
1	CG_sub_1_1	CG_pub_sw_01_1
2	CG_pub_sw_01_1	CG_pub_sw_02
3	CG_pub_sw_02	CG_pub_sw_01_2

24. Configure the continuous query’s trace log:

- Click .
- Expand **Debugging**.
- In the **Enable trace server logging for this query** field, select **unionWindow\_strict** from the drop-down list.

25. Click .

26. Click  **Enter Test Mode**.  
 A new page called **Test: union\_demo** appears.

27. In the **Test Server** drop-down list, select the Streaming Server on which you want to test the model.

28. Click  **Run Test**.  
 The results for each window appear on separate tabs:

- The **sourceWindow\_01** tab displays the first event stream
- The **sourceWindow\_02** tab displays the second event stream
- The **unionWindow\_strict** tab displays the unified event stream

29. To stop the test, click  **Stop**.  
 The project stops and is unloaded from the Streaming Server.

## 2.16 Example: Splitting Generated Events across Output Slots

### 2.16.1 Overview

This model enables you to send generated stock market events across a set of output slots. It contains a Source window, a Compute window, and three Copy windows. The Compute window uses an expression to determine what output slot or slots should be used for a newly generated stock market event. The Copy windows connect to the Compute window using different output slots.

Filtering events using window splitters with only one output slot can be more efficient than using multiple Filter windows. This is because the filtering is performed at the window splitter only, rather than at multiple times for each filter. For example, performing an alpha-split across a set of trades results in less data movement and data processing than performing an alpha-split across multiple Filter windows.

This example uses six files listed below:

- The XML file (model.xml) associated with this example.
- input.csv is an input file. This file contains a list of stock trades.

- compute.csv is an output file. When you run the model, the computed fields are written to the compute.csv file.
- cw\_01.csv is an output file. Stock market events from slot 0 are sent to the cw\_01.csv file.
- cw\_02.csv is an output file. Stock market events from slot 1 are sent to the cw\_02.csv file.
- cw\_03.csv is an output file. Stock market events from slot -1 are sent to the cw\_03.csv file.

## 2.16.2 Project Details

This project contains five windows:

- src\_win window is a Source window. This is where a list of securities transactions from the input.csv file enter the model.
- compute\_win window is a Compute window. The computed fields are listed in the compute.csv file.
- cw\_01 window is a Copy window. The window writes the results of the trades allocated to slot 0 to the cw\_01.csv file.
- cw\_02 window is a Copy window. The window writes the results of the trades allocated to slot 1 to the cw\_02.csv file.
- cw\_03 window is a Copy window. The window writes the results of the trades allocated to slot -1 to the cw\_03.csv file.

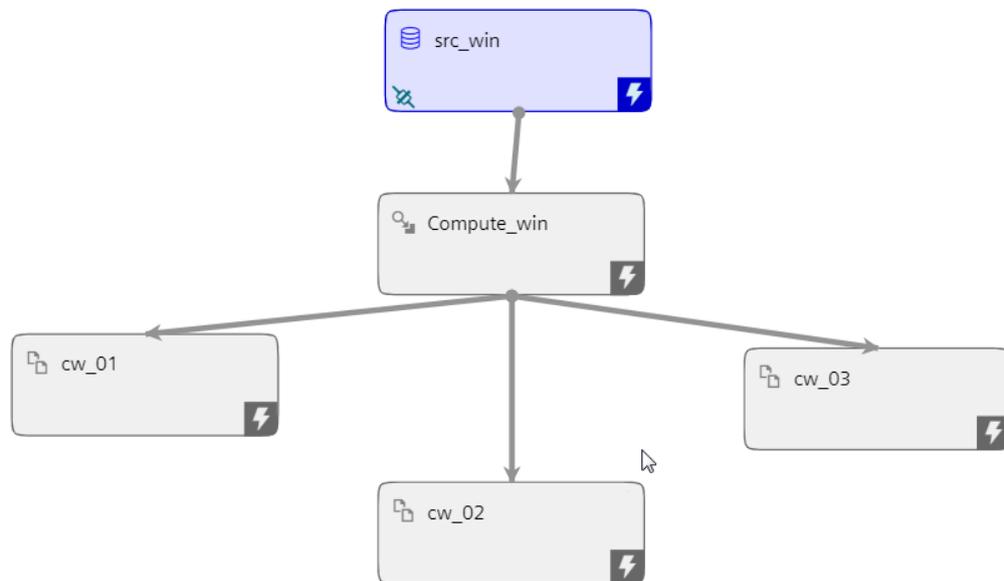


Figure 2-31 Diagram of the Split Generated Events Across Output Slots Project

### 2.16.3 Example Steps

1. On the **Projects** page, click .
2. In the New Project window, do the following:
  - In the **Project name** field, enter `modelingSplitterExp`.
  - Click **OK**.

If no Streaming Servers are currently configured, you are prompted to decide whether you want to configure a streaming Server now.

---

#### Note

It is assumed that no Streaming Servers are currently configured. If some already are, go to step 5.

---

3. Click **Yes** to configure a streaming Server now. The Streaming Server Properties window appears.
4. Configure a streaming Server:
  - In the **Name** field, enter a name to identify the new Streaming Server that you want to create.
  - In the **Host** field, enter the host name or the IP address of the new Streaming Server.
  - In the **HTTP port** field, enter the new Streaming Server's HTTP publish/subscribe port.
  - If required, in the **Description** field, enter a description of the new Streaming Server.
  - If required, in the **Tags** field, enter any keywords that describe the Streaming Server and then press Enter.
  - If required, click **Edit** to change the setting for the **Authentication** field:
    - **None**: This is the default option.
    - **Kerberos**: This option is relevant only if the Streaming Server is configured to require authentication using Kerberos.
    - **OAuth token**: This option is relevant only if the Streaming Server is configured to require authentication using an OAuth token. If you select this option, an additional field appears where you must enter the OAuth token.
    - **Username and password**: This option is relevant only if the Streaming Server is configured to require authentication using a username and password (SASLogon

Services). If you select this option, additional fields appear where you must enter the username and password.

- If required, select the **Connect using SSL** check box. Selecting this option is relevant only if the Streaming Server is configured to require SSL encryption.
- If required, select the **Enable server logging** check box to enable logging on the Streaming Server.
- If required, in the **Number of messages to retain** field, change the default number of messages that are retained by the Streaming Server log. The default is 10,000 messages.
- Click **OK**.

Your project is created with a set of default properties.

5. Configure the project's continuous query:

- Click .
- In the **Name** field, enter **cq\_01**.

6. Expand **Input Streams** on the **Windows** pane on the left and drag a Source window to the workspace. The right pane displays the Source window's properties.

7. Specify a name and description for the Source window:

- In the right pane, in the **Name** field, change the default name to **src\_win**.
- In the **Description** field, enter **This window receives an event stream of stock market trades**.

8. Configure the src\_win window's state and event type:

- In the right pane, expand **State and Event Type**.
- Select Stateful (**pi\_RBTREE**) from the Window state and index drop-down list.

9. Specify an output schema for the src\_win window:

- In the right toolbar, click .
- Click .

The Output Schema window appears.

- Click  to add a row to the schema table. After you add a row, click  again to add the next row.

Enter the following values:

Key	Field Name	Type
Y	ID	Int32
N	symbol	String
N	price	Double

- Click **OK**.

2.16 Example: Splitting Generated Events across Output Slots

10. The src\_win window streams a list of vehicles from a file called input.csv that contains example data. To add a connector to this CSV file:
  - Click  .
  - Expand **Input Data (Publisher) Connectors** and click  .  
The Connector Configuration window appears.
  - In the **Name** field, replace the default value with pub.
  - In the **Fsname** field, enter the path to the CSV file. For example, you might enter  
/opt/sas/viya/  
home/SASEventStreamProcessingEngine/<release>/examples/xml/slot\_exp\_xml/  
input.csv. Replace <release> with the release number in your installation directory path.
  - In the **Fstype** drop-down list, select csv.
  - Configure the pub connector's properties:
  - Click **All properties**.  
The All Properties window appears.
  - Select true in the **Value** field of the **transactional** property.
  - Enter 1 in the **Value** field of the **blocksize** property.
  - Click **OK**.
  - Click **OK**.
11. Expand **Transformations** on the **Windows** pane on the left and drag a Compute window to the workspace.  
The right pane displays the Compute window's properties.
12. Specify a name and description for the Compute window:
  - In the right pane, in the **Name** field, change the default name to **compute\_win**.
  - In the **Description** field, enter **This window uses expressions to calculate each field. The first field uses the expression to calculate the count. The last two fields are just passing through what is in the input window.**
13. Connect the src\_win window to the compute\_win window with an edge:
  - Position the cursor over the anchor point at the bottom of the src\_win window so that the color of the anchor point changes to white.
  - Click the white anchor point, hold the left mouse button down, and draw a line to the anchor point in the compute\_win window.  
The compute\_win window now accepts values from the src\_win window.

14. Specify an output schema for the compute\_win window:

- On the right toolbar, click .
- Click .
- Click .

The Output Schema window appears.

- Click  to add a row to the schema table. After you add a row, click  again to add the next row.

Enter the following values:

Key	Field Name	Type	Expression
Y	ID	Int32	not applicable
N	counter	Int32	counter=counter+1 return counter
N	symbol	String	symbol
N	price	Double	price

- Click **OK**.

15. Configure the compute\_win window's split method:

- Click .
- Expand **Advanced**.
- Select the **Split the output** check box.
- Confirm that **Expression** is selected in the **Split method** field.
- Enter **ID%2** in the **Expression** field.
- Collapse **Advanced**.

16. Configure the compute\_win window's compute settings:

- Expand **Compute Settings**.
- Confirm that **Expressions** is selected in the **Compute method** field.
- Select the **Include engine initialization expression** check box.
- Select **Int32** in the **Return type** drop-down list.
- In the **Expression** field, enter:
 

```
integer counter counter=0
```
- Collapse **Compute Settings**.

17. Configure a subscriber connector for the compute\_win window:

- Expand **Subscriber Connectors** and click .
- The Connector Configuration window appears.
- In the **Name** field, replace the default value with `sub`.
- In the **Fsname** field, enter the path to the CSV file. For example, you might enter `/opt/sas/viya/`

## 2.16 Example: Splitting Generated Events across Output Slots

home/SASEventStreamProcessingEngine/<release>/examples/xml/slot\_exp\_xml/compute.csv. Replace <release> with the release number in your installation directory path.

- In the **Fstype** drop-down list, select **csv**.
- Click **OK**.

18. Expand **Transformations** on the **Windows** pane on the left and drag a Copy window to the workspace.

The right pane displays the Copy window's properties.

19. Specify a name and description for the Copy window:

- In the right pane, in the **Name** field, change the default name to **cw\_01**.
- In the **Description** field, enter **This Copy window connects to the Compute window using the output slot 0**.

20. Configure a subscriber connector for the cw\_01 window:

- Expand **Subscriber Connectors** and click  .  
The Connector Configuration window appears.
- In the **Name** field, replace the default value with `sub1`.
- In the **Fsname** field, enter the path to the CSV file. For example, you might enter  
`/opt/sas/viya/home/SASEventStreamProcessingEngine/<release>/examples/xml/slot_exp_xml/cw_01.csv`. Replace <release> with the release number in your installation directory path.
- Select the **Snapshot** check box.
- In the **Fstype** drop-down list, select **csv**.
- Click **OK**.

21. Configure the cw\_01 window's retention properties:

- Click the cw\_01 window in the workspace.
- Expand **Retention**.
- Confirm that the **Type** field is set to **By time, sliding**.
- In the **Time limit** field, enter **1000** and select **Seconds** from the drop-down list.

22. Connect the compute\_win window to the cw\_01 window with an edge:

- Position the cursor over the anchor point at the bottom of the compute\_win window so that the color of the anchor point changes to white.
- Click the white anchor point, hold the left mouse button down, and draw a line to the anchor point in the compute\_win window.

The cw\_01 window now accepts values from the compute\_win window.

23. Assign a slot number to the edge:

- Click the edge that connects the compute\_win window with the cw\_01 window.
- In the right pane, enter **0** in the **Slot** field.

24. Expand **Transformations** on the **Windows** pane on the left and drag a Copy window to the workspace.

The right pane displays the Copy window's properties.

25. Specify a name and description for the Copy window:

- In the right pane, in the **Name** field, change the default name to `cw_02`.
- In the **Description** field, enter **This Copy window connects to the Compute window using the output slot 1.**

26. Configure a subscriber connector for the `cw_02` window:

- Expand **Subscriber Connectors** and click  .  
The Connector Configuration window appears.
- In the **Name** field, replace the default value with `sub2`.
- In the **Fsname** field, enter the path to the CSV file. For example, you might enter `/opt/sas/viya/home/SASEventStreamProcessingEngine/<release>/examples/xml/slot_exp_xml/cw_02.csv`. Replace `<release>` with the release number in your installation directory path.
- Select the **Snapshot** check box.
- In the **Fstype** drop-down list, select **csv**.
- Click **OK**.

27. Configure the `cw_02` window's retention properties:

- Click the `cw_02` window in the workspace.
- Expand **Retention**.
- Confirm that the **Type** field is set to **By time, sliding**.
- In the **Time limit** field, enter **1000** and select **Seconds** from the drop-down list.

28. Connect the `compute_win` window to the `cw_02` window with an edge:

- Position the cursor over the anchor point at the bottom of the `compute_win` window so that the color of the anchor point changes to white.
- Click the white anchor point, hold the left mouse button down, and draw a line to the anchor point in the `compute_win` window.

The `cw_02` window now accepts values from the `compute_win` window.

29. Assign a slot number to the edge you just created:

- Click the edge that connects the `compute_win` window with the `cw_02` window.
- In the right pane, enter **1** in the **Slot** field.

30. Expand **Transformations** on the **Windows** pane on the left and drag a Copy window to the workspace.

The right pane displays the Copy window's properties.

2.16 Example: Splitting Generated Events across Output Slots

31. Specify a name and description for the Copy window:
  - In the right pane, in the **Name** field, change the default name to `cw_03`.
  - In the **Description** field, enter **This Copy window connects to the Compute window using the output slot –1**.
32. Configure a subscriber connector for the `cw_03` window:
  - Expand **Subscriber Connectors** and click .
  - The Connector Configuration window appears.
  - In the **Name** field, replace the default value with `sub3`.
  - In the **Fsname** field, enter the path to the CSV file that will contain the output. For example, you might enter `/opt/esa/<release>/examples/xml/ slot_exp_xml/cw_03.csv`. Replace `<release>` with the release number in your installation directory path.
  - Select the **Snapshot** check box.
  - In the **Fstype** drop-down list, select `csv`.
  - Click **OK**.
33. Configure the `cw_03` window's retention properties:
  - Click the `cw_03` window in the workspace.
  - Expand **Retention**.
  - Confirm that the Type field is set to By time, sliding.
  - In the **Time limit** field, enter **1000** and select **Seconds** from the drop-down list.
34. Connect the `compute_win` window to the `cw_03` window with an edge:
  - Position the cursor over the anchor point at the bottom of the `compute_win` window so that the color of the anchor point changes to white.
  - Click the white anchor point, hold the left mouse button down, and draw a line to the anchor point in the `compute_win` window.
  - The `cw_03` window now accepts values from the `compute_win` window.
35. Assign a slot number to the edge you just created:
  - Click the edge that connects the `compute_win` window with the `cw_03` window.
  - In the right pane, enter `-1` in the **Slot** field.
36. The model is now complete. Click  to save your model.
37. Click  **Enter Test Mode** .
  - A new page called `Test: modellingSplitter_demo` appears.
38. In the Test Server drop-down list, select the Streaming Server on which you want to test the model.

39. Click  Run Test .

The results for each window appear on separate tabs:

- The src\_win tab lists the securities transactions
- The compute\_win tab lists the computed fields
- The cw\_01 tab lists the trades output to slot 0
- The cw\_02 tab lists the trades output to slot 1
- The cw\_03 tab lists the trades output to slot –1

---

**Note**

If the table is empty, check that the publisher connector for the src\_win window correctly points to the relevant CSV file.

---

40. To stop the test, click  Stop .

The project stops and then unloads from the Streaming Server.

## 2.17 Example: Identifying Trading Patters in a Stock Market

### 2.17.1 Overview

This model identifies increases in a stock's price within a specific time interval. The model contains a Source window and a Pattern window. The Source window receives an event stream of stock trades from an input file that is included with this example. The Pattern window defines the events of interest to be matched. The model is stateless, that is, the index on the Source window has the type pi\_EMPTY. Events are not retained in any window, and are transformed and passed through. This prevents the Pattern window from growing infinitely. The pattern defined in the Pattern window consists of the following events of interest:

- Event 1: Occurrences of the stock symbol GMTTC
- Event 2: Re-occurrences of the stock symbol GMTTC where the price and quantity of the stock has gone up 50% compared to event 1
- Event 3: Re-occurrences of the stock symbol GMTTC where the price and quantity of the stock has gone up 50% compared to event 2

---

**Note**

For the pattern to be matched, all events of interest must occur within 200 milliseconds of each other. The time that each event occurred is specified in the trade\_time field in the Source window's output schema.

---

2.17 Example: Identifying Trading Patterns in a Stock Market

This example uses three files listed below:

1. The XML file (model.xml) associated with this example.
2. 50k.csv is an input file. This file contains a list of stock trades.
3. output.csv is an output file. When you run the model, the matched patterns are written to the output.csv file.

2.17.2 Project Details

This project contains two windows:

- The sourceWindow\_01 window is a Source window. This is where a list of stock trades from the 50k.csv file enter the model.
- The patternWindow\_01 window is a Pattern window. This is where the stock trade patterns are matched and written to an output.csv file.

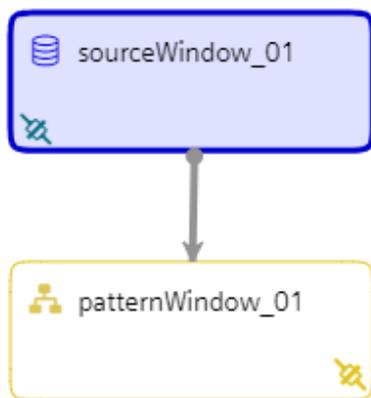


Figure 2-32 Diagram of the Identifying Trading Patterns in a Stock Market Project

### 2.17.3 Example Steps

To complete this example, follow these steps:

1. On the **Projects** page, click .  
The New Project window appears.
2. In the New Project window, do the following:
  - In the **Name** field, enter `pattern_empty_index`.
  - Click **OK**.

If you do not currently have any Streaming Servers configured, you are prompted to decide whether you want to configure a streaming Server now.

---

#### Note

It is assumed that you do not have any Streaming Servers configured. If you already have Streaming Servers configured, go to step 5.

---

3. Click Yes to configure a streaming Server now. The Streaming Server Properties window appears.
4. Configure a streaming Server:
  - In the **Name** field, enter a name to identify the new Streaming Server that you want to create.
  - In the **Host** field, enter the host name or IP address of the new Streaming Server.
  - In the **HTTP port** field, enter the new Streaming Server's HTTP publish/subscribe port.
  - If required, in the **Description** field, enter a description of the new Streaming Server.
  - If required, in the **Tags** field, enter any keywords that describe the Streaming Server and then press Enter.
  - If required, click **Edit** to change the setting for the **Authentication** field:

**None:** This is the default option.

**Kerberos:** This option is relevant only if the Streaming Server is configured to require authentication using Kerberos.

**OAuth token:** This option is relevant only if the Streaming Server is configured to require authentication using an OAuth token. If you select this option, an additional field appears where you must enter the OAuth token.

**Username and password:** This option is relevant only if the Streaming Server is configured to require authentication using a username and password (SASLogon

2.17 Example: Identifying Trading Patterns in a Stock Market

Services). If you select this option, additional fields appear where you must enter the username and password.

- If required, select the **Connect using SSL** check box. Selecting this option is relevant only if the Streaming Server is configured to require SSL encryption.
- If required, select the **Enable server logging** check box to enable logging on the Streaming Server.
- If required, in the **Number of messages to retain** field, change the default number of messages that are retained by the Streaming Server log. The default is 10,000 messages.
- Click **OK**.

Your project is created with a set of default properties.

5. Expand **Input Streams** on the **Windows** pane on the left and drag a Source window to the workspace.

The right pane displays the Source window's properties.

6. Specify a name and description for the Source window:

- In the right pane, in the **Name** field, change the default name to `sourceWindow_01`.
- In the **Description** field, enter `This Source window receives stock trades. This window contains a file/socket connector which reads in the stock trades from a file in CSV format and then publishes the trades to the ESP Engine.`

7. Configure the `sourceWindow_01` window's state and event type:

- In the right pane, expand **State and Event Type**.
- In the **Window state and index** drop-down list, select **stateless (pi\_EMPTY)**.
- Select the **Accept only "Insert" events** check box.

---

**Note**

If a Source window precedes a Pattern window, you must specify that the Source window is insert- only. This causes the Source window to reject any events with an opcode other than Insert, and permits an index type of `pi_EMPTY` to be used.

---

## 8. Specify an output schema for the sourceWindow\_01 window:

- On the right toolbar, click .
- Click .

The Output Schema window appears.

- Click  to add a row to the schema table. After you add a row, click  again to the add the next row.

Enter the following values:

Key	Field Name	Type
Y	ID	Int32
N	symbol	String
N	currency	Int32
N	update	Int64
N	msecs	Int32
N	price	Double
N	quant	Int32
N	venue	Int32
N	broker	Int32
N	buyer	Int32
N	seller	Int32
N	buysellflg	Int32
N	trade_time	Timestamp

- Click **OK**.

## 9. The sourceWindow\_01 window will stream a list of vehicles from a file called 50k.csv that contains example data. To add a connector to this CSV file:

- Click .
- Expand **Input Data (Publisher) Connectors** and click .

The Connector Configuration window is displayed.

- In the **Name** field, replace the default value with `pub`.
- In the **Fsname** field, enter the path to the CSV file. For example, you might enter `/opt/sas/viya/home/SASEventStreamProcessingEngine/<release>/examples/xml/pattern_empty_index/50k.csv`. Replace `<release>` with the release number in your installation directory path.
- In the **Fstype** drop-down list, select `csv`.
- Configure the `pub` connector's properties:
  - i. Click **All properties**.  
The All Properties window appears.
  - ii. Enter `%Y-%m-%d %H:%M:%S` in the **Value** field of the **dateformat** property.
  - iii. Click **OK**.
- Click **OK**.

2.17 Example: Identifying Trading Patterns in a Stock Market

10. Expand **Utilities** on the **Windows** pane on the left and drag a Pattern window to the workspace.

The right pane displays the Pattern window's properties.

Pattern windows are insert-only with respect to both their input windows and the output that they produce. As the input and output of a Pattern window are unbounded and insert-only, they are typically stateless windows (that is, windows with index type `pi_EMPTY`).

11. Specify a name and description for the Pattern window:

In the right pane, in the **Name** field, change the default name to `patternWindow_01`.

In the **Description** field, enter `The Pattern window generates pattern matches. The pattern element defines the pattern of interest.`

12. Click **OK**.

13. Connect the `sourceWindow_01` window to the `patternWindow_01` window with an edge:

- Position the cursor over the anchor point at the bottom of the `sourceWindow_01` window so that the anchor point color changes to white.
- Click the white anchor point, hold the mouse button down, and draw a line to the anchor point in the `patternWindow_01` window.

The `patternWindow_01` window now accepts values from the `sourceWindow_01` window.

14. Configure the patternWindow\_01 window's pattern of interest:

- Select the patternWindow\_01 window on the workspace.
- In the right pane, expand **Patterns**.
- Click .

A window is displayed enabling you to define the pattern's properties.

In the **Attributes** section, in the **Name** field, replace the default value with `pattern1`.

Select the **Specify a timefield** check box.

Confirm that **sourceWindow\_01** is selected from the **Source** drop-down list.

Confirm that **trade\_time** is selected by default from the **Field** drop-down list.

In the **Events** section, click .

A panel is displayed that enables you to define an event.

In the **Name** field, enter `e1`.

In the `e1` event's text box, enter `symbol=="GMTC" and s==symbol and p0==price and q0==quant`

#### Note

Alternatively, you can define event operators by double-clicking the relevant operator in the **Operators** section on the left toolbar.

In the **Events** section, click .

A panel is displayed that enables you to define an event.

In the **Name** field, enter `e2`.

In the `e2` event's text box, enter `s==symbol and p0<price*1.5 and q0<quant*1.5 and p1==price and q1==quant`

In the **Events** section, click .

A panel is displayed that enables you to define an event.

In the **Name** field, enter `e3`.

In the `e3` event's text box, enter `s==symbol and p1<price*1.5 and q1<quant*1.5`

In the **Logic** section, click the panel.

Enter `fby{200 milliseconds}(e1, e2, e3)` in the text box.

#### Note

Alternatively, you can define event logic by double-clicking the relevant event name, operator, or template name on the left toolbar.

On the breadcrumb trail on the toolbar, click **PatternWindow\_01**.

The workspace is displayed.

- Specify an output schema for the patternWindow\_01 window:  
On the right toolbar, click .

2.17 Example: Identifying Trading Patters in a Stock Market

Click  .

The Output Schema and Pattern Mappings window appears.

In the **Key Field** section, enter ID in the **Name** field.

In the **Mapped Fields** section, click  to add a row to the schema table. After you add a row, click  again to add the next row.

Enter the following values:

Field	Type	Output Type	Event	Value
ID1	Int32	Field-Selection	e1	ID
ID2	Int32	Field-Selection	e2	ID
ID3	Int32	Field-Selection	e3	ID

Click **OK**.

- Create a subscribe connector:

Click  .

In the right pane, expand **Subscriber Connectors** and click  .

The Connector Configuration window is displayed.

In the **Name** field, replace the default value with sub.

Select the **Snapshot** check box.

In the **Fsname** field, enter the path to the CSV file. For example, you might enter `/opt/sas/viya/ home/SASEventStreamProcessingEngine/<release>/examples/xml/pattern_empty_index/output.csv`. Replace `<release>` with the release number in your installation directory path.

In the **Fstype** drop-down list, select **csv**.

Click **OK**.

15. Configure the project's continuous query:

- Click  .
- Expand **Debugging**.
- In the **Trace in server log** field, select **patternWindow\_01**.

16. The model is now complete. Click  to save your model.

17. Click  **Enter Test Mode** .

A new page called **Test: pattern\_empty\_index\_demo** appears.

18. In the **Streaming Server** drop-down list, select the Streaming Server on which you want to the model.

19. Click  **Run Test**.

The results for each window appear on separate tabs:

- the **sourceWindow\_01** tab lists the stock trades that are received from the input file.
- the **patternWindow\_01** tab lists the matched patterns.

---

**Note**

If the table is empty, check that the publisher connector for the sourceWindow\_01 window is set correctly to point to the CSV file.

---

20. To stop the test, click  **Stop**.

The project stops and then unloads from the Streaming Server.

## 2.18 Example: Transitioning a Model from Stateful to Stateless

### 2.18.1 Transition a Model from Stateful to Stateless Overview

This example demonstrates how to facilitate the transition of a stateful part of a model to a stateless part of a model. A Remove State window converts all events that it receives into Inserts and adds a field named eventNumber, which is a monotone-increasing sequential integer. This added field is the only key of the Remove State window.

This example uses two files listed below:

- The XML file (removeState.xml) associated with this example.
- InputRemove.csv is an input file containing the event stream.

### 2.18.2 Project Details

This project contains three windows:

- SourceWindow is a Source window. This is where the events from the InputRemove.csv file enter the model.
- removeStateWindow is a Remove State Window. This is where the events are filtered.
- Copy window is a Copy window. This show the transition to a stateless model.

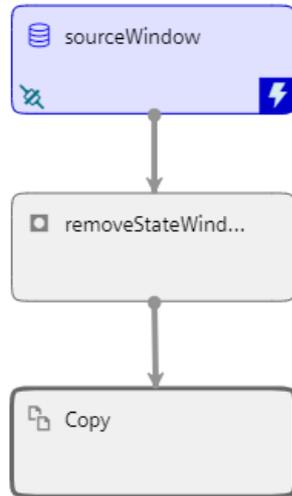


Figure 2-33 Diagram of the Transition a Model from Stateful to a Stateless Model.

### 2.18.3 Example Steps

To complete this example, follow these steps:

1. On the **Projects** page, click .  
The New Project window appears.
2. In the New Project window, do the following:
  - In the **Name** field, enter `RemoveState`
  - Click **OK**.

If you do not currently have any Streaming Servers configured, you are prompted to decide whether you want to configure a streaming Server now.

---

#### Note

It is assumed that you do not have any Streaming Servers configured. If you already have Streaming Servers configured, go to step 5.

---

3. Click **Yes** to configure a streaming Server now.  
The Streaming Server Properties window appears.

4. Configure a streaming Server:
  - In the **Name** field, enter a name to identify the new Streaming Server that you want to create.
  - In the **Host** field, enter the host name or IP address of the new Streaming Server.
  - In the **HTTP port** field, enter the new Streaming Server's HTTP publish/subscribe port.
  - If required, in the **Description** field, enter a description of the new Streaming Server.
  - If required, in the **Tags** field, enter any keywords that describe the Streaming Server and then press Enter.
  - If required, click **Edit** to change the setting for the Authentication field:  
**None:** This is the default option.
  - If required, select the **Connect using SSL** check box. Selecting this option is relevant only if the Streaming Server is configured to require SSL encryption.
  - If required, select the **Enable server logging** check box to enable logging on the Streaming Server.
  - If required, in the **Number of messages to retain** field, change the default number of messages that are retained by the Streaming Server log. The default is 10,000 messages.
  - Click OK.  
Your project is created with a set of default properties.
5. In the right pane, configure your project's properties: In the **Name** field, change the default name to `Removewindow_proj`.
6. Expand **Input Streams** on the **Windows** pane on the left and drag a Source window to the workspace. The right pane displays the Source window's properties.  
This window receives events about
7. Specify a name for the SourceWindow window: In the right pane, in the **Name** field, change the default name to `SourceWindow`.

2.18 Example: Transitioning a Model from Stateful to Stateless

8. Specify an output schema for the SourceWindow window:

- On the right toolbar, click .
- Click .

The Output Schema window appears.

- Click  to add a row to the schema table. After you add a row, click  again to the add the next row.

Enter the following values:

Key	Field Name	Type
Y	ID	Int32
N	symbol	String
N	true_test	Int32
N	price	Double

- Click **OK**.

9. Configure the SourceWindow window to stream events from a file called InputRemove.csv that contains data. You can find this example CSV file in the removeState folder in the examples directory. To add a connector to this CSV file:

- In the right pane, click .
- Expand **Input Data (Publisher) Connectors**.
- Click .

The Connector Configuration window appears.

- In the **Name** field, replace the default value with `pub1`.
- In the **Fsname** field, enter the path to the CSV file. For example, you might enter `/opt/sas/viya/home/SASEventStreamProcessingEngine/<release>/examples/xml/removeState/ InputRemove.csv`. Replace `<release>` with the release number in your installation directory path.
- In the **Fstype** drop-down list, select `csv`.
- Configure the `pub1` connector's properties:

Click **All properties**.

The All Properties window appears.

Enter `false` in the **Value** field of the **transactional** property.

Enter `1` in the **Value** field of the **blocksize** property.

Click **OK**.

- Click **OK**.
- Collapse **Input Data (Publisher) Connectors**.

10. Expand **Transformations** on the **Windows** pane on the left and drag a Remove State window to the workspace.

The right pane displays the Remove State window's properties.

11. Specify a name for the Remove State window. In the right pane, in the **Name** field, change the default name to `removeStateWindow`.
12. Connect the sourceWindow window to the removeStateWindow window with an edge:
  - Position the cursor over the anchor point at the bottom of the sourceWindow window so that the anchor point color changes to white.
  - Click the white anchor point, hold the mouse button down, and draw a line to the anchor point in the removeStateWindow window.

The removeStateWindow window now accepts values from the sourceWindow window.
13. Click the removeStateWindow on the workspace.
14. Configure the removeStateWindow's settings:
  - In the right pane, expand **Settings** if it is not already expanded.
  - Clear the **Update block deletes** check box.
  - Confirm that the **Deletes**, **Retention updates**, and **Retention deletes** check boxes are selected. This setting filters events to pass through the window by their type.
  - In addition to these selections, select the **Updates** check box.
  - In the **Log Fields** field, select the **Include opcode and flag fields** check box.
15. View the removeStateWindow's output schema:
  - On the right toolbar, click .
  - Examine the output schema and notice that the event number, original flag, and original opcode have all been added to the schema. These are strings that specify the opcode and the flags of the event. These two fields directly follow the eventNumber field.
  - Click .
16. Expand **Transformations** on the **Windows** pane on the left and drag a Copy window to the workspace. The right pane displays the Copy window's properties.
17. Specify a name for the Copy window: In the right pane, in the **Name** field, change the default name to Copy.
18. Connect the removeStateWindow window to the Copy window with an edge:
  - Position the cursor over the anchor point at the bottom of the removeStateWindow window so that the anchor point color changes to white.
  - Click the white anchor point, hold the mouse button down, and draw a line to the anchor point in the removeStateWindow window.

The Copy window now accepts values from the removeStateWindow window.
19. Click the Copy on the workspace.
20. Configure the Copy window's state and index:
  - In the right pane, expand **State**.
  - In the **Window state and index** drop-down list, select **Stateful (pi\_RBTREE)**.

## 2.19 Example: Transposing Data from an Aircraft

21. Configure the Copy window's retention policy:

- In the right pane, expand **Retention**.
- Confirm that the **Type** field is set to **By time, sliding**.
- In the **Time limit** field, enter 30 and select **Seconds** from the drop-down list.

22. Configure the project's continuous query:

- Click .
- In the right pane, expand **Attributes**.
- Select the **Instantiate Source windows at run time** check box.
- Expand **Debugging**.
- In the **Enable trace server logging for this query** field, select **removeStateWindow** from the drop-down list.

23. The model is now complete. Click  to save your model.

24. Click  **Enter Test Mode**.

A new page called **Test: RemoveState** appears.

25. In the **Streaming Server** drop-down list, select the Streaming Server on which you want to test the model.

26. Click  **Run Test**.

The results for each window appear on separate tabs:

- The **sourceWindow** tab lists all events from the event stream.
- The **removeStateWindow** tab lists the events from the event stream excluding the Delete events that you specified must be removed.
- The **Copy** tab lists retained events.

27. To stop the test, click  **Stop**.

The project stops and then unloads from the Streaming Server.

## 2.19 Example: Transposing Data from an Aircraft

### 2.19.1 Transpose Aircraft Information Example Overview

This example conceptualizes an event as a row that consists of multiple columns. You can use a Transpose window to interchange an event's rows as columns, and columns as rows. Use attributes of the Transpose window to govern the rearrangement of data. You will process information about the pitch, yaw, roll, and velocity of an aircraft in flight.

As the Transpose window has two modes, long and wide, this example consists of two parts:

- Transpose aircraft information in wide mode. For more information, see "**Wide Mode Example Steps**".
- Transpose aircraft information in long mode. For more information, see "**Long Mode Example Steps**".

The long mode part of this example uses the following files:

- The XML file (transpose\_long.xml) associated with this example.
- input\_long.csv. This is an input file. This file contains event streams from the aircraft in flight.

The wide mode part of this example uses the following files:

- The XML file (transpose\_wide.xml) associated with this example.
- input\_wide.csv. This is an input file. This file contains event streams from the aircraft in flight.

## 2.19.2 Project Details

This project contains two windows:

- A Source window, where aircraft events from the input file enter the model. In the wide mode version of this example, this file is called input\_wide.csv. In the long mode version of this example, this file is called input\_long.csv.
- A Transpose window, where the transposition of the aircraft events occurs. You can configure the attributes of the Transpose window to govern the rearrangement of data. In the wide mode version of this example, this window is called TransposeW. In the long mode version of this example, this window is called TranposeL.

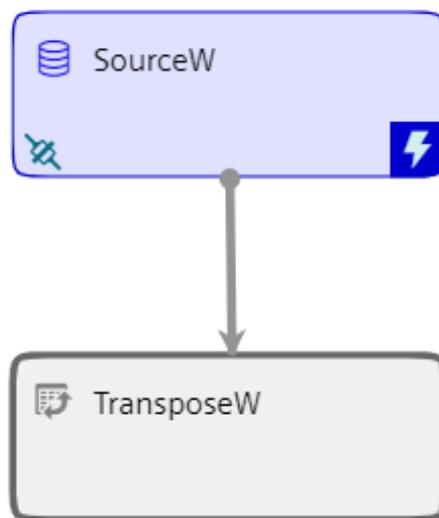


Figure 2-34 Diagram of the Transpose Model in Wide Mode

### 2.19.3 Wide Mode Example Steps

To complete this example, follow these steps:

1. On the Projects page, click .

The New Project window appears.

2. In the New Project window, do the following:

- In the **Name** field, enter `Transpose_wide`.
- In the **Description** field, enter a description: `This example transposes information in wide mode about the pitch, yaw, roll, and velocity of an aircraft in flight.`
- Click **OK**.

If you do not currently have any Streaming Servers configured, you are prompted to decide whether you want to configure a streaming Server now.

---

#### Note

It is assumed that you do not have any Streaming Servers configured. If you already have Streaming Servers configured, go to step 5.

---

3. Click **Yes** to configure a streaming Server now

The Streaming Server Properties window appears.

4. Configure a streaming Server:

- In the **Name** field, enter a name to identify the new Streaming Server that you want to create.
- In the **Host** field, enter the host name or IP address of the new Streaming Server.
- In the **HTTP port** field, enter the new Streaming Server's HTTP publish/subscribe port.
- If required, in the **Description** field, enter a description of the new Streaming Server.
- If required, in the **Tags** field, enter any keywords that describe the Streaming Server and then press **Enter**.
- If required, click **Edit** to change the setting for the **Authentication** field:
- **None**: This is the default option.
- If required, select the **Connect using SSL** check box. Selecting this option is relevant only if the Streaming Server is configured to require SSL encryption.
- If required, select the **Enable server logging** check box to enable logging on the Streaming Server.
- If required, in the **Number of messages to retain** field, change the default number of messages that are retained by the Streaming Server log. The default is 10,000 messages.
- Click **OK**.

Your project is created with a set of default properties.

5. Configure the project's threading level:
  - In the right pane, expand **Attributes**.
  - In the **Threads** field, enter **2**.
6. Expand Input Streams on the Windows pane on the left and drag a Source window to the workspace.
 

The right pane displays the Source window's properties.

This window receives information about an aircraft.
7. Specify a name for the Source window: in the right pane, in the **Name** field, change the default name to **SourceW**.
8. Change the SourceW window's state and event type:
  - Expand **State and Event Type**.
  - In the **Window state and index** drop-down list, select **Stateless (pi\_EMPTY)**.
  - Select the **Accept only "Insert" events** check box.
9. Specify an output schema for the SourceW window:

- In the right pane, click .

- Click .

The Output Schema window appears.

- Click  to add a row to the schema table. After you add a row, click  again to add the next row.

Enter the following values in the rows:

Key	Field Name	Type
Y	ID	Int64
N	PlaneID	String
N	TAG	String
N	value	Double
N	time	Stamp
N	lat	Double
N	long	Double

- The PlaneID field of the schema identifies which aircraft provides the data.
- The TAG field specifies whether the data contains the aircraft's pitch, yaw, roll, or velocity.
- The value field records the numerical value for the TAG and the specific time that the data is recorded.
- The event captured the plane's latitude (lat) and longitude (long) when the pitch, yaw, roll, or velocity is recorded.
- Click **OK**.

2.19 Example: Transposing Data from an Aircraft

10. Configure the SourceWindow window to stream events from a file called input.csv that contains data from an aircraft in flight. You can find this example CSV file in the transpose\_wide folder in the examples directory. To add a connector to this CSV file:

- In the right pane, click .
- Expand **Input Data (Publisher) Connectors**.
- Click .

The Connector Configuration window appears.

- In the **Name** field, replace the default value with pub.
- In the **Fsname** field, enter the path to the CSV file. For example, you might enter `/opt/sas/viya/home/SASEventStreamProcessingEngine/<release>/examples/xml/transpose_wide/ input_wide.csv`. Replace <release> with the release number in your installation directory path.
- In the **Fstype** drop-down list, select **csv**.
- Configure the pub connector's properties:
- Click **All properties**.

The All Properties window appears.

- Select false in the **Value** field of the **transactional** property.
- Enter 1 in the **Value** field of the **blocksize** property.
- Enter `%Y-%m-%d %H:%M:%S` in the Value field of the **dateformat** property.
- Enter 1 in the **Value** field of the **Rate** property.
- Click OK.
- Click OK.

11. Collapse **Input Data (Publisher) Connectors**.

## 12. Open the input\_wide.csv file and examine its contents:

```

i,n,1,turboprop #1, pitch,1.1, 2017-08-30 15:21:00.000000,10,10
i,n,2,turboprop #2, velocity,-1.2, 2017-08-30 15:21:00.000001,20,20
i,n,3,turboprop #1, roll,-1.1, 2017-08-30 15:21:00.000002,11,11
i,n,4,turboprop #2, yaw,2.2, 2017-08-30 15:21:00.000003,21,21
i,n,5,turboprop #2, pitch,1.2, 2017-08-30 15:21:00.000004,22,22
i,n,6,turboprop #1, velocity,-1.1, 2017-08-30 15:21:00.000005,12,12
i,n,7,turboprop #2, roll,-1.2, 2017-08-30 15:21:00.000006,23,23
i,n,8,turboprop #1, yaw,2.1, 2017-08-30 15:21:00.000007,13,13
i,n,9,jet #1, pitch,1.3, 2017-08-30 15:21:00.000012,30,30
i,n,10,jet #2, velocity,-4.4, 2017-08-30 15:21:00.000013,40,40
i,n,11,jet #1, roll,-4.3, 2017-08-30 15:21:00.000014,31,31
i,n,12,jet #2, yaw,8.4, 2017-08-30 15:21:00.000015,41,41
i,n,13,jet #2, pitch,4.4, 2017-08-30 15:21:00.000016,42,42
i,n,14,jet #1, velocity,-4.3, 2017-08-30 15:21:00.000017,32,32
i,n,15,jet #2, roll,-4.4, 2017-08-30 15:21:00.000018,43,43
i,n,16,jet #1, yaw,8.3, 2017-08-30 15:21:00.000019,33,33
i,n,17,turboprop #1, pitch,23.1, 2017-08-30 15:21:06.000022,14,14

```

In these seventeen events, four planes stream data through the Source window:

turboprop #1, turboprop #2, jet #1, and jet #2. Each event contains either a pitch, velocity, roll, or yaw value at a specific time.

- Wide mode produces output events that contain multiple columns, each based on data streamed through the input events.
- The values of TAG in the input events determine columns in output events.
- Tags-included specify which specific values of TAG are to be included in the output events. Here, all four values of TAG are specified.
- The value and time associated with pitch, yaw, roll, and velocity are included in the output event. The associated latitude and longitude are passed through.
- Output events are grouped by the value of PlaneID.

Output columns are formed by taking the cross product of the following:

```
{pitch, yaw, roll, velocity} times {value, time}
```

This yields pitch\_value, pitch\_time, yaw\_value, yaw\_time, and so on.

13. Expand **Transformations** on the **Windows** pane on the left and drag a Transpose window to the workspace.

The right pane displays the Transpose window's properties.

The input data is to be streamed through a Transpose window to create a single event (row). This event contains the pitch, velocity, roll, and yaw of each aircraft at a specific time.

2.19 Example: Transposing Data from an Aircraft

14. Specify a name and description for the Transpose window. In the right pane, in the **Name** field, change the default name to **TransposeW**.
15. Connect the SourceW window to the TransposeW window with an edge:
  - a Position the cursor over the anchor point at the bottom of the SourceW window so that the anchor point color changes to white.
  - Click the white anchor point, hold the mouse button down, and draw a line to the anchor point in the TransposeW window.

The TransposeW window now accepts values from the SourceW window.
16. Click the TransposeW window on the workspace.
17. Configure the TransposeW window's settings:
  - If necessary, click **Settings** to expand the section.
  - Confirm the **Mode** field is set to Wide.
  - In the **Tag name** field, select **TAG** from the drop-down list.
  - In the **Included tags** field, enter the following tag names: `pitch`, `roll`, `yaw`, and `velocity`. After you have entered each tag name, press **Enter** to confirm its creation.

---

**Note**

Ensure you enter each tag in all lowercase characters.

---

- In the Tag values field, double-click the row in the table.

The Select Tag Value Fields window appears.

Select the value and time fields.

Click **OK** to confirm your selection.
- In the **Group By** field, select **PlaneID** from the drop-down list.

18. Configure a subscriber connector for the TransposeW window:

- Expand **Subscriber Connectors**.
- Click .
- The Connector Configuration window appears.
- In the **Name** field, enter `sub`.
- In the **Fsname** field, enter the path to the output file: **result.out**. For example, you might enter `/opt/esa/<release>/examples/xml/ transpose_wide/result.out`. Replace `<release>` with the release number in your installation directory path.
- In the **Fstype** drop-down list, select `csv`.
- Configure the sub connector's properties:
- Click **All properties**.
- The All Properties window appears.
- Enter `%Y-%m-%d %H:%M:%S` in the Value field of the **dateformat** property.
- Click **OK**.
- Click **OK**.

19. Configure the project's continuous query:

- Click .
- In the right pane, in the **Name** field, change the continuous query's default name `cq1` to `transpose_cq`.
- Expand **Debugging**.
- In the **Trace in server log** field, select `TransposeW` from the drop-down list.

20. The model is now complete. Click  to save your model.

21. Click  **Enter Test Mode**.

A new page called Test: **transpose\_wide** appears.

22. In the **Streaming Server** drop-down list, select the Streaming Server on which you want to test the model.

23. Click  **Run Test**.

The results for each window appear on separate tabs:

- The **SourceW** tab shows events representing the data received from the aircraft.
- The **TransposeW** tab shows the transposed aircraft data in wide format.

24. To stop the test, click  **Stop**.

The project stops and is unloaded from the Streaming Server.

### 2.19.4 Long Mode Example Steps

To complete this example, follow these steps:

1. On the Projects page, click .

The New Project window appears.

2. In the New Project window, do the following:

- In the **Name** field, enter `Transpose_long`.
- In the **Description** field, enter a description: `This example transposes information in long mode about the pitch, yaw, roll, and velocity of an aircraft in flight.`
- Click **OK**.

If you do not currently have any Streaming Servers configured, you are prompted to decide whether you want to configure a streaming Server now.

---

#### Note

It is assumed that you do not have any Streaming Servers configured. If you already have Streaming Servers configured, go to step 5.

---

3. Click **Yes** to configure a streaming Server now.

The Streaming Server Properties window appears.

4. Configure a streaming Server:

- In the **Name** field, enter a name to identify the new Streaming Server that you want to create.
- In the **Host** field, enter the host name or IP address of the new Streaming Server.
- In the **HTTP port** field, enter the new Streaming Server's HTTP publish/subscribe port.
- If required, in the **Description** field, enter a description of the new Streaming Server.
- If required, in the **Tags** field, enter any keywords that describe the Streaming Server and then press Enter.
- If required, click **Edit** to change the setting for the **Authentication** field:
- **None**: This is the default option.
- **Kerberos**: This option is relevant only if the Streaming Server is configured to require authentication Kerberos.
- **OAuth token**: This option is relevant only if the Streaming Server is configured to require authentication using an OAuth token. If you select this option, an additional field appears where you must enter the OAuth token.
- **Username and password**: This option is relevant only if the Streaming Server is configured to require authentication using a username and password (SASLogon

Services). If you select this option, additional fields appear where you must enter the username and password.

- If required, select the **Connect using SSL** check box. Selecting this option is relevant only if the Streaming Server is configured to require SSL encryption.
- If required, select the **Enable server logging** check box to enable logging on the Streaming Server.
- If required, in the **Number of messages to retain** field, change the default number of messages that are retained by the Streaming Server log. The default is 10,000 messages.
- Click **OK**.

Your project is created with a set of default properties.

5. Configure your project's threading level: a
  - In the right pane, expand **Attributes**.
  - In the **Threads** field enter 2.
6. Expand Input Streams on the Windows pane on the left and drag a Source window to the workspace.

The right pane displays the Source window's properties.

This window receives events about

7. Specify a name for the Source window: in the right pane, in the **Name** field, change the default name to SourceW
8. Change the SourceW window's state and event type:
  - Expand **State and Event Type**.
  - In the Window state and index drop-down list, select **Stateless (pi\_EMPTY)**.
  - Select the **Accept only "Insert" events** check box.

2.19 Example: Transposing Data from an Aircraft

9. Specify an output schema for the SourceW window:

- In the right pane, click .
- Click .

The Output Schema window appears.

- Click  to add a row to the schema table. After you add a row, click  again to add the next row.

Enter the following values in the rows:

Key	Field Name	Type
Y	ID	Int64
N	pitch_value	Double
N	pitch_time	Stamp
N	yaw_value	Double
N	yaw_time	Stamp
N	roll_value	Double
N	roll_time	Stamp
N	velocity_value	Double
N	velocity_time	Stamp
N	lat	Double
N	long	Double

- The event captures the value of the aircraft’s pitch, yaw, roll, and velocity along with the time at which they were recorded.
- The event captures the plane’s latitude (lat) and longitude (long) when the pitch, yaw, roll, or velocity is recorded.
- Click **OK**.

10. Configure the SourceWindow window to stream events from a file called input\_long.csv that contains data from an aircraft in flight. You can find this example CSV file in the transpose\_long folder in the examples directory. To add a connector to this CSV file:

- In the right pane, click .
- Expand Input Data (Publisher) Connectors.
- Click .

The Connector Configuration window appears.

- In the **Name** field, replace the default value with `pub`.
- In the **Fsname** field, enter the path to the CSV file. For example, you might enter `/opt/sas/viya/home/SASEventStreamProcessingEngine/<release>/examples/xml/transp`

ose\_long/ input\_long.csv. Replace <release> with the release number in your installation directory path.

- In the **Fstype** drop-down list, select csv.
- Configure the pub connector's properties:
- Click **All properties**.  
The All Properties window appears.
- Select false in the Value field of the transactional property.
- Enter 1 in the **Value** field of the **blocksize** property.
- Enter %Y-%m-%d %H:%M:%S in the **Value** field of the **dateformat** property.
- Click **OK**.
- Click **OK**.

#### 11. Collapse **Input Data (Publisher) Connectors**.

#### 12. Open the input\_long.csv file and examine its contents:

```
I,N,1,23.100000,2017-08-30 15:21:06.000022,2.100000,2017-08-30 15:21:00.000007,
-1.100000,2017-08-30 15:21:00.000002,-1.100000,2017-08-30
15:21:00.000005,10.000000,10.000000
```

- `Long` mode produces one or more event per incoming event.
- The `value` and `time` associated with pitch, yaw, roll, and velocity are included in the output event. The associated latitude and longitude are passed through.
- Output events are grouped by the value of `ID`.

When you use long mode, you obtain the inverse results of wide mode. The Transpose window streams a number of events for each wide event that it receives. Input schema for the Source window must reflect combinations of fields.

#### 13. Expand **Transformations** on the **Windows** pane on the left and drag a Transpose window to the workspace.

The right pane displays the Transpose window's properties.

The input data is to be streamed through a Transpose window to create a single event (row). This event contains the pitch, velocity, roll, and yaw of each aircraft at a specific time.

#### 14. Specify a name and description for the Transpose window. In the right pane, in the Name field, change the default name to TransposeL.

#### 15. Connect the SourceW window to the TransposeL window with an edge:

- Position the cursor over the anchor point at the bottom of the SourceW window so that the anchor point color changes to white.
- Click the white anchor point, hold the mouse button down, and draw a line to the anchor point in the TransposeL window.

The TransposeL window now accepts values from the SourceW window.

#### 16. Click the TransposeL window on the workspace.

17. Configure the TransposeL window's settings:

- If necessary, click **Settings** to expand the section.
- In the **Mode** field, select **Long**.
- In the **Tag name** field, enter `TAG`.
- In the **Included values** field, enter the following tag names: **value** and **time**. After you have entered each tag name, press **Enter** to confirm its creation.
- In the **Included tags** field, enter the following tag names: **pitch**, **roll**, **yaw**, and **velocity**. After you have entered each tag name, press **Enter** to confirm its creation.
- Collapse **Settings**.

18. Create a subscribe connector:

- In the right pane, expand **Subscriber Connectors** and click .
- The Connector Configuration window is displayed
- In the **Name** field, replace the default value with `sub`.
- In the **Fsname** field, enter the path to the CSV file. For example, you might enter `/opt/sas/viya/home/SASEventStreamProcessingEngine/<release>/examples/xml/tranpose_long/output.csv`. Replace `<release>` with the release number in your installation directory path.
- In the **Fstype** drop-down list, select **csv**.
- Configure the sub connector's properties:
- Click **All properties**.
- The All Properties window appears.
- Enter `%Y-%m-%d %H:%M:%S` in the **Value** field of the **dateformat** property.
- Enter `1` in the **Value** field of the **Rate** property.
- Click **OK**.
- Click **OK**.

19. Configure the project's continuous query:

- Click .
- In the right pane, in the **Name** field, change the continuous query's default name `cq1` to `transpose_cq`.

20. Configure your project's debugging properties:

- Click .
- In the right pane, expand **Debugging**.
- In the **Enable trace server logging for this query** field, select **transposeL** from the drop-down list.

21. The model is now complete. Click  to save your model.

22. Click  **Enter Test Mode**.

A new page called **Test: transpose\_long** appears.

23. In the **Streaming Server** drop-down list, select the Streaming Server on which you want to test the model.

24. Click  **Run Test**.

The results for each window appear on separate tabs:

- The **SourceW** tab shows events representing the data received from the aircraft.
- The **TransposeL** tab shows the transposed aircraft data in long format.

25. To stop the test, click  **Stop**.

The project stops and is unloaded from the Streaming Server.

## 2.20 Working with SAS Micro Analytic Service Modules in Edge Streaming Creator

### 2.20.1 Overview

You can use SAS Micro Analytic Service modules to create input handler functions in Edge Streaming Creator using Python, DS2, and C.

Projects can also reference models that are stored in the SAS Model Manager common model repository. When a project is deployed, the model is retrieved from the SAS Model Manager common model repository and written to the Streaming Server. SAS Micro Analytic Service modules are used to accommodate the imported content that was created in SAS Model Manager. The module is uploaded and then referenced from the model's Calculate window's input handler. For more information, see "**Example Overview**".

### 2.20.2 Create a SAS Micro Analytic Service Module

To create a new module:

1. Open your project and click .
2. In the right pane, expand **SAS Micro Analytic Service Modules**.
3. Click .

The SAS Micro Analytic Service Module window appears.

4. In the **Name** field, enter a name for the module.
5. In the **Language** drop-down list, select the language that you want to use to write the module.
6. In the **Description** field, enter a description of the module.
7. In the **Function names** field, enter a comma-separated list of function names.

8. In the **Code source** field, select one of the following options:
  - **Embedded code** to enter your own code.
  - **External file** to use code located in an external file.
  - **SAS Micro Analytic Service store** to use code in an analytic store file.
9. If you selected **Embedded code** in the **Code source** field, enter your code in the **Embedded code** field.
10. If you selected **External file** in the **Code source** field, enter the file path to the external file in the **External file** field.
11. If you selected **SAS Micro Analytic store** in the **Code source** field:
  - In the **External file** field, enter the file path to the analytic store file.
  - In the **SAS Micro Analytic Service store** field, enter the module store location.
  - In the **SAS Micro Analytic Service store version** field, enter the version of the module store location.
  - In the **Module Members** field, enter your module's member names. To add a new module member, click  and fill in the applicable fields.
12. Click **OK**.

The module that you created appears in the SAS Micro Analytic Service Modules table.

### 2.20.3 Upload a SAS Micro Analytic Service Module

To upload an existing module:

1. Open your project and click .
2. In the right pane, expand **SAS Micro Analytic Service Modules**.
3. Click .
4. In the **ZIP file** field, click **Choose File**.
5. Select the SAS Model Manager ZIP file that you want to upload and click **Open**.

The Import a SAS Micro Analytic Service Module from SAS Model Manager window reloads to display information about the module's roles.
6. Review the module's role properties and modify them if necessary.
7. To copy the input schema for future use, click **Model input schema** and then copy the input schema to your clipboard. To copy the output schema for future use, click **Model output schema** and then copy the output schema to your clipboard.
8. Click **OK**.

The module that you uploaded appears in the SAS Micro Analytic Service Modules table.

## 2.20.4 Delete a SAS Micro Analytic Service Module

To delete a module, select the module that you want to delete from the SAS Micro Analytic Service Modules table and click .

The module is deleted from the SAS Micro Analytic Service Modules table.

## 2.21 Working with Input Handlers

### 2.21.1 Overview

You can register event stream input handlers for the Procedural and Calculate windows. Input handlers process incoming event streams in your model. You can import score code created in SAS Model Manager directly into a specific Calculate window in your model. You define a SAS Micro Analytic Service map in a Calculate window to bind a function to an input window. This binding acts as the input handler for the Calculate window.

### 2.21.2 Creating Input Handler Functions in Calculate Windows

To create an input handler function within a Calculate window:

1. Open the relevant project.
2. Click the relevant Calculate window.  
The right pane displays the properties of the Calculate window.
3. In the right pane, expand **Settings**.
4. In the **Calculation** drop-down list, select **User-specified**.  
A **Handlers** section appears.
5. In **Handlers**, click the row for the input window that you want to link the import handler function to.
6. Click .

The Input Handler window appears.

7. In the **Handler type** field, select one of the following handler types:
  - **SAS Micro Analytic Service** – enables you to reference a SAS Micro Analytic Service module.
  - **Import a module from SAS Model Manager** – enables you to reference a SAS Micro Analytic Service module.
  - **Import a module from SAS Model Manager ZIP file** – enables you to reference a SAS Micro Analytic Service module.

The Input Handlers window reloads to display fields that relate to the handler type that you selected.

---

**Note**

In the **Function** drop-down list, the function that is automatically selected by default is the first function in the drop-down list. Alternatively, if the drop-down list does not contain any functions, a Score method is displayed.

---

8. Update any other fields as required.
9. Click **OK**.

The Handlers section refreshes to display the imported function.

### 2.21.3 Creating Input Handler Functions in Procedural Windows

To create an input handler function within a Procedural window:

1. Open the relevant project.
2. Click the relevant Procedural window.

The right pane displays the properties of the Procedural window.
3. In the right pane, expand **Input Handlers**.
4. Click the row for the input window that you want to link the import handler function to.
5. Click .

The Input Handler window appears.

6. In the **Handler type** field, select one of the following handler types:
  - **Plug-in** – enables you to reference a plug-in library
  - **DS external** – enables you to enter DATA step code directly
  - **DS external file** – enables you to reference an external file that contains DATA step code Note:

---

**Note**

When you configure a model that contains a Procedural window that executes DATA step code, you must add the ds-initialize element to your project using the XML Editor.

The Input Handlers window reloads to display fields that relate to the handler type that you selected.

7. Update any other fields as required.
8. Click **OK**.

The Input handler functions section refreshes to display the imported function.



# Creating and Using Windows

## 3.1 Window Types

### 3.1.1 Window Types

An event stream processing model specifies how input event streams are transformed and analyzed into meaningful resultant event streams. Every model contains an engine. An engine contains one or more projects, and each project contains one or more continuous queries.

A continuous query contains one or more Source windows and one or more derived windows. Windows are connected by edges, which have an associated direction.

Edge Streaming Analytics supports a variety of window types, each having a specialized purpose.

Table 3- 1 Window Types

Window Type	Description
Source window	All event streams must enter continuous queries by being published or injected into a source window. Event streams cannot be published or injected into any other window type.
Compute window	Enables a one-to-one transformation of input events into output events through the computational manipulation of the input event stream fields.
Aggregate window	Similar to a Compute window in that non-key fields are computed. An Aggregate window uses the key field or fields for the group-by condition. All unique key field combinations form their own group within the Aggregate window. All events with the same key combination are part of the same group.
Copy window	Makes a copy of the parent window. Making a copy can be useful to set new event state retention policies. Retention policies can be set only in source and Copy windows. You can set event state retention for a Copy window only when the window is not specified to be Insert-only and when the window index is not set to <code>pi_EMPTY</code> . All subsequent sibling windows are affected by retention management. Events are deleted when they exceed the windows retention policy.
Counter window	Enables you to see how many events are streaming through your model and the rate at which they are being processed.
Filter window	Uses a registered Boolean filter function or expression. This function or expression determines what input events are allowed into the Filter window.

3.1 Window Types

Window Type	Description
Functional window	Enables you to use different types of functions to manipulate or transform the data in events. Fields in a Functional window can be hierarchical, which can be useful for applications such as web analytics.
Geofence window	Enables you to create a window to determine whether the location of an event stream is inside or near an area of interest.
Join window	Takes two input windows and a join type. Supports equijoins that are one to many, many to one, or many to many. Both inner and outer joins are supported.
Notification window	Enables you to send notifications through email, text, or multimedia message. You can create any number of delivery channels to send the notifications. A Notification window uses the same underlying language and functions as the Functional window.
Object Tracking window	Enables you to perform multi-object tracking (MOT) in real time.
Pattern window	<p>Enables the detection of events of interest (EOI). A pattern defined in this window type is an expression that logically connects declared events of interest.</p> <p>To define a pattern window, you need to define events of interests and then connect these events of interest using operators. The supported operators are "AND", "OR", "FBY", "NOT", "NOTOCCUR", and "IS". The operators can accept optional temporal conditions.</p>
Procedural window	Enables the specification of an arbitrary number of input windows and input-handler functions for each input window (that is, event stream).
Remove State window	Facilitates the transition of a stateful part of a model to a stateless part of a model.
Text Category window	<p>Enables you to categorize a text field in incoming events. A Text Category window is Insert- only. The text field could generate zero or more categories with scores.</p> <p>This object enables users who have licensed SAS Contextual Analysis to use its MCO files to initialize a Text Category window.</p>
Text Context window	<p>Enables the abstraction of classified terms from an unstructured string field.</p> <p>This object enables users who have licensed SAS Contextual Analysis to use its Liti files to initialize a Text Context window. Use this window type to analyze a string field from an event's input to find classified terms. Events generated from those terms can be analyzed by other window types. For example, a Pattern window could follow a text context window to look for tweet patterns of interest.</p>

Window Type	Description
Text Sentiment window	Determines the sentiment of text in the specified incoming text field and the probability of its occurrence. The sentiment value is "positive," "neutral," or "negative." The probability is a value between 0 and 1. A Text Sentiment window is Insert-only.  This object enables users who have licensed SAS Sentiment Analysis to use its SAM files to initialize a Text Sentiment window.
Text Topic window	Run SAS Text Miner analytics on events. Text topic windows receive and process text from documents as string fields. Text mining analytics models enter a text topic window through an analytic store file.
Transpose window	Enables you to interchange an event's rows as columns, or columns as rows.
Union window	Combines multiple event streams with the same schema into a single stream, similar to an SQL union operation.

Edge Streaming Analytics provides Calculate windows, Model Reader windows, Model Supervisor windows, Score windows, and Train windows for analytics. For more information, see **Edge Streaming Analytics**.

## 3.2 Creating a Continuous Query with Windows and Edges

Suppose that you wanted to create a continuous query to process stock trades. For that query, you want to do the following:

- Publish two events streams into the query: one for trades and another for the corresponding traders.
- Filter out trades of less than 100 shares.
- Compute the total trade cost after all shares are transacted.
- Write a file grouped by security that shows total cost.

You can create the model in Edge Streaming Creator.

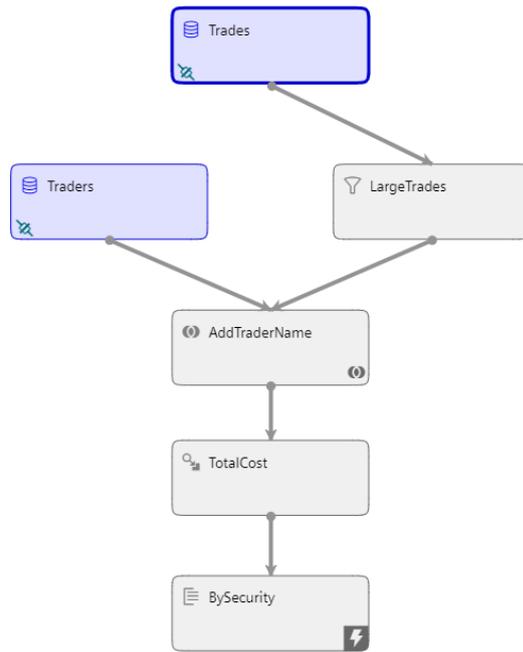


Figure 3-1 Continuous Query Diagram

Here is the corresponding Edge Streaming Analytics XML modeling language. You can find this code in \$DFESP\_HOME/examples/xml/trades\_xml.

```

<engine name='trades' port='54321' <!-- 1 -->
  <projects><project name='trades_project' pubsub='auto'
threads='4'> <!-- 2 -->
  <contqueries>
    <contquery name='trades_cq'> <!-- 3 -->
      <windows>

```

1. An engine named trades on a publish/subscribe port of 54321 is established.
2. A project named trades\_proj with a publish/subscribe mode of auto is established. Four threads are used from the available thread pool.
3. A continuous query named trades\_cq is established.

```

<window-source name='Trades' insert-only='true' index='pi_EMPTY'>
<!-- 1 -->
  <schema>
    <fields>
      <field name='tradeID' type='string' key='true' />
      <field name='security' type='string' />
      <field name='quantity' type='int32' />
      <field name='price' type='double' />
      <field name='traderID' type='int64' />
      <field name='time' type='stamp' />
    </fields>
  </schema>
<connectors>

```

```

    <connector class="fs" name="publisher"> <!-- 2 -->
      <properties>
        <property name="type">pub</property>
        <property name="fstype">csv</property>
        <property name="fsname">trades.csv</property>
        <property name="transactional">true</property>
        <property name="blocksize">1</property>
      </properties>
    </connector>
  </connectors>
</window-source>

<window-source name='Traders' insert-only='true' index="pi_EMPTY">
<!-- 3 -->
  <schema>
    <fields>
      <field name='ID' type='int64' key='true' />
      <field name='name' type='string' />
    </fields>
  </schema>
</connectors>
  <connector class="fs" name="publisher"> <!-- 4 -->
    <properties>
      <property name="type">pub</property>
      <property name="fstype">csv</property>
      <property name="fsname">traders.csv</property>
      <property name="transactional">true</property>
      <property name="blocksize">1</property>
    </properties>
  </connector>
</connectors>
</window-source>

```

1. A Source window named Trades is established with a pi\_EMPTY index type. This window streams data about securities transactions from a CSV file.
2. A file and socket connector is established to publish events into the Trades window from a file in the current working directory named trades.csv.
3. A source window named Traders is established. This window streams data about who performs those transactions. The data could be published from a file, a database, or some other source.
4. A file and socket connector is established to publish events into the Traders window from a file in the current working directory named traders.csv.

A Filter window named LargeTrades is established to receive events from the Trades window. It filters out any event that involve fewer than 100 shares.

```

<window-filter name='LargeTrades' index="pi_EMPTY">
  <expression>quantity >= 100</expression>
</window-filter>

```

A Join window named AddTraderName performs a join operation with values from the two Source windows. It matches filtered transactions with their associated traders.

```

<window-join name='AddTraderName' index="pi_EMPTY">

```

```

<join type="leftouter" no-regenerate="true">
  <conditions>
    <fields left='traderID' right='ID' />
  </conditions>
</join>
<output>
  <field-selection name='security' source='l_security' />
  <field-selection name='quantity' source='l_quantity' />
  <field-selection name='price' source='l_price' />
  <field-selection name='traderID' source='l_traderID' />
  <field-selection name='time' source='l_time' />
  <field-selection name='name' source='r_name' />
</output>
</window-join>

```

A Compute window named TotalCost uses data from the Join window to calculate the cost of the transaction.

```

<window-compute name='TotalCost' index='pi_EMPTY'>
  <schema>
    <fields>
      <field name='tradeID' type='string' key='true' />
      <field name='security' type='string' /> <!-- 1 -->
      <field name='quantity' type='int32' />
      <field name='price' type='double' />
      <field name='totalCost' type='double' />
      <field name='traderID' type='int64' />
      <field name='time' type='stamp' />
      <field name='name' type='string' />
    </fields>
  </schema>
  <output>
    <field-expr>security</field-expr>
    <field-expr>quantity</field-expr>
    <field-expr>price</field-expr>
    <field-expr>price*quantity</field-expr> <!-- 2 -->
    <field-expr>traderID</field-expr>
    <field-expr>time</field-expr>
    <field-expr>name</field-expr>
  </output>
</window-compute>

```

1. This and the following fields are the non-key fields.
2. This is how totalCost is computed.

An Aggregate window is established named BySecurity. The ESP\_aSum function is used to sum total quantity and cost. A subscriber connector publishes the output to a CSV file named result.out.

```

<window-aggregate name='BySecurity'>
  <schema>
    <fields>
      <field name='security' type='string' key='true' />
      <field name='quantityTotal' type='double' />
      <field name='costTotal' type='double' />
    </fields>
  </schema>
<output>

```

```

    <field-expr>ESP_aSum(quantity)</field-expr>
    <field-expr>ESP_aSum(totalCost)</field-expr>
</output>
<connectors>
  <connector class="fs" name="sub">
    <properties>
      <property name="type">sub</property>
      <property name="fstype">csv</property>
      <property name="header">>true</property>
      <property name="fsname">result.csv</property>
      <property name="snapshot">>true</property>
    </properties>
  </connector>
</connectors>
</window-aggregate>
</windows>

```

Edges connect the windows.

```

  <edges>
    <edge source='LargeTrades' target='AddTraderName' /> <!--
1 -->
    <edge source='Traders' target='AddTraderName' />
    <edge source='Trades' target='LargeTrades' /> <!-- 2 -->
    <edge source='AddTraderName' target='TotalCost' />
    <edge source='TotalCost' target='BySecurity' />
  </edges>
</contquery>
</contqueries>
</project>
</projects>
</engine>

```

1. The Filter window and the Traders Source window flow events to the Join window.
2. The Trades window flows events to the Filter window. The Join window flows events to the Compute window, which flows events to the Aggregate window.

The output CSV file shows two retained events. The first indicates that, after all trades were transacted, 1000 shares of IBM were sold at a total cost of \$100,100.00. The second indicates that 750 shares of SAP were sold at a total cost of \$25,650.00.

security	quantity	Total	costTotal		
I	N	ibm	1000	100100	
I	N	sap	750	25650	
UB	N	ibm	2000	200300	
D	N	ibm	1000	100100	
UB	N	ibm	3000	300600	
D	N	ibm	2000	200300	
UB	N	ibm	4000	401000	
D	N	ibm	3000	300600	
UB	N	sap	1750	59950	
D	N	sap	750	25650	
UB	N	ibm	5000	501300	
D	N	ibm	4000	401000	
UB	N	sap	2750	91950	
D	N	sap	1750	59950	
UB	N	ibm	6000	601300	
D	N	ibm	5000	501300	

Figure 3-2 Output from the Aggregate Window

## 3.3 Using Expressions

### 3.3.1 Overview to Expressions

Event stream processing applications can use expressions to define the following:

- Filter conditions in Filter windows.
- Non-key field calculations in Compute, Aggregate, and Join windows.
- Matches to window patterns in events of interest.
- Window-output splitter-slot calculations (for example, use an expression to evaluate where to send a generated event)

You can use user-defined functions instead of expressions in all of these cases except for pattern matching. With pattern matching, you must use expressions.

Writing and registering expressions with their respective windows can be easier than writing the equivalent user-defined functions in C. Expressions run more slowly than functions. For very low-latency applications, you can use user-defined functions to minimize the overhead of expression parsing and processing.

Use prototype expressions whenever possible. Based on results, optimize them as necessary or exchange them for functions. Most applications use expressions instead of functions, but you can use functions when faster performance is critical.

For information about how to specify expressions, refer to the [Expression Language: Reference Guide](#).

---

#### Note

Edge Streaming Analytics uses a subset of the functionality that is documented for the Expression Engine Language. This subset is robust for the needs of event stream processing.

---

Each expression window and window splitter has its own expression engine instance. Expression engine instances run window and splitter expressions for each event that is processed by the window. You can initialize expression engines before they are used by expression windows or window splitters (that is, before any events stream into those windows). Expression engine initialization can be useful to declare and initialize expression engine variables used in expression window or window splitter expressions. They can also be useful to declare regular expressions used in expressions.

To initialize expression engines for expression windows and window splitters, use the `<expr-initialize>` element in your XML code. For example, the following XML code initializes a user-defined function for a splitter:

```

<expr-initialize>
  <udfs>
    <udf name='udf1' type='int32'>
      <![CDATA[private integer p p = parameter(1); return
p%2]]>
    </udf>
  </udfs>

```

```
</expr-initialize>
```

You can find examples in `$DFESP_HOME/examples/xml/splitter_initexp_xml`, `$DFESP_HOME/examples/xml/splitter_udf_xml`, and `$DFESP_HOME/examples/xml/regex_xml`.

### 3.3.2 Understanding Data Type Mappings

An exact data type mapping does not exist between the data types supported by the Edge Streaming Analytics API and those supported by the Expression Engine Language.

The following table shows the supported data type mappings:

Table 3-2 Expression Data Type Mappings Table

Event Stream Processing Expressions	Expressions	Notes and Restrictions
String (utf8)	String (utf8)	Strings passed to expressions might be truncated to 1024 bytes. Use the window-specific attribute <code>exp-max-string= 'N'</code> or the C++ window method <code>dfESPwindow::set_EXP_size rMax (int N)</code> to set the maximum size of expression strings for a window.
date (second granularity)	date (second granularity)	Seconds granularity
timestamp (microsecond granularity)	date (second granularity)	Constant milliseconds in <code>dfExpressions</code> not supported
Int32 (32 bit)	Integer (64 bit)	64-bit conversion for <code>dfExpressions</code>
Int64 (64 bit)	Integer (64 bit)	64-bit, no conversion
double (64 bit IEEE)	real (192 bit fixed decimal)	real 192-bit fixed point, double 64-bit float
money (192 bit fixed decimal)	real (192 bit fixed decimal)	192-bit fixed point, no conversion

### 3.3.3 Using Event Metadata in Expressions

Edge Streaming Analytics provides a set of reserved words that you can use to access an event's metadata. You can use these reserved words in filter, compute, and Join window expressions and in window output splitter expressions. The metadata is not available to Pattern window expressions because Pattern windows are insert-only.

Table 3- 3 Reserved Words to Access an Event's Metadata

Reserved Word	Opcode
<p>ESP_OPCODE</p> <p>Use this reserved word to obtain the opcode of an event in a given window.</p>	<p>I — Insert</p> <p>U — Update</p> <p>P — Upsert</p> <p>D— Delete</p> <p>SD — Safe Delete</p> <p>Note: A safe delete does not generate a "key not found" error. Note: Values are case-sensitive.</p>
<p>ESP_FLAGS</p> <p>Use this reserved word in expressions to get the flags of an event in a given window.</p>	<p>N — Normal</p> <p>P — Partial</p> <p>R — Retention Delete</p>

### 3.3.4 Using Expression Language Global Functions

The Expression Engine Language supports global functions, also called user-defined functions (UDFs). You can register them as global functions and reference them from any expression window or window splitter expression.

For more information about global functions, see [Expression Language: Reference Guide](#).

There are two Edge Streaming Analytics functions to which you can register global functions:

- `dfESPexpression_window::regWindowExpUDF(udfString, udfName, udfRetType)`
- `dfESPwindow::regSplitterExpUDF(udfString, udfName, udfRetType)`

After you register global functions for a window splitter or an expression window, a splitter expression or a window expression can reference the `udfName`. The `udfName` is replaced with the `udfString` as events are processed.

Filter, Compute, Join, and Pattern expression windows support the use of global functions. Aggregate windows do not support global functions because their output fields are create-only through aggregate functions. All windows support global functions for output splitters on the specified window.

### 3.3.5 Using SAS Data Quality Functions

Event stream processing expressions support the use of the SAS Data Quality functions. The following functions are fully documented in the [Expression Language: Reference Guide](#).

- `dq.case`
- `dq.gender`
- `dq.getlasterror`
- `dq.identify`
- `dq.initialize`
- `dq.loadqkb`
- `dq.matchcode`
- `dq.pattern`
- `dq.standardize`

You must set two environment variables as follows:

- `DFESP_QKB` to the share folder under the SAS Data Quality installation. After you have installed SAS Data Quality on Linux systems, this share folder is `/QKB_root/data/ci/qkb_version_number_no_dots` (for example, `/QKB/data/ci/22`).
- `DFESP_QKB_LIC` to the full file pathname of the SAS Data Quality license.

After you set up SAS Data Quality for Edge Streaming Analytics, you can include these functions in any of your event stream processing expressions. These functions are typically used to normalize event fields in the non-key field calculation expressions in a Compute window.

## 3.4 Using Source Windows

### 3.4.1 Overview to Source Windows

At least one Source window is required for each continuous query. All event streams enter continuous queries by being published or injected into a Source window. Event streams cannot be published or injected into any other window type.

Source windows are typically connected to one or more derived windows. Derived windows can detect patterns in the data, transform the data, aggregate the data, analyze the data, or perform computations based on the data.

Source windows accept streaming data or raw data files through in-process connectors or executable adapters. Source windows can also accept data from the publish/subscribe API, HTTP clients, or by injection from the C++ Modeling API.

For information about the XML elements associated with Source windows, see “**window-source**”

### 3.4.2 Defining Event Index Types

Source windows can have a primary index and a retention index. There are seven stateful index types and one stateless index type. The primary index type of a source window affects the performance of the rest of the project.

For example, when a Source window has index type `pi_RBTREE`, the window is stateful and retains all incoming events. Retaining events at the Source window without a retention policy uses substantially more memory as data is read into the continuous query than when the Source window is stateless.

For more information about index types, see "[Understanding Primary and Specialized Indexes](#)".

### 3.4.3 Retention Policies in Source Windows

Retention policies limit the flow of incoming data by time or event count, and thus can control the total number of events that stream through the model. Events are deleted automatically by the engine when they exceed the window's retention policy.

You can define a retention policy in stateful Source and Copy windows. To use retention policies, the window cannot be specified as Insert-only and the index type cannot be specified as `pi_EMPTY`. Usually, you want to follow any insert-only Source window with a Copy window with a retention policy.

For more information about retention policies, see "[Understanding Retention](#)".

### 3.4.4 Propagation of Insert-Only Processing

You can explicitly set a Source window to accept only Insert events. This can optimize event stream processing. Insert-only processing propagates to all derived windows unless one of the following conditions is met:

- A window is determined not to produce Insert-only data (for example, a window with retention).
- The Streaming Server cannot determine whether the derived window produces only Inserts:
  - When a Procedural or Calculate window has set `algorithm='MAS'`, it runs an arbitrary block of user-written code that could result in a non-Insert event. If you are certain that your code produces only Inserts, explicitly add the attribute `produces-only-inserts='true'` to the XML specification of the window.
  - When a Functional window uses a function that changes the opcode of a produced event. You can override this with the attribute `produces-only-inserts='true | false'`.
  - The Aggregate window produces numerous Updates. It always sets `produces-only-inserts='false'`.

### 3.4.5 Automatically Generating Key Values

Source windows can automatically generate identification keys for incoming events. To automatically generate key values, the Source window must be insert-only and have only one key with type INT64 or STRING. The `autogen-key` attribute of the window-source element must be set to true.

For INT64 keys, the value is an incremental count (0, 1, 2, ...). For STRING keys, the value is a Globally Unique Identifier (GUID).

### 3.4.6 The Publisher Connector

The publisher connector is unique to the Source window. It determines the following:

- the path of the data source.
- the data type of the events received from the data source.
- other important properties that are related to translating incoming data into events that are used throughout the project.

The Source window's publisher connector determines how the data is read and in what format events are pushed to derived windows. In XML code, the required and optional properties of the connectors are defined in the connector element.

The fields defined by the schema of the Source window determine the structure of the data read into the project and used by derived windows.

### 3.4.7 Enabling Metering Windows

An engine can contain a metering window to track the number of events processed (and related timestamps) by all of the Source windows in a model. To enable this functionality, you must define the field `enableMetaProject` with the value `true` in `esp-properties.yml`. This sets up the metering window in a continuous query named `_meta_`, which is contained in a project named `_meta_`.

Source windows inject events into the metering window at a default interval of five seconds. Subscribing applications can subscribe to the metering window as they do for any other window.

The metering window is itself a special Source window named `_eventmetering_`. It has the following schema:

```
"project*:string,query*:string,window*:string,currenttime:stamp,lasttime:stamp,numevents:int64"
```

You can append string fields to the end of the metering window schema when you start a metering server. Subsequently, all metering events generated by the metering window contain those fields. You can define or redefine the values of those fields at any time. If not defined at engine start-up, the values are null until you define them. You can also reconfigure the default metering interval of five seconds.

When you run the metering server and its `meteringhost` and `meteringport` parameters are set, each update to a metering window is forwarded to the metering server for aggregation.

### 3.4.8 Using Singletons

A Source window that is defined within a continuous query but not included in an edge element is called a singleton. Singletons can be useful to validate a connection between the outside world and the Streaming Server and validating input formats when designing a project.

For example, suppose that the primary input into your project is a message bus that streams events that are in JSON format. You have an idea of the format but are uncertain about the actual content. In this case, you can design a singleton with a publisher connector to test the connection, inspect the JSON passing parameters, and inspect the incoming data. You could use Edge Streaming Creator in test mode to inspect data flows.

Afterward, you might want to collect a snapshot of events to a static data set for closer inspection. You could add a publisher connector to the singleton to store data to a specified file.

You specify whether or not to include singletons in a continuous query using the `include-singletons` attribute of the `contquery` element. For more information, see “contquery”.

## 3.5 Using Aggregate Windows

### 3.5.1 Overview to Aggregate Windows

Aggregate windows are similar to Compute windows in that non-key fields are computed. However, you must specify key fields of Aggregate windows; they are not inherited from the input window. Those key fields must correspond to existing fields in the input event. Incoming events are placed into aggregate groups. Each event in an aggregate group has identical values for the specified key fields.

For example, suppose that the following schema is specified for input events:  
`"ID*:int32,symbol:string,quantity:int32,price:double"`

Now suppose that you specify the schema for an Aggregate window as follows:  
`"symbol*:string,totalQuant:int32,maxPrice:double"`

When events arrive in the Aggregate window, they are placed into aggregate groups based on the value of the `symbol` field. Aggregate field calculation functions or expressions that are registered to the Aggregate window must appear in the non-key fields, which in this example are `totalQuant` and `maxPrice`. Either expressions or functions must be used for all of the non-key fields. They cannot be mixed. The functions or expressions are called with a group of events as one of their arguments every time a new event comes in and modifies one or more groups.

These groups are internally maintained in the `dfESPwindow_aggregate` class as `dfESPgroupstate` objects. Each group is collapsed every time that a new event is added or removed from a group by running the specified aggregate functions or expressions on all non-key fields. The Aggregate window produces one aggregated event per group.

For information about the XML elements associated with Aggregate windows, see “window-aggregate”.

### 3.5.2 Flow of Operations

The flow of operations while processing an Aggregate window is as follows:

1. An event, E arrives and the appropriate group is found, called G. The group is found by looking at the values in the incoming event that correspond to the key fields in the Aggregate window.
2. The event E is merged into the group G. The key of the output event is formed from the group-by fields of G.
3. Each non-key field of the output schema is computed by calling an aggregate function with the group G as input. The aggregate function computes a scalar value for the corresponding non-key field.
4. The correct output event is generated and output.

### 3.5.3 Using Aggregate Functions

#### Overview to Using Aggregate Functions

During aggregation, events that enter an Aggregate window are placed into a group based on the Aggregate window's key fields. The aggregate functions provided with Edge Streaming Analytics are run on each group to compute each non-key field of the Aggregate window.

The functions that are specified for non-key fields of the Aggregate window are special functions. They operate on groups of values and collapse the group to a single scalar value.

In the following diagram, the key of the Aggregate window is `symbol`. The Aggregate window has only one non-key field, `sumQuant`. This field holds the sum of the field `quant` that arrives from the Source window.

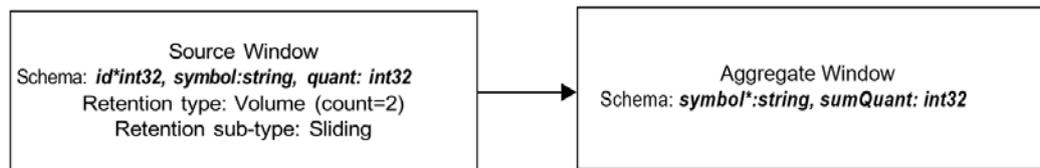


Figure 3-3 Example of Aggregation

The function that computes sums of field values is `ESP_aSum(fieldname)`. Here, the Aggregate window has one non-key field that is computed as `ESP_aSum(quant)`. Conceptually, when an event enters the Aggregate window, it is added to the group, and the function `ESP_aSum(quant)` is run, producing a new sum for the group.

You can write your own aggregate functions. For more information, see “Writing Aggregate Functions to Embed in Applications”.

#### Aggregate Functions for Aggregate Window Field Calculation Expressions

The following aggregate functions are available for Aggregate window field calculation expressions. Additive functions can be calculated from retained state and the values determined from the incoming event. They do not need to maintain a group state. This means that if these are the only functions used in an Aggregate window, special

optimizations are performed. Speed-ups of an order of magnitude in the Aggregate window processing can occur. Some functions are additive regardless of the opcode of the incoming event. Others are additive only when they get Inserts.

Table 3- 4 Aggregate Functions

Aggregate Function	Additive	Additive for Inserts	Returns
ESP_aAve(fieldname)	True	True	Average of the group
ESP_aCat(fieldname, stringConstant)	False	True	A concatenation of fieldname values for each event in an aggregation group using the specified stringConstant as the separator.  For example, ESP_aCat(ID, " ") returns a " " separated list of all the IDs that make up the aggregation group.
ESP_aCount()	True	True	Number of events in the group
ESP_aCountDistinct(field name)	True	True	Number of distinct non-null values in the column specified by field name within a group
ESP_aCountNonNull(fieldname)	False	True	Number of events with non-null values in the column for the specified field name within the group
ESP_aCountNull(fieldname)	False	True	Number of events with null values in the column for the specified field name within the group
ESP_aCountOpcodes(opcode)	True	True	Count of the number of events matching opcode for group
ESP_aFirst(fieldname)	False	True	First event added to the group
ESP_aGUID()	True	True	A unique identifier
ESP_aLag(fieldname, lag_value)	False	True	A lag value where the following holds: <ul style="list-style-type: none"> <li>ESP_aLag(fieldname,0) == ESPaLast(fieldname)</li> <li>ESP_aLag(fieldname,1) returns the second lag. This is the previous value of fieldname that affected the group.</li> </ul>
ESP_aLast(fieldname)	False	True	Field from the last record with non-null value that affected the group

Aggregate Function	Additive	Additive for Inserts	Returns
ESP_aLastNonNull(fieldname)	False	True	Field from the last record with non-null value that affected the group
ESP_aLastOpcode()	True	True	The opcode of the last record to affect the group
ESP_aMax(fieldname)	False	True	Maximum of the group
ESP_aMin(fieldname)	False	True	Minimum of the group
ESP_aMode(fieldname)	True	True	Mode, or most popular of the group
ESP_aStd(fieldname)	True	True	Standard deviation of the group
ESP_aSum(fieldname)	True	True	Sum of the group
ESP_aAve(weight_fieldname, payload_fieldname)	True	True	Weighted group average

All aggregate functions are additive when you feed them Insert events. Thus, consider the following points with regard to model performance:

- When an insert-only Aggregate window has a key with an unbounded number of elements, the window experiences unlimited memory growth. This cannot be automatically detected by a streaming Server, because you cannot predict the values passed by incoming events. In this case a WARNING is issued.
- For Aggregation windows that are not Insert-only, the window can be one of the following:
  - Additive, that is, one that uses only the aggregation functions that are marked as always additive. If the cardinality of the keys is unbounded, then the window experiences unlimited memory growth. Because it is not receiving Inserts, no WARNING is issued. Here, you should put a Copy window with retention in front of the Aggregate window in order to control the total number of events in the Aggregate window.
  - Nonadditive, that is, one that uses nonadditive aggregation functions such as ESP\_aMax or ESP\_aMin. In this case memory growth is primarily driven by the maximum number of events that are active in the Aggregation window at any time. For this case, the Aggregation window maintains a copy of each event that streams in. It also keeps track of what group the event belongs to. Here, it is recommended that you precede the Aggregation window with a Copy window with retention. That ensures that there is always a finite number of events in the Aggregate window. The number of events would be determined by retention policy.

Event history also affects performance:

- Some aggregate functions (for example, ESP\_aMin and ESP\_aMax) require source events to be stored (history) in order to process Updates and Deletes.
- All aggregate functions can handle Insert-only event streams, like sensor readings, without history.
- For event streams with Updates and Deletes, use only aggregate functions that do not require history (for example, ESP\_aSum, ESP\_aAve) for best performance.

Source event history, when applicable, is kept in an internal index.

You can easily use aggregate functions for non-key field calculation expressions. For example:

```

<window-aggregate name='brokerAlertsAggr' index='pi_HASH'>

<schemastring>brokerName*:string,frontRunningBuy:int32,frontRunningSell:int32,-
openMarking:int32,closeMarking:int32,restrictedTrades:int32,total:int64
  </schema-string>
  <output>
    <field-expr>ESP_aSum(frontRunningBuy)</field-expr>
    <field-expr>ESP_aSum(frontRunningSell)</field-expr>
    <field-expr>ESP_aSum(openMarking)</field-expr>
    <field-expr>ESP_aSum(closeMarking)</field-expr>
    <field-expr>ESP_aSum(restrictedTrades)</field-expr>
    <field-expr>ESP_aSum(total)</field-expr>
  </output>
</window-aggregate>

```

---

**Note**

In `ESP_aSum`, `ESP_aMax`, `ESP_aMin`, `ESP_aAve`, `ESP_aStd`, and `ESP_aWAve`, null values in a field are ignored. Therefore, they do not contribute to the computation.

---

**Using an Aggregate Function to Add Statistics to an Incoming Event**

You can use the `ESP_aLast(fieldName)` aggregate function to pass incoming fields into an aggregate event. This can be useful to add statistics to events through the Aggregate window without having to use an Aggregate window followed by a Join window. Alternatively, using a Join window after an Aggregate window joins the aggregate calculations or event to the same event that feeds into the Aggregate window. But the results in that case might not be optimal.

Suppose that you are processing stock transactions. The Source window that processes incoming events that contains several fields.

```

<schema>
  <fields>
    <field name="ID" type="int32" key="true"/>
    <field name="symbol" type="string"/>
    <field name="currency" type="int32"/>
    <field name="update" type="int64"/>
    <field name="msecs" type="int32"/>
    <field name="price" type="double"/>
    <field name="quant" type="int32"/>
    <field name="venue" type="int32"/>
    <field name="broker" type="int32"/>
    <field name="buyer" type="int32"/>
    <field name="seller" type="int32"/>
    <field name="buysellflg" type="int32"/>
  </fields>
</schema>

```

You want to restrict the incoming event schema for the Aggregate window to three variables: `symbol`, `price`, and `quant`. To those variables, you want to add two aggregate statistics:

`priceAVE` and `quantMAX`.

```

<schema>
<fields>

```

```
<field name="symbol" type="string" key="true"/>
<field name="price" type="double"/>
<field name="quant" type="int32"/>
<field name="priceAVE" type="double"/>
<field name="quantMAX" type="int32"/>
</fields>
</schema>
```

---

**Note**

The group-by is the key of the aggregation, which in this case is symbol.

---

```
<output>
<field-expr>ESP_aLast(price)</field-expr>
<field-expr>ESP_aLast(quant)</field-expr>
<field-expr>ESP_aAve(price)</field-expr>
<field-expr>ESP_aMax(quant)</field-expr>
</output>
```

Suppose that the following event streams into the Source window:

```
i n 21 SAIA 0 55110 932 16.2328 100 4 92 3 58 1
```

The following event is subsequently written by the Aggregate window:

```
I N SAIA 16.2328 100 16.2328 100
```

Later, the following event streams into the Source window:

```
i n 392 SAIA 7 55110 934 15.1296 400 3 64 5 75 1
```

The following event is then written by the Aggregate window:

```
U N SAIA 15.1296 400 15.6812 400
```

Lastly, the following event streams into the Source window:

```
i I n 531 SAIA 1 55110 934 15.2872 1100 5 52 73 82 0
```

The following event is then by the Aggregate window:

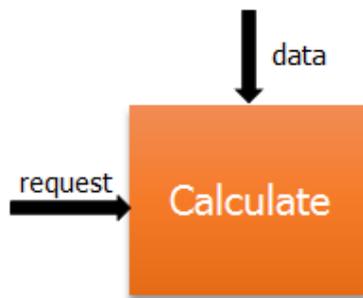
```
U N SAIA 15.2872 1100 15.549867 1100
```

By using `ESP_aLast(fieldName)` and then adding the aggregate fields of interest, you can avoid the subsequent Join window. This makes the modeling cleaner.

## 3.6 Using Calculate Windows

### 3.6.1 Overview to Calculate Windows

Calculate windows create real time, running statistics that are based on established analytical techniques. They receive data events and publishes newly transformed score data into output events. (No role is assigned to the outgoing edges, so they do not appear in the diagram.) Calculate windows can also receive request events.



Calculate windows are designed for data normalization and transformation methods, as well as for learning models that bundle training and scoring together.

For more information, see [Edge Streaming Analytics: Using Streaming Analytics](#).

### 3.6.2 Migrating from a Procedural Window to a Calculate Window

Support for SAS Micro Analytic Service (MAS) modules and stores has moved from the Procedural window to the Calculate window.

To migrate from a Procedural Window to a Calculate Window requires minimal change to your XML code.

**Example Code A.1** Procedural Window

```
<window-procedural name='pw_01'>
...
</window-procedural>
<edge source='w_source' target='pw_01' />
```

**Example Code A.2** Calculate Window

```
<window-calculate name='pw_01' algorithm='MAS'>
...
</window-calculate>
<edge source='w_source' target='pw_01' role='data' />
```

You can use the `dfesp_xml_migrate` command to convert Procedural windows to Calculate windows in your model code. However, you must manually convert DS2 code in table server mode to code that uses SAS Micro Analytic Service modules before running the tool. For more information about this command, see "Migrating XML Code across Product Releases".

## 3.7 Using Compute Windows

### 3.7.1 Overview to Compute Windows

Use a Compute window to enable a one-to-one transformation of input events to output events through the computational manipulation of the input event stream fields. You can use the Compute window to project input fields from one event to a new event and to augment the new event with fields that result from a calculation.

The set of key fields can be changed within the Compute window, but use this capability with caution. When you make a key field change within the compute window, the Inserts, Updates, and Deletes for the input events' keys must be equivalent Inserts, Update, and Deletes for the new key set.

For information about the XML elements associated with Compute windows, see “**window-compute**” and “XML Language Elements Relevant to Compute and Filter Windows”.

### 3.7.2 Using Compute Functions

Events that enter a Compute window are placed into a group based on the Compute window's key fields. Functions or expressions are run on each group to compute each non-key field of the Compute window.

Compute windows perform computations on input streams using expressions, user-defined functions, or plug-in functions. Output fields from Compute windows can be pushed to another window or to a subscribe connector.

User-defined functions are specified in the udf of the udfs and expr-initialize elements at the beginning of window-compute.

Registered plug-in functions are specified in the field-plug XML language element within the output element of window-compute. These functions are sourced from a shared library, such as libmethod.so or libmethod.dll found in the DFESP\_HOME directory.

## 3.8 Using Copy Windows

### 3.8.1 Overview to Copy Windows

Use a Copy windows to copy a parent window of any other type. They are useful to retain events with specified event state retention policies. You can set retention policies only in Source and Copy windows.

For information about the XML elements associated with Copy windows, see “**window-copy**”.

### 3.8.2 Retention Policies in Copy Windows

You can set event state retention for a Copy window only when the window is not specified to be insert-only and when the window index is not set to `pi_EMPTY`. All subsequent sibling windows are affected by retention management. Events are deleted when they exceed the windows retention policy.

For more information about retention policies, see “Understanding Retention”.

## 3.9 Using Counter Windows

Use a Counter window to determine how many events are streaming through your model and the rate at which they are being processed.

For information about the XML elements associated with Counter windows, see "window-counter" .

You cannot configure the schema for a Counter window; it is hardcoded as follows:

```
"input*:string,totalCount:int64,totalSeconds:double,totalRate:double,intervalCount:
int64,intervalSeconds:double,intervalRate:double" .
```

The value of input is the name of the window that sent the event to the Counter window.

The opcode for generated events is based on the index of the Counter window. If the index is `pi_EMPTY`, the opcode is Insert. For any other index value, the opcode is Upsert.

When you specify a `count-interval`, the Counter window reports performance statistics regularly at that interval. Event generation can be driven by either the arrival of an event or by the window receiving a heartbeat. The window checks to see whether it is time to report the values and generate an event. This event contains overall values plus the interval values:

```
<window-counter name='counter'
    count-interval='2 seconds'
    clear-interval='30 seconds' />
<event opcode='upsert' window='trades/trades/counter'>
    <value name='input'>trades</value>
    <value name='intervalCount'>288215</value>
    <value name='intervalRate'>144108</value>
    <value name='intervalSeconds'>2</value>
    <value name='totalCount'>794312</value>
    <value name='totalRate'>132385</value>
    <value name='totalSeconds'>6</value>
</event>
```

If you do not specify `count-interval`, an event with performance numbers is generated each time the window receives a heartbeat. This event contains only overall values:

```
<window-counter name='counter' />
<event opcode='upsert' window='trades/trades/counter'>
    <value name='input'>trades</value>
    <value name='totalCount'>7815189</value>
    <value name='totalRate'>132461</value>
    <value name='totalSeconds'>59</value>
</event>
```

To use a Counter window, add an edge with the Counter window as the target and the window to monitor as the source. You can connect multiple windows to the same Counter window. Edge Streaming Viewer (future product) can subscribe to the Counter window to show the results. Alternatively, you can add the Counter window to the trace attribute of the `<contquery>` element that prints formatted events to standard output.

## 3.10 Using Filter Windows

Filter windows use expressions, user-defined functions (global functions), and registered plug-in functions to determine what input events are permitted to stream through. These functions and expressions are called filter conditions.

For more information about available filter conditions, see “Overview to Expressions”.

For information about the XML elements associated with Filter windows, see “window-filter” and “XML Language Elements Relevant to Compute and Filter Windows” .

## 3.11 Using Functional Windows

### 3.11.1 Overview to Functional Windows

Use a Functional window to call different types of functions in order to manipulate or transform event data. You define a schema and then a function context that contains functions and supporting entities such as regular expressions, XML, and JSON. When an event enters a Functional window, the window looks for a function with a name that corresponds to each field in its schema. If the function exists, then it is run. The resulting value is entered into the output event. If no function is specified for a field, and a field with the same name exists in the input schema, then the input value is copied directly into the output event.

For information about the XML elements associated with Functional windows, see “window-functional” and “XML Language Elements Relevant to Functional Windows” .

For an example showing how to use Functional windows to monitor stock trades, go to the Edge Streaming Analytics.

Use the XML element `<generate>` to specify a function to run in order to determine whether you want to generate an event from an input event.

Generate multiple output events from a single input event using the `<event-loop>` element. You can specify a function to create some type of data and then grab any number of entities from that created data. For each of these entities, you can generate an event using a function context specific to that event loop.

For a complete reference on support functions available for defining functions in functional windows, see “Functional Window and Notification Window Support Functions”.

### 3.11.2 Using Event Loops

Event loops enable you to generate any number of events from a single input event. You can specify any number of event loops. Each loop deals with a particular type of data.

For each input event, a Functional window does the following for each event loop entry:

1. Uses a function or reference to generate the data to be used as input to the loop. For example, in an `event-loop-xml` loop, you would specify the `use-xml` element to generate valid XML. This content can be either a function or a reference to a property in the window's function-context.
2. Applies an appropriate expression, such as XPATH or JSON, to the data to retrieve 0 or more entities.
3. For each of these entities, sets a data item specified by the data attribute to the string value of the entity. Then, any functions in the function-context are run and an event is generated. Any property or event value in the window's function context is accessible to the loop's function context. Also, the variable specified by the data attribute is accessible via '\$' notation.

The event loop creates a contextual XML object for each iteration. You can refer to this object as `#__context` within a function. The name space of this object is set to that of the top-level container. If you use a string representation instead of the `#__context` object, you cannot set the name space.

### 3.11.3 Understanding and Using Function Context

#### Overview to Function Context

Function context enables you to define functions within a Functional window. You can use regular expressions, XML and XPATH, JSON, and other capabilities to transform data from different types of complex input into usable output.

#### Types of Functions You Can Use

You can use two types of functions within a function context:

- general functions
- functions specific to event stream processing

You can reference event fields in either the input event or the output event using the '\$' notation: `${name of field}`

Table 3- 5 Data Mappings Relevant to Using Functions in a Function Context

string	ESP_UTF8STR
float	ESP_DOUBLE
long	ESP_INT64
integer	ESP_INT32
Boolean	ESP_INT32

For example, suppose that you have a name field in the input event and you want to generate an `occupation` field in the output event. You could code the function as follows using the `ifNext` and `equals` support functions:

```
<function name='occupation'>
ifNext
(
  equals($name, 'larry'), 'plumber',
  equals($name, 'moe'), 'electrician',
  equals($name, 'curly'), 'carpenter'
)
</function>
```

You can also reference fields in the output event. Continuing with this example, perhaps you want to add the `hourlyWage` field to the output event depending on the value of `occupation`, again using the `ifNext` and `equals` support functions:

```
<function name='hourlyWage'>
ifNext
(
  equals($occupation, 'plumber'), 85.0,
  equals($occupation, 'electrician'), 110.0,
  equals($occupation, 'carpenter'), 60.0
)
</function>
```

---

### Note

It is critical to pay attention to the sequence of fields when you define functions. If a function references an output event field, then that field must be computed before the referring field.

---

### Using Expressions

Use the `<expressions>` element to specify POSIX regular expressions that are compiled a single time.

```
<expressions>
  <expression
name='expname'>[posix_regular_expression]</expression>
  ...
</expressions>
```

After you specify an expression, you can reference it from within a function using the following notation and the `rgx` support function:

```
<function name='myData'>rgx(#expname,$inputField,1)</function>
```

---

### Note

POSIX regular expressions must follow the standards specified by the [IEEE](#).

---

Suppose that you were getting a data field that contained a URI, and you wanted to extract the protocol from the URI. If you use `<function name='protocol'>rgx('(.*):',$uri,1)</function>`, the regular expression is compiled each time that the function is run. However, if you use the following code:

```
<expressions>
  <expression name='getProtocol'>(.*):</expression>
</expressions>
```

```
<function name='protocol'>rgx(#getProtocol,$uri,1)</function>
```

The expression is compiled a single time and used each time that the function is run.

### Specifying Properties

Properties are similar to expressions in that they are referenced from within functions using the '#' notation: #[property-type]

There are five types of properties:

- map executes the function to generate a map of name-value pairs to be used for value lookups by name
- set executes the function to generate a set of strings to be used for value lookups
- XML executes the function to generate an XML object that can be used for XPATH queries.
- JSON executes the function to generate a JSON object that can be used for JSON lookups
- string executes the function to generate a string for general use in functions

Each property is generated using functions. These functions can reference properties defined before them in the XML.

Table 3- 6 How to Code Each Property Type

Property Type	Description
<pre>&lt;property-map name='name' outer='outdelim' inner='indelim'&gt;[code]&lt;/property-map&gt;</pre>	<ul style="list-style-type: none"> <li>• name - the name of the property</li> <li>• outer - the outer delimiter to use in parsing the data</li> <li>• inner - the inner delimiter to use in parsing the data</li> <li>• code - the function to run to generate the data to be parsed into a name-value map</li> </ul> <p>For example, suppose there exists an input field data that looks like this: <code>firstname=joe;lastname=smith;occupation=software</code></p> <p>You could create the following property-map:  <pre>&lt;property-map name='myMap' outer=';' inner=' '&gt;\$data&lt;/property-map&gt;</pre></p>
<pre>&lt;property-xml name='name'&gt;[code]&lt;/ property-xml&gt;</pre>	<ul style="list-style-type: none"> <li>• name - the name of the property</li> <li>• code - the function to run to generate valid XML</li> </ul>
<pre>&lt;property-json name='name'&gt;[code]&lt;/ property-json&gt;</pre>	<ul style="list-style-type: none"> <li>• name - the name of the property</li> <li>• code - the function to run to generate valid JSON</li> </ul>

Property Type	Description
<code>&lt;property-string name='name'&gt;[code]&lt;/property-string&gt;</code>	<ul style="list-style-type: none"> <li>name - the name of the property</li> <li>code - the function to run to generate a string value</li> </ul>
<code>&lt;property-set name='name' delimiter='delim'&gt;[code]&lt;/property-set&gt;</code>	<p>name - the name of the property  delimiter - the delimiter to use in parsing the data  code - the function to run to generate the data to be parsed into a value set</p> <p>For example, suppose there exists an input field data that looks like this: <code>ibm,sas,oracle</code>. This would yield the following property set: <code>&lt;property-set name='mySet' delimiter=', '&gt;\$data&lt;/property-set&gt;</code></p>

Suppose you had some employee information streaming into the model.

```
<event>
  <value name='map'>name:[employee name];
    position:[employee position]
  </value>
  <value name='developerInfo'>
    <![CDATA[<info>this is developer info</info>]]>
  </value>
  <value name='managerInfo'>
    <![CDATA[<info>this is manager info</info>]]>
  </value>
</event>
```

You can create a `property-map` to store employee data and then examine the position field to create a `property-xml` containing the appropriate data. If the employee is a developer, the XML is from `developerInfo`. Otherwise, it uses `managerInfo`. Your function-context would look like this, where the functions are coded using the `if`, `equals`, `mapValue`, and `xpath` support functions:

```
<function-context>
  <properties>
    <property-map name='myMap' outer=';'
inner=':'>$map</property-map>
    <property-xml
name='myXml'>if(equals(mapValue(#myMap,'position'),'developer'),
$developerInfo,$managerInfo)
    </property-xml>
  </properties>
  <functions>
    <function name='employee'>mapValue(#myMap,'name')</function>
    <function name='info'>xpath(#myXml,'text()')</function>
  </functions>
</function-context>
```

Streaming in the following event:

```
<event>
<value name='map'>
name:curly;position:developer
</value>
  <value name='developerInfo'>
    <![CDATA[<info>this is developer info</info>]]>
  </value> <value name='managerInfo'>
    <![CDATA[<info>this is manager info</info>]]>
  </value>
```

```

        </value>
    </event>
    <event>
        <value name='map'>name:moe;position:manager</value>
        <value name='developerInfo'><![CDATA[<info>this is developer
info</info>]]></value>
        <value name='managerInfo'><![CDATA[<info>this is manager
info</info>]]></value>
    </event>

```

Yields the following result:

```

<event opcode='insert' window='project/query/transform'>
    <value name='employee'>curly</value>
    <value name='id'>fd26bf36-3d65-4d17-8dc6-317409bbf5b6</value>
    <value name='info'>this is developer info</value>
</event>
<event opcode='insert' window='project/query/transform'>
    <value name='employee'>moe</value>
    <value name='id'>84c56bb7-9f3c-4cb8-93a5-8dc2f75d353b</value>
    <value name='info'>this is manager info</value>
</event>

```

## 3.12 Using Geofence Windows

### 3.12.1 Overview to Geofence Windows

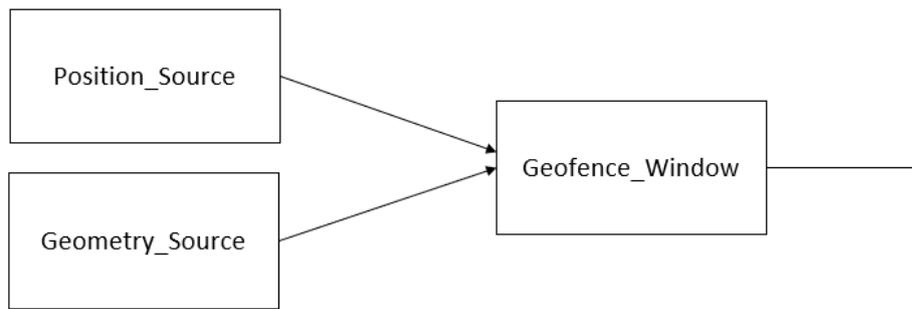
A geofence is a virtual perimeter for a real-world geographic area. You can dynamically generate a geofence as a radius around a specific location or create one as a set of specific boundaries. The Geofence window determines whether the location of an event stream is inside or close to an area of interest. You can augment an event with location details.

Geofence windows require two input windows: one to inject streaming events and another to inject geofence areas and locations. By default, you connect the window that injects events to the Geofence window with the first edge that you specify in the project. You connect the window that injects geofence areas and locations to the Geofence window with the second edge.

```

<edges>
    <edge source='position_source' target='geofence_window' />
    <edge source='geometry_source' target='geofence_window' />
</edges>

```



Thus, the Geofence window behaves like an outer Join window or a lookup operation. The events are on the streaming side and the geofence areas and locations are on the dimension side.

Alternatively, you can explicitly assign a role to the edges that connect input windows to Geofence Windows:

position or geometry. For example:

```
<edges>
  <edge source='geometry_source' target='geofence_window'
role='geometry' />
  <edge source='position_source' target='geofence_window'
role='position' />
</edges>
```

The Geofence window is designed to support Cartesian or geographic coordinate types. The only requirement is that all coordinates must be consistent and must refer to the same space or projection. For geographic coordinates, the coordinates must be specified in the (X,Y) Cartesian order (longitude, latitude). All distances are defined and calculated in meters.

You can restrict the geofence lookup to selected geometries depending on input position metadata. For example, if your project receives events that identify the position of multiple automobiles, you can choose to look up only geometry geofences relevant to a specific subset. You do this using the `join-key-fieldname` attributes of the `geometry` and `position` inputs. Attribute values are compared and evaluated before a geofence lookup is performed.

For information about the XML elements associated with Geofence windows, see “window-geofence” and “XML Language Elements Relevant to Geofence Windows”.

## 3.12.2 Geometries

### Overview

Areas and locations of interest are defined as geometries. The Geofence window supports the following geometries: polygons, polylines, and circles. Geometries are published as events, one event per geometry. Use the `geotype-fieldname` attribute of the `geometry` element to specify what type of geometry to include. When this attribute is not specified, the Geofence window determines the type automatically, based on the data characteristics. When the data does not match the type specified (for example, data specifies a non-closed ring but `geotype-fieldname='polygon'`), the geometry is rejected.

The Geofence window supports Insert, Update, and Delete opcodes, which can dynamically update the geometries. Each Geofence window instance can implement polygon geometries,

polyline geometries, or circle geometries, and it can perform geofence analysis on all types simultaneously.

Remember that the Geofence window behaves like a lookup join. You must build its output schema using all or a subset of the fields that come from the geometries input window. For more information about the output schema, see “Output Schema”.

### Polygons

A polygon is a plane shape representing an area of interest. The Geofence window supports polygons, multi- polygons, and polygons with holes or multiple rings.

Define a polygon as a list of position coordinates that represent the polygon's ring(s). A ring is a closed list of position coordinates. To be considered closed, the last point of the ring list must be the same as the first one. For example, a ring that is geometrically defined with four points, like a square, must declare five position coordinates, the last being the same as the first.

The input polygon window schema must have at least the following two fields:

- A single key field of type Int64 or String. This field defines the ID of the polygon.
- A data field of type array(dbl).

You can also specify an optional radius field of type Double. This field represents a proximity distance around the polygon. When this field is not specified, the default distance that is defined by the parameter radius is used. This value is used only when the property proximity-analysis is set to true.

When the polygon is not closed and does not contains any closed ring, it is considered a polyline.

For example, the following string represents a polygon made of four points. Notice that the fifth pair of numbers is identical to the first.

```
5.281 9.455 3.607 7.112 6.268 6.181 8.414 7.705 5.281 9.455
```

For polygons with multiple rings, the first ring defined must be the exterior ring and any others must be interior rings or holes.

---

### Note

Overlapping holes for a polygon's definition are not supported.

---

The schema can also have an optional description field. All other fields are ignored.

When working with polygons, the Geofence window analyzes each event position coming from the streaming window and returns the polygon in which this position is inside. If there are multiple matching geometries (in cases of overlapping polygons) and if the option `output-multiple-results` is set to true, then multiple events are produced (one per geometry).

In addition, when the property proximity-analysis is set to true, the Geofence window returns the ID of the polygon that is within the distance defined by the radius property value. Distance is measured from the position to the closest outer ring segment (holes are ignored).

When a polygon is detected, the output `geodistance-fieldname` returns the following:

- When the `polygon-compute-distance-to` property is set to `segment`: the exact distance from the position to the polygon. This value is negative when the position is within the polygon and it is positive when it is outside. When `proximity-analysis` is set to `true`, a polygon crossing can then easily be detected by a sign change using a Pattern window.
- When the `polygon-compute-distance-to` property is set to `centroid`: the exact distance from the position to the centroid of the polygon.

## Polylines

A polyline is a shape that represents a border or a trip wire. Define a polyline as a list of coordinates that represent the polyline's segment or segments. The sequence of the points in the segment definition defines the polyline's vector direction and its left and right side.

When working with polylines, a Geofence window analyzes each event position that comes from the positions Source window. It returns the ID of the polyline that is within the distance defined by the radius property value. Distance is measured from the position to the closest segment.

When a polyline is detected, the output `geodistance-fieldname` returns the exact distance from the event position to the polyline. This value is negative whenever the position is on the left side of the polyline. It is positive whenever it is on the right side. A trip wire crossing can then easily be detected by a sign change using a Pattern window.

Consider the following diagram. The polyline is depicted as a bold black line. The dashed line around the polyline is the area within the distance of the radius. A, B, C, and D are incoming event positions.

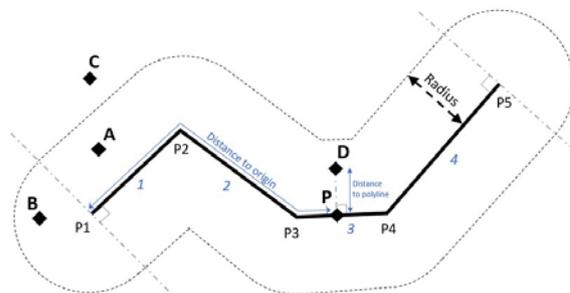


Figure 3-4 Polyline Relative to Several Event Positions

The distance between event position A and the polyline is less than the radius. For that event, the Geofence window returns the polyline ID.

The event position B is close to the polyline when `strict-projection='false'`. In that case, the polyline ID is returned. B is out of the polyline proximity when `strict-projection='true'`. The polyline ID is not returned.

The event position C is out of the polyline proximity. The polyline ID is not returned.

With regard to event position D:

- P is the projected point of position D. The coordinates of P are returned by the fields specified by the projection-fieldname parameter.
- The distance between D and P is returned by the field specified by the geodistance-fieldname parameter.
- P to P3 to P2 to P1 is the distance to origin. That distance is returned by the field specified by the distance-to-origin-fieldname parameter.
- The segment number is 3, which is returned by the field specified by the segmentnumber-fieldname parameter.

For more information about the projection-fieldname, geodistance-fieldname, distance-to-origin-fieldname, and segmentnumber-fieldname parameters, see “output”

### Circles

A circle encompasses the position of a location of interest. Define a circle with three values:

- two coordinates, X and Y (longitude and latitude), that represent the center of the circle
- a radius distance around the center

The following three fields of the input circle geometry window schema are required:

- A single key field of type Int64 or String. This field defines the ID of the circle.
- One of the following:
  - Two coordinate fields of type Double that contain the X and Y coordinates of the circle center
  - A data field of type Array(dbl), string, or rstring with coordinates defined as numbers in the X, Y order, separated by spaces

The schema can also contain the following optional fields:

- A radius field (double) that represents a circular area around the center point position. If you do not specify this field, the default distance defined by the parameter radius is used.
- A description field.

All other fields are ignored.

When working with circles, the Geofence window analyzes each event position that comes from the streaming window and returns the circle ID that matches the following criteria:

- If the position lookup distance is set to 0, then the position behaves like a simple point. It is either in or out of the circle. If it is in the circle, there is a match.
- Similarly, for circle geometries, if the circle radius is set to 0, then the circle behaves like a bare point. This point must be in the position lookup distance area to return a match.
- For any other values of the position lookup distance and the circle radius, the position and the circle's center must be within each other's distance. Otherwise, no match is returned. The position is within the circle and the distance between the circle's center and the position is lower than the lookup distance.
- If both the position lookup distance and the circle radius are equal to 0, then they must be the exact same point to have a match.

The position lookup distance is either defined by an additional event input field value or by the parameter lookupdistance.

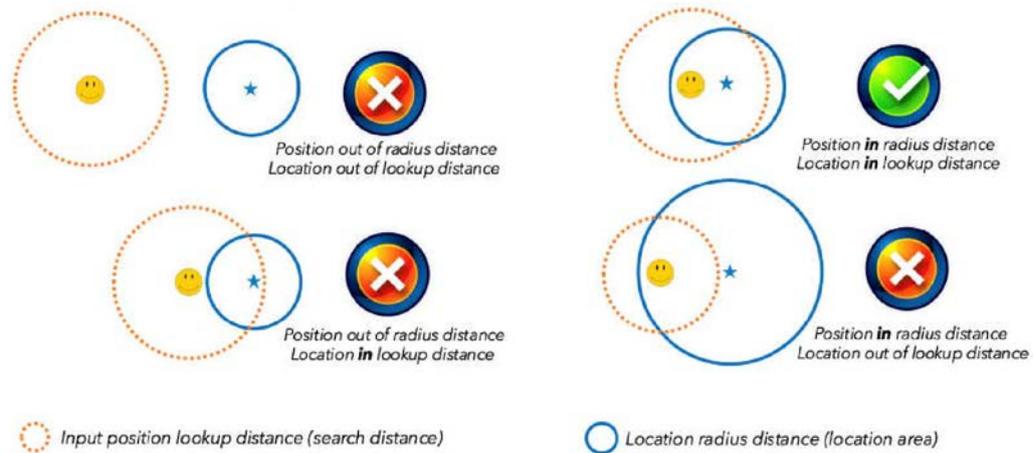


Figure 3-5 Circles Geometry Lookup Logic

If there are multiple matching geometries in the lookup distance and if the option output-multiple-results is set to true, then multiple events are produced (one per geometry).

### 3.12.3 Positions Window Schema

The positions input window's schema can contain any type and number of fields that are propagated to the output schema. The following schema fields are used by the Geofence window:

- Two mandatory coordinate fields of type Double that contain the X and Y or longitude and latitude coordinates.
- An optional field of type Double that contains the lookup distance for the current position event. This field is used only by Geofence windows that are configured to use circles geometries.

All other fields are ignored and propagated to the output schema.

### 3.12.4 Output Schema

The Geofence window behaves like a lookup join, so its output schema is automatically defined by the following fields in the following order:

- All fields that come from the input position window in their respective order.
- A mandatory field of type int64 or string that receives the ID of the geometry. If no geometries are found, then the value of this field is null in the produced event. This field is defined by the parameter geoid-fieldname.

Output schema are appended with the following fields:

- A field that receives the description of the geometry when it exists in the geometry window schema. This field's presence and name are defined by the parameter `geodesc-fieldname`.
- A field (double) that receives the distance from the position to the geometry. The value of this field is one of the following:
  - the distance to the center of the circle
  - the distance to the closest segment of the polyline
  - the distance to the centroid or closest segment of the polygon

This field's presence and name are defined by the parameter `geodistance-fieldname`.

- If `output-multiple-results` is set to true, then there is a mandatory key field of type Int64 that receives the event number of the matching geometry. This field's name is defined by the parameter `eventnumber-fieldname`.
- A field of type string that receives geometry type. This field's presence and name are defined by the parameter `geotype-fieldname`.
- Two fields (double) that receive the coordinates of the projected point on the closest segment of the polyline. These fields' values are null for geometry types other than polylines. These fields' presence and names are defined by the parameter `projection-fieldname`. These fields' names are prepended with `_X` and `_Y` respectively.
- A field (int32) that receives the segment number of the projected point on the closest segment of the polyline. Its value is null for geometry types other than polylines. This field's presence and name are defined by the parameter `segmentnumber-fieldname`.
- A field (array(double)) that receives the segments' length of the polyline. The sizes are in meters for geographic coordinates and in units for Cartesian coordinates. Its value is null for geometry types other than polylines. This field's presence and name are defined by the parameter `segmentsizes-fieldname`.
- A field (double) that receives the distance from the projected point to the first point of the polyline along the polyline. Its value is null for geometry types other than polylines. This field's presence and name are defined by the parameter `distance-to-origin-fieldname`.
- A list of fields that are propagated from the geometry input schema. These field's presence and names are defined by the parameter `include-geo-fields`.

### 3.12.5 Mesh Index

For fast and low latency lookup processing, the Geofence window implements an optimized mesh index algorithm that uses a spatial data structure. This structure subdivides space into buckets of grid shapes called cells. This structure is independent of the coordinate system in use, enabling the seamless use of any type of Cartesian, geographic, or projection coordinates space.

The mesh algorithm uses a parameter (called a mesh factor) that defines the scale of the space subdivision. This mesh factor is an integer within the [-5, 5] range that represents a power of 10 of the coordinate units in use.

For example, the default factor of 0 generates 1 subdivision per coordinate unit. A factor of 1 generates a subdivision per 10 units. A factor of -1 generates 10 subdivisions per unit. This factor can be set for both X and Y axes or independently for each axis.

Consider the following set of coordinates that represents a square polygon. Note the repeated first point at the end, closing the polygon:

```
[(1001.12,9500.12) (1001.12,9510.12) (1010.12,9510.12) (1010.2,9500.12)  
(1001.12,9500.12)]
```

- With a mesh factor of 1, the Geofence window divides the coordinates by 101. This results in [(100,950) (100,951) (101,950) (101,951)] and creates four mesh cells for this geometry:  $(101-100+1)*(951-950+1) = 4$ .
- With a factor of 2, it creates  $(10-10+1)*(95-95+1) = 1$  mesh cell.
- When the mesh factor is set to -1, the window creates 9,191 mesh cells. This results in an oversized mesh:  $(10101-10011+1)*(95101-95001+1)=91*101 = 9191$ .

For optimal performance, you must adapt the mesh factor to the spatial coverage and to the number of loaded geometries. When the number of mesh cells per geometry is too high, the ingestion of geometries is slowed and an oversized index is generated. When the number of mesh cells per geometries is too low, the lookup process is slowed, which affects stream performance and latency.

A dedicated parameter `max-meshcells-per-geometry` sets the maximum allowed mesh cells created per geometries to avoid creating an oversized mesh, which would generate useless intensive calculations. When a geometry exceeds this limit, it is rejected. When you intend to cover a very large area, consider setting a higher mesh factor or setting a higher maximum number of mesh cells per geometry.

The Geofence window provides an internal algorithm that automatically computes and sets an appropriate mesh factor by analyzing the first 1,000 geometries ingested. It is automatically active by default. To set the mesh factors manually, set the parameter `autosize-mesh` to false.

### 3.12.6 Example

```
<window-geofence name="geofence_win" index="pi_HASH">  
<geofence coordinate-type="geographic"  
log-invalid-geometry="true"  
output-multiple-results="true"  
output-sorted-results="false"  
max-meshcells-per-geometry="100"  
autosize-mesh="true"  
>  
<geometry data-fieldname="GEO_data"  
desc-fieldname="GEO_desc"  
x-fieldname="GEO_x"  
y-fieldname="GEO_y"  
radius-fieldname="GEO_radius"  
radius="0"  
data-separator=" "  
>  
</position
```

```

x-fieldname="GPS_longitude"
y-fieldname="GPS_latitude"
lookupdistance="110"
/>
<output geoid-fieldname="GEO_id_out"
geodesc-fieldname="GEO_desc_out"
eventnumber-fieldname="event_nb"
geodistance-fieldname="GEO_dist"
/>
</window-geofence>

```

## 3.13 Using Join Windows

### 3.13.1 Overview to Join Windows

A Join window receives events from a left input window and a right input window. It produces a single output stream of joined events. Joined events are created according to a user-specified join type and user-defined join conditions.

The join order is determined at the edge level of the project. The left window is the first window that is defined as a connecting edge to the Join window. The second window that is defined as a connecting edge is the right window. Using XML, you can explicitly assign a role to the edge to define which window connects to the Join window:

**left** or **right**. For example:

```

<edges>
  <edge source='w_source1' target='w_join' role='left'/>
  <edge source='w_source2' target='w_join' role='right'/>
</edges>

```

Because an engine is based on primary keys and supports Inserts, Updates, and Deletes, there are some restrictions placed on the types of joins that can be used. The four join types include:

left-outer join

A left-outer join produces joined output events for every event that arrives from the left window. Joined events are created even when there are no matching events from the right window.

right-outer join

A right-outer join produces joined output events for every event that arrives from the right window. Joined events are created even when there are no matching events from the left window.

inner join

An inner join creates joined events only when there is one or more matching events on the side opposite of the input event.

full outer join

A full outer join creates joined events for every event that arrives from the left window or right window of the joins. Output is always produced.

User-defined join conditions specify what key fields from the left and right input windows are used to generate joined events. Join conditions are an n-tuple of equality expressions. Each expression involves one field from the left window and one field from the right. For example: (left.f1 == right.f10), (left.f2 == right.f7), ... (left.field10 == right.field1) .

To calculate the join non-key fields when new input events arrive, a Join window takes one of the following:

- A join selection string that is a one-to-one mapping of input fields to join fields
- Field calculation expressions
- Field calculation functions

For information about the XML elements associated with a Join window, see ("**window-join**") and ("**XML Language Elements Relevant to Join Windows**")

### 3.13.2 Understanding Streaming Joins

#### Overview to Streaming Joins

The following definition is essential to understanding streaming joins: an X-to-Y join is a join where the following holds:

- A single event from the left window can effect at most X events in the Join window
- A single event in the right window can effect at most Y events in the Join window.

Given a left window, a right window, and a set of join conditions, a streaming join can be classified into one of three different categories.

Join Category	Description
one -to-one joins	An event on either side of the join can match at most one event from the other side of the join. This type of join always involves two dimension tables.
one-to-many joins (or many-to-one joins)	An event that arrives on one side of the join can match many rows of the join. An event that arrives on the other side of the join can match at most one row of the join. In order for a join to be classified as a one-to-many (or many-to-one) join, the following must be true: <ul style="list-style-type: none"> <li>• a change to one side of the join can affect many rows</li> <li>• a change to the other side can affect at most one row</li> </ul>
many-to-many joins	A single event that arrives on either side of the join can match more than one event on the other side of the join.

In a streaming context, every window has a primary key that enables the insertion, deletion, and updating of events. The key fields of a Join window are derived from its input windows and are never specified by the user. When an event arrives on either side, you must be able to compute how the join changes, given the nature of the arriving data (Insert, Update, or Delete).

3.13 Using Join Windows

Edge Streaming Analytics determines the keys for a Join window based on the join type and join conditions that are specified by the user. Edge Streaming Analytics follows a set of axioms to maintain consistency for the most common join cases:

Join Category	Key Derivation Axiom
one-to-one joins	For a left-outer join, the keys of the left input window are passed through to the Join window. The keys of the right window are never passed to the Join window. This is because a Join window event is produced when a left window event arrives, even if there is no matching right window event.  For a right-outer join, the keys of the right input window are passed through to the Join window. The keys of the left window are never passed to the Join window. This is because a Join window event is produced when a right window event arrives, even if there is no matching left window event.
one-to-many joins (or many-to-one joins)	For a one-to-many and many-to-one join, the keys of the "many side" are used as the Join window key. In order to distinguish between Join window events, the many side is the side without all its keys specified in the join conditions.
many-to-many joins	For a many-to-many join, the union of the keys of the left and right input windows are used as the key fields for the Join window. Because a left window event might match multiple right window events, the keys from the right window are needed to distinguish between Join window events. Likewise, because a right window event might match multiple left window events, the keys from the left window are needed to distinguish between Join window events.

Left and right input windows are classified as dimension windows or fact windows. Dimension windows are those whose entire set of key fields participate in the join conditions. Fact windows are those that have at least one key field that does not participate in the join conditions. Input windows are not classified as dimension or fact windows before sending events to a join window.

The following table summarizes the allowed join sub-types and key derivation based on the axioms and the specified join conditions.

Join Category	Left Window	Right Window	Allowed Type	Key Derivation	Streaming Window
one -to -one	dimension	dimension	left outer	Join keys are keys of left window.	left
			right outer	Join keys are keys of right window.	right
			full outer	Join keys are keys of left window (arbitrary choice).	left
			inner	Join keys are keys of left window (arbitrary choice).	left
one-to-many	fact	dimension	left-outer	Join keys are keys of left window (right window is lookup).	left
			inner	Join keys are keys of left window (right window is lookup).	left

Join Category	Left Window	Right Window	Allowed Type	Key Derivation	Streaming Window
many-to-one	dimension	fact	right-outer	Join keys are keys of right window (left window is lookup).	right
			inner	Join keys are keys of right window (left window is lookup).	right
many-to-many	fact	fact	inner	Join keys are the full set of keys from the left and right windows.	none

**Note**

When all the keys of a window are used in a join condition, adding additional non-key fields on the side of the condition is not honored. For example, suppose that the set of left-hand fields that participate in a join condition contain all of the keys of the left window. Any non-key fields in that set are ignored.

**Using Secondary Indices**

For allowed one-to-many and many-to-one joins, a change to the fact table enables immediate lookup of the matching record in the dimension table through its primary index. All key values of the dimension table are mapped in the join conditions. However, a change to the dimension table does not include a single primary key for a matching record in the fact table. This illustrates the many-to-one nature of the join. By default, matching records in the fact table are sought through a table scan.

For very limited changes to the dimension table there is no additional secondary index maintenance, so the join processing can be optimized. Here, the dimension table is a static lookup table that can be pre-loaded. All subsequent changes happen on the fact table.

When a large number of changes are possible to the dimension table, it is suggested to enable a secondary index on the join. Automatic secondary index generation is enabled by specifying a join parameter when you construct a new Join window. This causes a secondary index to be generated and maintained automatically when the join type involves a dimension table.

There is a slight performance penalty when you run with secondary indices turned on. The index needs to be maintained with every update to the fact table. But generating a secondary index has the advantage of eliminating all table scans when changes are made to the dimension table. Secondary index maintenance is insignificant compared with elimination of table scans. With large tables, you can achieve time savings of two to three orders of magnitude by using secondary indices.

For many-to-many joins, it is recommended to enable secondary indices.

**Using Regeneration versus No Regeneration**

The default join behavior is to always regenerate the appropriate rows of a Join window when a change is made to either side of the joins. The classic example of this is a left outer

join: the right window is the lookup window, and the left table is the fact (streaming) window. The lookup side of the join is usually pre-populated, and as events stream through the left window, they are matched and the joined events output. Typically, this is a one-to-one relation for the streaming side of the join: one event in, one combined event out. Sometimes a change is made on the dimension side. This change can be in the form of an update to an event, a deletion of an event, or an insertion of a new event. The default behavior is to issue a change set of events that keeps the join consistent.

In regeneration mode, the behavior of a left outer join on a change to the right window (lookup side) is as follows:

- **Insert:** find all existing fact events that match the new event. If any are found, issue an update for each of these events. They would have used nulls for fields of the lookup side when they were previously processed
- **Delete:** find fact events that match the event to be deleted. If any are found, issue an update for each of these events. They would have used matching field values for the lookup event, and now they need to use nulls as the lookup event is removed.
- **Update:** Behaves like a delete of the old event followed by an insert of the new event. Any of the non-key fields of the lookup side that map to keys of the streaming side are taken into account. It is determined whether any of these fields changed value.

With no-regeneration mode, when there is a left outer join on a change to the right window (lookup side), changes to the dimension (lookup) table affect only new fact events. All previous fact events that have been processed by the join are not regenerated. This frequently occurs when a new dimension window is periodically flushed and re-loaded.

The Join window has a `no-regenerates` flag that is false by default. This gives the join full-relational join semantics. Setting this flag to true for your Join window enables the `no-regenerates` semantics. Setting the

flag to true is permitted for any of the left or right outer joins, along with one-to-many, many-to-one, and one-to-one inner joins. When a Join window is running in `no-regenerates` mode, it optimizes memory usage by

omitting the reference-counted copy of the fact window's index that is normally maintained in the Join window.

### 3.13.3 Creating Empty Index Joins

Suppose there is a join of type `LEFT_OUTER` or `RIGHT_OUTER`. The index type is set to `pi_EMPTY`, rendering a stateless Join window. The `no-regenerates` flag is set to `TRUE`. This is as lightweight a join as possible. The

only retained data in the join is a local reference-counted copy of the dimensions table data. This copy is used to perform lookups as the fact data flows into, and then out of, the join.

On a Join window, you cannot specify insert-only for left and right inputs independently. Specifying insert-only for both sides of the join by setting the Join window to "insert only" is

too restrictive. This would not permit changes to the lookup, or non-streaming side of the join. You must follow these rules to ensure expected results.

- A many-to-many join cannot have an empty index.
- The streaming side of a join, as specified in the join classification table, can receive only inserts

### 3.13.4 Examples of Join Windows

The following example shows a left outer join. The left window processes fact events and the right window processes dimension events.

```
left input schema: "ID*:int32,symbol:string,price:double,quantity:int32,
  traderID:int32"
```

```
right input schema: "tID*:int32,name:string"
```

Your code for the Join window would look like this:

```
<window-join name='myJoinWindow' index='pi_RBTREE'>
  <join type='leftouter'>
    <conditions>
      <fields left='traderID' right='tID' />
    </conditions>
  </join>
  <output>
    <field-selection name='sym' source='l_symbol' />
    <field-selection name='price' source='l_price' />
    <field-selection name='tID' source='l_traderID' />
    <field-selection name='traderName' source='r_name' />
  </output>
</window-join>
```

Note the following:

- Join conditions take the following form. They specify what fields from the left and right events are used to generate matches.

```
"l_fieldname=r_fieldname, ...,l_fieldname=r_fieldname"
```

- Join selection takes the following form. It specifies the list of non-key fields that are included in the events generated by the Join window.

```
"{l|r}_fieldname, ...{l|r}_fieldname"
```

- Field signatures take the following form. They specify the names and types of the non-key fields of the output events. The types can be inferred from the fields specified in the join selection. However, when using expressions or user-written functions (in C++), the type specification cannot be inferred, so it is required:

```
"fieldname:fieldtype, ..., fieldname:fieldtype
```

When you use non-key field calculation expressions, your code looks like this:

```
<window-join name='myJoinWindow' index='pi_RBTREE'>
<join type='leftouter'>
  <conditions>
    <fields left='traderID' right='tID' />
  </conditions>
</join>
  <output>
    <field-expr name='sym' type='string'>l_symbol</field-expr>
    <field-expr name='price' type='double'>l_price</field-expr>
    <field-expr name='tID' type='int32'>l_traderID</field-expr>
    <field-expr name='traderName' type='string'>r_name</field-expr>
  </output>
</window-join>
```

This shows one-to-one mapping of input fields to join non-key fields. You can use calculation expressions and functions to generate the non-key join fields using arbitrarily complex combinations of the input fields. For XML and C++ code examples of full projects, go to **\$DFESP\_HOME/examples** on Linux systems. Go to **%DFESP\_HOME%\examples** on Windows systems.

## 3.14 Using Model Reader Windows

### 3.14.1 Using Model Reader Windows

In most cases, *Model Reader windows* receive **(request events)** that include the location and type of an offline model. Offline models are specified, developed, trained, and stored separately from the Streaming Server. Model Reader windows publish a model event that contains the model to Score windows or to Model Supervisor windows.



For recommender systems, you can specify offline model property values within the Model Reader window itself. The window publishes a model event based on those values.

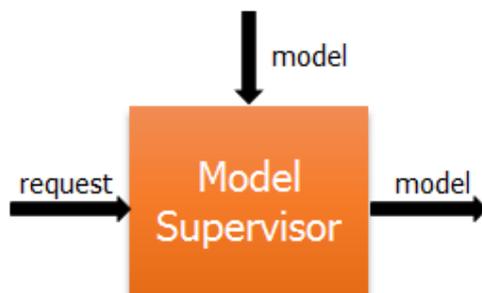


For more information, see [Edge Streaming Analytics: Using Streaming Analytics](#).

## 3.15 Using Model Supervisor Windows

### 3.15.1 Using Model Supervisor Windows

*Model Supervisor windows* manage the flow of model events. Through input (**request events**), you can control what model to deploy and when and where to deploy it. Model events are published to Score windows.



A Model Supervisor window can receive any number of model events. In a streaming analytics project, model events are typically sent by a Train window or a Model Reader window. After receiving a model event, a Model Supervisor window processes and publishes

events to other streaming analytics windows based on the Model Supervisor window's deployment mode and on user requests.

The `<window-model-supervisor>` element has three properties:

- `name=` specifies the name of the window
- `deployment-policy=` `-capacity=` specifies the number of current model events to keep. After the capacity is met, older model events are discarded. For more information, see [Edge Streaming Analytics: Using Streaming Analytics](#)

**immediate**

sends model events to any receiving window immediately after receiving them

**on-demand**

sends model events according to the requests specified at the command line

- `capacity=` specifies the number of current model events to keep. After the capacity is met, older model events are discarded.

For more information, see [Edge Streaming Analytics: Using Streaming Analytics](#).

## 3.16 Using Notification Windows

### 3.16.1 Overview to Notification Windows

Use a Notification windows to send notifications through email using the Simple Mail Transfer Protocol (SMTP), text using the Short Message Service (SMS), or multimedia messages using the Multimedia Messaging Service (MMS). These windows, like Functional windows, enable you to define a function context to transform incoming events before processing them for possible notifications. Each of the different types of notification has its own configuration requirements. For example, an email requires that the configuration specify the event field that contains the 'send to' email address. SMS and MMS require phone numbers and phone provider gateway information.

You can format notifications as you want and include the event values within the message. To include event values, include the name of the field, preceded by a \$ character, in your message formatting:

```
<b>$broker</b> sold $quant1 shares of <b>$symbol</b>  
  
$tstamp1 for self for $$price1, then sold $quant2 shares for customer at $tstamp2.
```

Notification windows enable you to create any number of delivery channels to send notifications. You can specify functions to determine whether to send the notification. Given the potentially massive amounts of streaming data that could cause an avalanche of notifications, you can specify a throttle interval for each channel. If you set the interval to '1 hour', you send at most one notification from that channel to any recipient every hour.

Notification windows never generate events. Nevertheless, you can use the schema element to specify values for the function-context to generate. You can use these values to format notification messages.

For information about the XML elements associated with a Notification window, see (**"window-notification"**) and (**"XML Language Elements Relevant to Notification Windows"**).

To use delivery channels, you must specify a `smtp` element to provide information about an SMTP server:

```
<smtp host='host' user='user' password='password' port='port' (opt, default='25') />
```

Only the `host` attribute of the element is required, because many SMTP servers run on the default port and do not require authentication:

```
<smtp host='mailhost.fyi.sas.com' />
```

However, it is a good practice to supply values for all the attributes of the `smtp` element:

```
<smtp host='smtp-server.ec.rr.com'  
user='esptest@ec.rr.com'  
password='esptest1' port='587' />
```

## 3.16.2 Notification Window Delivery Channels

### Overview to Notification Window Delivery Channels

The Notification window uses three types of delivery channel:

- `email` sends a multipart email message that contains text, HTML, and images to a specified email address
- `sms` sends an SMS text message that contains text to an address in the format **phoneNumber@gateway**
- `mms` sends a MMS message that contains text and images to an address in the format **phoneNumber@gateway**

### Using the Email Delivery Channel

Here is XML code to use the email delivery channel:

```
<email throttle-interval='' test='true | false'  
<deliver>[code]</deliver>  
<email-info>  
<sender>[code]</sender>  
<recipients>[code]</recipients>  
<subject>[code]</subject>  
<from>[code]</from>  
<to>[code]</to>  
</email-info>  
<email-contents>  
<text-content name=''>...</text-content>
```

```
<html-content name=''>...</html-content><image-content name=''>...</image-content>
...
</email-contents>
</email>
```

The *email* element contains the following attributes:

- *throttle-interval* specifies a time period in which at most one notification is sent to a recipient
- *test* is a Boolean attribute that specifies whether to run in test mode. When running in test mode, the notification is not sent but written to the console. This can be useful when drafting notification messages.

The *deliver* element is optional. It contains a function to run in order to determine whether the notification should be sent.

The *email-info* element contains functions or hardcoded values that represent the data to be used to send an email notification. It contains the following elements:

- the *sender* email address
- the *recipients* to whom the email message is sent. Multiple recipients can be delimited by the ',' character or the ';' character.
- the *subject* of the email
- the *from* text of the email message
- the *to* text of the email
- The *email-contents* element, which contains the following elements:
  - the *text-content* element encloses the plain text content of the message
  - the *html-content* element encloses the HTML content of the message
  - the *image-content* element encloses a URI to image data. This URI can point to an external entity as follows: **file:///directory/image\_filename** or **http://website/image\_filename**. It can also point to image data directly embedded from an event specified in the schema of the Notification window or the immediately preceding window as follows: **field://fieldname**.

These elements can be interspersed in any way you want. The content of each element is included in the message in the order in which it appears. Any image data is retrieved and base64 encoded before being inserted into the message.

### Using the SMS Delivery Channel

Here is XML code to use the sms delivery channel:

```
<sms throttle-interval='' test='true | false'>
  <deliver>[code]</deliver>
  <sms-info> <sender>[code]</sender>
  <subject>[code]</subject>
  <from>[code]</from>
  <gateway>[code]</gateway>
```

```
<phone>[code]</phone>
  </sms-info> <sms-contents>
<text-content name=''>...</text-content>
  </sms-contents>
</sms>
```

The *sms* element contains the following attributes:

- *throttle-interval* specifies a time period in which at most one notification is sent a recipient.
- *test* is a Boolean attribute that specifies whether to run in test mode. When running in test mode, the notification is not sent but written to the console. This can be useful when drafting notification messages.

The *deliver* element is optional. It contains a function to run in order to determine whether the notification should be sent.

The *sms-info* element contains functions or hardcoded values that represent the data to be used to send an SMS notification. It contains the following elements:

- the *sender* SMS address.
- the *subject* of the message.
- the *from* text of the message.
- the *gateway* element specifies the recipient's provider's SMS gateway. For example, AT&T is *txt.att.net*. Sprint is *messaging.sprintpcs.com*.
- the *sms-contents* element contains the body of the message to be sent. It contains the following element:
  - the *text-content* element encloses the plain text content of the message.

### Using the MMS Delivery Channel

Here is XML code to use the MMS delivery channel:

```
<mms throttle-interval='' test='true | false'>
  <deliver>[code]</deliver>
<mms-info>
  <sender>[code]</sender>
  <subject>[code]</subject>
  <gateway>[code]</gateway>
  <phone>[code]</phone>
</mms-info>
<mms-contents>
  <text-content name=''>...</text-content>
  <image-content name=''>...</image-content>
  ...
</mms-contents>
```

</mms>

The mms element contains the following attributes:

- *throttle-interval* specifies a time period in which at most one notification is sent to a recipient.
- *test* is a Boolean attribute that specifies whether to run in test mode. When running in test mode, the notification is not sent but written to the console. This can be useful when drafting notification messages

The *deliver* element is optional. It contains a function to run in order to determine whether the notification should be sent.

The *mms-info* element contains functions or hardcoded values that represent the data to be used to send an MMS notification. It contains the following elements:

- the *sender* MMS address.
- the *subject* of the message.
- the *gateway* element specifies the recipient's provider's SMS gateway. For example, AT&T is *txt.att.net*. Sprint is *messaging.sprintpcs.com*.
- the recipient *phone* number.
- the *mms-contents* element contains the body of the message to be sent. It contains the following elements:
  - the *text-content* element encloses the plain text content of the message.
  - the *image-content* element encloses a URI to image data. This URI can point to an external entity as follows: **file:///directory/image\_filename** or **http://website/image\_filename**. It can also point to image data directly embedded from an event specified in the schema of the Notification window or the immediately preceding window as follows: **field://fieldname**.

These elements can be interspersed in any way you want. The content of each element is included in the message in the order it appears. Any image data is retrieved and *base64* encoded before being inserted into the message.

### 3.16.3 Using the Function-Context Element

The *function-context* element enables you to define functions to manipulate event data. You can use regular expressions, XML and XPath, or JSON to transform data from complex input information into more usable data.

Here is XML code that uses the function-context element:

```
<function-context>
  <expressions>
    <expression name=''>[Regular Expression]</expression>
    ...
  </expressions>
</properties>
```

```
<property-map name='' outer='' inner=''>[code]</property-map>
<property-xml name=''>[code]</property-xml>
<property-json name=''>[code]</property-json>
<property-string name=''>[code]</property-string>
<property-list name='' delimiter=''>[code]</property-list>
<property-set name='' delimiter=''>[code]</property-set>
...
</properties>
<functions>
<function name=''>[code]</function>
...
</functions>
</function-context>
```

You can use two types of functions in the function-context element:

- general functions (for example, *abs*, *ifNext*, and so on)
- functions that are specific to event stream processing (for example, *eventNumber*)

You can reference event fields in either the input event or the output event using the \$ notation (for example, *\$(name\_of\_field)*).

Suppose that you have a name field in the input event and you want to generate an occupation field in the output event based on the value of name. In this case, you could use the following function:

```
<function name='occupation'>
ifNext
(
equals($name,'larry'),'plumber',
equals($name,'moe'),'electrician',
equals($name,'curly'),'carpenter'
)
</function>
```

Now suppose that you want to add an hourlyWage to the output event that depends on occupation:

```
<function name='hourlyWage'>
ifNext
(
equals($occupation,'plumber'),85.0,
equals($occupation,'electrician'),110.0,
```

```

equals($occupation, 'carpenter'), 60.0
)
</function>

```

**Note**

**Sequence is important when you define functions in the function-context element. When a function references an output event field, that field needs to be computed before the referring field**

Use POSIX regular expressions in your code. Several functions are available to deal with regular expressions. Because regular expressions must be compiled before they can be used, use the expressions element to specify that expressions are compiled a single time when the function context is created. Then, the expression can be referenced from within functions using the following notation:

```

#[name_of_expression]
<function name='myData'>rgx(#myExpression,$inputField,1)
</function>

```

For example, suppose you receive a data field that contains a URI and you want to extract the protocol from it. When you use the following function, the regular expression is compiled each time that the function runs:

```

<function name='protocol'>rgx('(.*):',$uri,1)
</function>

```

If you use the following code, the expression is compiled a single time and used each time that the function runs:

```

<expressions>
<expression name='getProtocol'>(.*):
</expression>
</expressions>

<function name='protocol'>rgx(#getProtocol,$uri,1)
</function>

```

Reference properties from within functions using the # notation: #[name\_of\_property]. The properties element is a container for the following elements:

Element	Description
property-map	executes the function to generate a map of name-value pairs to be used for value lookups by name
property-list	executes the function to generate a list of strings to be used for indexed access
property-set	executes the function to generate a set of strings to be used for value lookups
property-xml	executes the function to generate an XML object that can be used for XPath queries
property-json	executes the function to generate a JSON object that can be used for JSON lookups
property-string	executes the function to generate a string for general use in functions

Each property is generated using functions. These functions can reference properties defined before them in the XML.

Suppose you had employee information streaming into the model.

```
<event>
<value name='map'>name:[employee name];position:[employee position]</value>
  <value name='developerInfo'><![CDATA[<info>this is developer info</info>]]></value>
  <value name='managerInfo'><![CDATA[<info>this is manager info</info>]]></value>
</event>
```

You can use the *property-map* element to store employee data and examine the *position* field of the event in order to create a *property-xml* that contains the appropriate data. When the employee is a developer, the

XML is created from *developerInfo*. Otherwise, it uses *managerInfo*. Specify the *function-context* element as follows:

```
<function-context>
<properties>
  <property-map name='myMap' outer=';' inner=':'>$map</property-map>
  <property-xml name='myXml'>
if(equals(mapValue(#myMap,'position'),'developer'),
  $developerInfo,$managerInfo)</property-xml>
</properties>
<functions>
<function name='employee'>mapValue(#myMap,'name')</function>
  <function name='info'>xpath(#myXml,'text()')</function>
</functions>
</function-context>
```

When you stream the following events:

```
<event>
<value name='map'>name:curly;position:developer</value>
<value name='developerInfo'><![CDATA[<info>this is developer info</info>]]></value>
<value name='managerInfo'><![CDATA[<info>this is manager info</info>]]></value>
</event>
<event>
<value name='map'>name:moe;position:manager</value>
<value name='developerInfo'><![CDATA[<info>this is developer info</info>]]></value>
<value name='managerInfo'><![CDATA[<info>this is manager info</info>]]></value>
</event>
```

The *function-context* yields the following:

```
<event opcode='insert' window='project/query/transform'>
```

```
<value name='employee'>curly</value>

<value name='id'>fd26bf36-3d65-4d17-8dc6-317409bbf5b6</value>

<value name='info'>this is developer info</value>

</event>

<event opcode='insert' window='project/query/transform'>

<value name='employee'>moe</value>

<value name='id'>84c56bb7-9f3c-4cb8-93a5-8dc2f75d353b</value>

<value name='info'>this is manager info</value>

</event>
```

For more information about how to define each property, see( **"XML Language Elements for Expressions and Functions"**.)

## 3.17 Using Object Tracking Windows

### 3.17.1 Overview to Object Tracking Windows

Use an Object Tracking window to perform multi-object tracking (MOT) in real time. MOT is the process of accurately estimating the identity and position of multiple objects over time using a model that is based on incoming observations. These observations are the output of an object detection model.

The challenges of multi-object tracking include scene clutter, target dynamics, intra-class and inter-class variation, measurement noise, and frame rate. The Object Tracking window addresses these challenges by coupling trackers with detectors in a paradigm called tracking-by-detection. Specifically, it uses an intersection- over-union (IOU) method of tracking-by-detection that is explained in Bochinski, Eiselein, and Sikora (2017).

For information about the XML elements associated with Object Tracking windows, see (**window-object-tracker**) and (**"XML Language Elements Relevant to Object-Tracking Windows"** .)

### 3.17.2 The Intersection Over Union (IOU) Method of Tracking- By-Detection

The IOU method tracks objects using only detection data. Using this method enables incremental and event- oriented tracking. It has produced good results on the ( **MOTChallenge**) and the( **UA-DETRAC benchmarks**.)

The IOU method implemented by the Object Tracking window provides the following functionality:

- It uses a velocity vector to predict the next position of the tracked object when it is missing in the next frame. The *velocity-vector-frames* parameter defines the number of frames used to calculate the velocity vector.
- After the first tentative match with existing tracks and detection occurs ( $\text{IOU} > \sigma\text{IOU}$ ), any detections that remain unassociated are matched as follows: detections that have an  $\text{IOU} > \sigma\text{IOU}_2$  with some unmatched tracks are matched with whatever unmatched track has the greater IOU.
- Remaining unmatched detections create a new track only when their IOU with existing tracks is less than the value of *iou-sigma-dup*. This avoids reassignments of double detections. Setting the *iou-sigma-dup* parameter to the default value of 0.0 disables this feature.
- Tracks begin with multiple lives. When a track is left unmatched with any detection on a frame, it loses a life. After several frames without matching a detection, it dies. The parameter *max-track-lives* sets the life duration of tracks without detection.

### 3.17.3 Input Schema for Object Tracking Windows

The Object Tracking window uses incoming detection data in order to perform multi-object tracking. It uses an input schema that consists of the following fields to propose detections:

- The detected objects count for the event.
- For each detected object, the following fields:
  - The label of the detected object.
  - The detection score.
  - The x coordinate of the bounding box center or the lower corner, respective to the origin of the coordinates. In object detection output, this often refers to the top left corner.
  - The y coordinate of the bounding box center or the lower corner, respective to the origin of the coordinates. In object detection output, this often refers to the top left corner.
  - The width of the bounding box or the x coordinate of the higher bounding box corner, respective to the origin of the coordinates. In object detection output, this often refers to the bottom right corner.
  - The height of the bounding box or the y coordinate of the higher bounding box corner, respective to the origin of the coordinates. In object detection output, this often refers to the bottom right corner.
- All other fields are propagated to the output event, including the event key fields.

For example:

```
frame_id*:int64,image:blob,objCount:int32,obj_0_x:double,obj_0_y:double,obj_0_h:doubl  
e,
```

3.17 Using Object Tracking Windows

```
obj_0_w:double,obj_0_score:double,obj_0_label:string,obj_1_x:double,obj_1_y:double,obj_1_h:double,
obj_1_w:double,obj_1_score:double,obj_1_label:string,obj_2_x:double,obj_2_y:double,obj_2_h:double,
obj_2_w:double,obj_2_score:double,obj_2_label:string,obj_3_x:double,obj_3_y:double,obj_3_h:double,
obj_3_w:double,obj_3_score:double,obj_3_label:string,time:stamp
```

The **(input)** element of the Object Tracking window defines the object’s input fields names convention. Use the %character as a placeholder for the object number. For example, if the ‘x’ attribute of the <input> element has the value ‘obj\_%\_x’, then the window assumes that the x coordinate of the object is *obj\_0\_x*, *obj\_1\_x*, *obj\_2\_x*, ....If the <input> element is not used, then the standard analytic store output field name conventions are used.

3.17.4 Output Schema for Object Tracking Windows

The output of an Object Tracking window conforms to a wide or a long schema, depending of the value of the *mode* attribute of the element. For wide mode, the output schema includes all non-object fields of the input window. This includes the input key fields. Then, for each tracked object, the output schema includes the following fields:

```
prefix_density :int32
prefixN_id :int64 1 2 3
prefixN_label :string
prefixN_score :double
prefixN_x :double
prefixN_y :double
prefixN_w :double
prefixN_h :double
prefixN_center_x :double 4
prefixN_center_y :double
prefixN_track_x :array(dbl) 5
prefixN_track_y :array(dbl)
```

1. Use the *prefix* attribute of the <output> element to define the value of *prefix*. The default value of *prefix* is ‘Object’.
2. The *N* value is the 0-based index of the object number. Use the *tracks* attribute of the <output> element to define the number of *output* objects.
3. The <.>\_id field contains the identifier of the object’s track.

4. The `<.>_center_x` and `<.>_center_y` fields contain the x and y coordinates of the center of the bounding box of the object's last detection.
5. The `<.>_track_x` and `<.>_track_y` fields contain the list of coordinates of the center of the bounding box of the successive object's positions.

When `velocity-vector="true"`, the following additional fields appear in the output schema:

`prefixN_vvect_x :double 1`

`prefixN_vvect_y :double`

1. The `<.>_vvect_x` and `<.>_vvect_y` fields contain the coordinates of the last velocity vector of the object.

For example, if `prefix='Object'`, `tracks=4`, and `velocity-vector="true"` and the input schema is the one specified in ("**Input Schema for Object Tracking Windows**") , then the output schema becomes the following:

```
frame_id*:int64,image:blob,objCount:int32,time:stamp,Object_density:int32,Object0_id:
int64,Object0_label:string,
Object0_score:double,Object0_x:double,Object0_y:double,Object0_w:double,Object0_h:dou
ble,
Object0_center_x:double,Object0_center_y:double,Object0_track_x:array(dbl),Object0_tr
ack_y:array(dbl),
Object0_vvect_x:double,Object0_vvect_y:double,Object1_id:int64,Object1_label:string,O
bject1_score:double,
Object1_x:double,Object1_y:double,Object1_w:double,Object1_h:double,Object1_center_x:
double,
Object1_center_y:double,Object1_track_x:array(dbl),Object1_track_y:array(dbl),Object1
_vvect_x:double,
Object1_vvect_y:double,Object2_id:int64,Object2_label:string,Object2_score:double,Obj
ect2_x:double,
Object2_y:double,Object2_w:double,Object2_h:double,Object2_center_x:double,Object2_ce
nter_y:double,
Object2_track_x:array(dbl),Object2_track_y:array(dbl),Object2_vvect_x:double,Object2_
vvect_y:double,
Object3_id:int64,Object3_label:string,Object3_score:double,Object3_x:double,Object3_y
:double,
Object3_w:double,Object3_h:double,Object3_center_x:double,Object3_center_y:double,
Object3_track_x:array(dbl),Object3_track_y:array(dbl),Object3_vvect_x:double,Object3_
vvect_y:double
```

For long mode, the output schema includes all non-object fields of the input window. This includes the input key fields. After those fields, the output schema includes the following:

```
prefix_density :int32
prefix_id :int64 key='true'
prefix_label :string
prefix_score :double
prefix_x :double
prefix_y :double
prefix_w :double
```

```
prefix_h :double  
prefix_center_x :double  
prefix_center_y :double  
prefix_track_x :array(dbl)  
prefix_track_y :array(dbl)
```

When velocity-vector="true", the output schema contains the following additional fields:

```
prefix_vvect_x :double  
prefix_vvect_y :double
```

The value of prefix is defined by the prefix attribute of the element. The default value of prefix is 'Object'.

For example, when prefix='Object', velocity-vector="true", and the input schema is the one specified in ("Input Schema for Object Tracking Windows") , the output schema becomes the following:

```
frame_id*:int64,image:blob,objCount:int32,time:stamp,Object_density:int32,Object_id*:  
int64,Object_label:string,  
Object_score:double,Object_x:double,Object_y:double,Object_w:double,Object_h:double,  
Object_center_x:double,Object_center_y:double,Object_track_x:array(dbl),Object_track_  
y:array(dbl), Object_vvect_x:double,Object_vvect_y:double
```

### 3.17.5 Reference

Bockinski, Erik, Eiselein, Volker, and Sikora, Thomas. (**"High-Speed Tracking-by-Detection Without Using Image Information."**) IEEE International Conference on Advanced Video and Signal-based Surveillance, August 2017, Lecce, Italy.

## 3.18 Using Pattern Windows

### 3.18.1 Overview of Pattern Windows

Use a Pattern window to detect events of interest (EOIs) as they stream through. To create a Pattern window, specify the list EOIs and assemble them into an expression that uses logical operators. The expression can also include temporal conditions.

Specify EOIs by providing the following:

- A pointer for the window from where the event is coming
- A string name for the EOI
- A WHERE clause on the fields of the incoming event. The WHERE clause can include a number of unification variables (bindings).

Logical Operator	Function
and	All of its operands are true. Takes any number of operands.
or	Any of its operands are true. Takes any number of operands.
fby	Each operand is followed by the one after it. Takes any number of operands.
not	The operand is not true. Takes one operand.
nonoccur	The operand never occurs. Takes one operand.
is	Ensure that the following event is as specified.

To apply a temporal condition to the fby function, append the condition to the function inside braces. For example, specify

```
fby {1 hour}(event1,event2)
```

when event2 happens within an hour of event1. Specify

```
fby{10 minutes}(event1,event2,event3)
```

when event3 happens within ten minutes of event2, and event2 happens within ten minutes of event1.

For information about the XML elements associated with a Pattern window, see( "**window-pattern**" )and ("**XML Language Elements Relevant to Pattern Windows**")

### 3.18.2 Creating Patterns

Here is an XML example of a pattern from a broker surveillance model. EOIs are specified within *event* elements. The pattern logic is specified within a *logic* element.

```
<pattern>
  <events>
    <event name='e1'>((buysellflg==1) and (broker == buyer)
and (s == symbol) and (b == broker) and (p == price))</event>
    <event name='e2'>((buysellflg==1) and (broker != buyer)
and (s == symbol) and (b == broker))</event>
    <event name='e3'><![CDATA[ ((buysellflg==0) and (broker == seller)
and (s == symbol) and (b == broker) and (p < price))] ]></event>
  </events>
```

```

<logic>fby{1 hour}(fby{1 hour}(e1,e2),e3)</logic>
...
</output>
</pattern>
<pattern>
<events>
<event name='e1'>((buysellflg==0) and (broker == seller)
and (s == symbol) and (b == broker))</event>
<event name='e2'>((buysellflg==0) and (broker != seller)
and (s == symbol) and (b == broker))</event>
</events>
<logic>fby{10 minutes}(e1,e2)</logic>
...
</pattern>
</patterns>
</window-pattern>

```

Here is an XML example of a pattern from an e-commerce model:

```

<pattern>
<events>
<event name='e1'>eventname=='ProductView'
and c==customer and p==product</event>
<event name='e2'>eventname=='AddToCart'
and c==customer and p==product</event>
<event name='e3'>eventname=='CompletePurchase'
and c==customer</event>
<event name='e4'>eventname=='Sessions'
and c==customer</event>
<event name='e5'>eventname=='ProductView'
and c==customer and p!=product</event>
<event name='e6'>eventname=='EndSession'
and c==customer</event>
</events>
<logic>fby(e1,fby(e2,not(e3)),e4,e5,e6)</logic>
...
</pattern>

```

You can define multiple patterns within a Pattern window. Each pattern typically has multiple EOIs, possibly from multiple windows or just one input window. Suppose that there is a single window that feeds a Pattern window, and the associated schema is as follows:

```
<schema>
<fields>
<field name="ID" type="int32" key="true"/>
<field name="symbol" type="string"/>
<field name="price" type="double"/>
<field name="buy" type="int32"/>
<field name="tradeTime" type="date"/>
</fields>
</schema>
```

Suppose further that there are two EOIs and that their relationship is temporal. You are interested in one event followed by the other within some period of time. This is depicted in the following code segment:

```
<pattern>
<events>
<!-- Someone buys IBM at price > 150.00 followed within 5
seconds of buying MSFT at price > 110.00 -->
<event name="e1">symbol=="IBM" and price > 150.00 and b==buy</event>
<event name="e2">symbol=="MSFT" and price > 110.00 and b==buy</event>
</events>
<logic>fby{5 seconds}(e1, e2)</logic>
...
</pattern>
```

There are two EOIs, *e1* and *e2*. The beginning of the WHERE clauses is standard: *symbol==constant and price>constant*. The last part of each WHERE clause is where event unification occurs.

Because *b* is not a field in the incoming event, it is a free variable that is bound when an event arrives. It matches the first portion of the WHERE clause for event *e1* (for example, an event for IBM with price > 150.00.) In this case, *b* is set to the value of the field *buy* in the matched event. This value of *b* is then used in evaluating the WHERE clause for subsequent events that are candidates for matching the second event of interest *e2*. The added unification clause and *b == buy* in each event of interest ensures that the same value for the field *buy* appears in both matching events. The *FBY* operator (*fby*) is sequential in nature. A single event cannot match on both sides. The left side must be the first to match on an event, and then a subsequent event could match on the right side. The *AND* and *OR* operators are not sequential. Any incoming event can match EOIs on either side of the operator and for the first matching EOI causes the variable bindings. Take special care in this case, as this is rarely what you intend when you write a pattern. For example, suppose that the incoming schema is as defined previously and you define the following pattern:

```

<pattern>
<events>
<!-- Someone buys or sells IBM at price > 150.00 and also
buys or sells IBM at a price > 152.00 within 5 seconds -->
<event name="e1"> symbol=="IBM" and price > 150.00</event>
<event name="e2"> symbol=="IBM" and price > 152.00</event>
</events>
<logic>fby{5 seconds}(e1, e2)</logic>
...
</pattern>

```

Now suppose an event comes into the window where symbol is "IBM" and price is "152.10". Because this is an AND operator, no inherent sequencing is involved, and the WHERE clause is satisfied for both sides of the "and" by the single input event. Thus, the pattern becomes true, and event e1 is the same as event e2. This is probably not what you intended. Therefore, you can make slight changes to the pattern as follows:

```

<pattern>
<events>
<!-- Someone buys or sells IBM at price > 150.00 and <=152.00 and also
buys or sells IBM at a price > 152.00 within 5 seconds -->
<event name="e1"> symbol=="IBM" and price > 150.00 and price <= 152.00</event>
<event name="e2"> symbol=="IBM" and price > 152.00</event>
</events>
<logic>fby{5 seconds}(e1, e2)</logic>
...
</pattern>

```

Suppose that you specify a temporal condition for an AND operator such that event e1 and event e2 must occur within five seconds of one another. In that case, temporal conditions for each of the events are optional.

### 3.18.3 State Definitions For Operator Trees

Operator trees can have one of the following states:

- initial - no events have been applied to the tree
- waiting - an event has been applied causing a state change, but the left (and right, if applicable) arguments do not yet permit the tree to evaluate to TRUE or FALSE
- TRUE or FALSE - sufficient events have been applied for the tree to evaluate to one of these logical Boolean values

The state value of an operator sub-tree can be FIXED or not-FIXED. When the state value is FIXED, no further events should be applied to it. When the state value is not-FIXED, the state value could change based on application of an event. New events should be applied to the sub-tree.

When a pattern instance fails to emit a match and destroys itself, it folds. The instance is freed and removed from the active pattern instance list. When the top-level tree in a pattern instance (the root node) becomes FALSE, the pattern folds. When it becomes TRUE, the pattern emits a match and destroys itself.

An operator tree (OPT) is a tree of operators and EOIs. Given that EOI refers to an event of interest or operator tree (EOI | OPT):

Table 3- 7 Expressions and Associated Outcomes

Expression	Outcome
not EOI	A Boolean negation. This remains in the waiting state until EOI evaluates to TRUE or FALSE. Then it performs the logical negation. It becomes FIXED only when EOI becomes FIXED <b>Note:</b> It is recommended to always use not with a time limit. Otherwise, you could wait indefinitely for something not to occur.
not OPT	A Boolean negation. This remains in the waiting state until OPT evaluates to TRUE or FALSE. Then it performs the logical negation. It becomes FIXED only when OPT becomes FIXED
notoccur EOI	Becomes TRUE on application of an event that does not satisfy the EOI, but it is not marked FIXED. This implies that more events can be applied to it. As soon as it sees an event that matches the EOI, it becomes FALSE and FIXED
notoccur EOI	Not Permitted
EO or EO	An event that is always applied to all non-FIXED sub-trees. It becomes TRUE when one of its two sub-trees become TRUE. It becomes FALSE when both of the sub- trees becomes FASLE. It is FIXED when one of its sub-trees is TRUE and FIXED if both of its sub-trees are FALSE and not FIXED
EO and EO	An event that is always applied to all non-FIXED sub-trees. It becomes TRUE when both of its two sub-trees become TRUE. It becomes FALSE when one of the sub- trees becomes FALSE. It is FIXED when one of its sub-trees is FALSE and FIXED or both of its sub-trees are TRUE and FIXED

Expression	Outcome
EO FBY EO	<p>Attempts to complete the left hand side (LHS) with the minimal number of event applications before applying events to the right hand side (RHS). The apply rule is as follows:</p> <ul style="list-style-type: none"> <li>• If the LHS is not TRUE or FALSE, apply event to the LHS until it become TRUE or FALSE.</li> <li>• When the LHS becomes FALSE, set the followed by state to FALSE and become FIXED.</li> <li>• When the LHS becomes TRUE, apply all further events to the RHS until the RHS becomes TRUE or FALSE. If the RHS becomes FALSE, set the FBY state to FALSE and FIXED, if it becomes TRUE set the FBY state to TRUE and FIXED.</li> </ul> <p>This algorithm seeks the minimal length sequence of events that completes an FBYpattern.</p>
is EOI	bBecomes TRUE on the application of an event that satisfies the EOI and FALSE otherwise. Becomes FIXED on the first application of an event.
is OPT	Not Permitted

Table 3- 8 Logic Underlying a Set of Sample Operator Trees and Events

Logic	Description
(a fby b)	Detect a, ..., b, where ... can be any sequence.
( a fby ( (notoccur c) and b ) )	Detect a, ..., b: but there can be no c between a and b
( a fby (not c) ) fby (not (not b))	Detect a, X, b: when X cannot be c.
(( (a fby b) fby (c fby d) ) and (notoccur k) )	Detect a, ..., b, ..., c, ..., d : but k does not occur anywhere in the sequence.
a fby (notoccur(c) and b)	Detect a FBY b with no occurrences of c in the sequence
is(a) fby is(b)	Detect a FBY b directly, with nothing between a and b.
a fby (b fby ((notoccur c) and d))	Detect a ... b ... d, with no occurrences of c between a and b.
(notoccur c) and (a fby (b fby d))	Detect a ... b ... d, with no occurrences of c anywhere

### 3.18.4 Restrictions on Patterns

The following restrictions apply to patterns that you define in Pattern windows:

- The data type of the key field must be int64.
- An event of interest should be used in only one position of the operator tree. For example, the following code would return an error:

```
a fby (notoccur(a) and b)
```

- Pattern windows work only on Insert events.

If there might be an input window generating updates or deletions, then you must place a Procedural window between the input window and the Pattern window. The Procedural window then filters out or transforms non- insert data to insert data.

Patterns also generate only Inserts. The events that are generated by Pattern windows are indications that a pattern has successfully detected the sequence of events that they were defined to detect. The schema of a pattern consists of a monotonically increasing pattern HIT count in addition to the non-key fields that you specify from events of interest in the pattern.

`dfESPpattern::addOutputField()` and `dfESPpattern::addOutputExpression()`

- When defining the WHERE clause expression for pattern events of interests, binding variables must always be on the left side of the comparison (like *bindvar == field*) and cannot be manipulated.

For example, the following *addEvent* statement would be flagged as invalid:

```
e1 = consec->addEvent(readingsWstats, "e1",
"((vmin < aveVMIN) and (rCNT==MeterReadingCnt) and (mID==meterID));");
e2 = consec->addEvent(readingsWstats, "e2",
"((mID==meterID) and (rCNT+1==MeterReadingCnt) and (vmin < aveVMIN));");
op1 = consec->fby_op(e1, e2,28800000001);
```

Consider the WHERE clause in **e1**. It is the first event of interest to match because the operator between these events is a followed-by. It ensures that event field **vmin** is less than field **aveVMIN**. When this is true, it binds the variable **rCNT** to the current meter reading count and binds the variable **mID** to the **meterID** field. Now consider **e2**. Ensure the following:

- the **meterID** is the same for both events
- the meter readings are consecutive based on the **meterReadingCnt**
- **vmin** for the second event is less than **aveVMIN**

The error in this expression is that it checked whether the meter readings were consecutive by increasing the **rCNT** variable by 1 and comparing that against the current meter reading. Variables cannot be manipulated. Instead, you confine manipulation to the right side of the comparison to keep the variable clean. The following code shows the correct way to accomplish this check. You want to make sure that meter readings are consecutive (given that you are decrementing the meter reading field of the current event, rather than incrementing the variable).

```
e1 = consec->addEvent(readingsWstats, "e1",
"((vmin < aveVMIN) and (rCNT==MeterReadingCnt) and (mID==meterID));");
e2 = consec->addEvent(readingsWstats, "e2",
"((mID==meterID) and (rCNT==MeterReadingCnt-1) and (vmin < aveVMIN));");
op1 = consec->fby_op(e1, e2,28800000001);
```

### 3.18.5 Using Stateless Pattern Windows

Because both the input and output of a Pattern window are unbounded and insert-only, they are natural candidates for stateless windows (that is, windows with index type *pi\_EMPTY*). Usually, you want to have a Copy window with a retention policy follow any insert-only window.

Pattern windows are automatically marked as insert-only. They reject records that are not Inserts. Thus, no problems are encountered when you use an index type of *pi\_EMPTY* with Pattern windows. If a Source window feeds the Pattern window, then it needs to be explicitly told that it is insert-only, using the *dfESPwindow::setInsertOnly()* call. This causes the Source window to reject any events with an opcode other than Insert, and permits an index type of *pi\_EMPTY* to be used. Stateless windows are efficient with respect to memory use. More than one billion events have been run through pattern detection scenarios such as this with only modest memory use (less than 500MB total memory).

```
Source Window [insert only, pi_EMPTY index] --> PatternWindow[insert only,
    pi_EMPTY index]
```

### 3.18.6 Enabling Pattern Compression

When an event affects a pattern and partially completes it, the event is stored in the pattern instance for future use. When a pattern event completes through a later sequence of events, the stored event is accessed. When the system has an exceptionally large number of partially completed patterns, a large amount of memory might be required for the associated stored events. To address this issue, you can compress partially completed patterns and then uncompress them upon pattern completion.

There are two ways to enable pattern compression on projects:

- In C++, call *dfESPproject::setPatternCompression(true)* before a project is started.
- In XML, use the *compress-open-patterns='true'* attribute on a *project* element.

Pattern compression can be useful when a project has a very large number of open patterns waiting for possible completion. It can decrease pattern memory usage by as much as 40% at the expense of a slight increase in CPU usage.

### 3.18.7 Enabling the Heartbeat Interval

Patterns that can time out are sent heartbeats by the system. When there are millions of open, uncompleted patterns, the default heartbeat interval of one second is too short. In this case, the system attempts to time out every pattern each second, and that can slow system performance.

To remedy this problem, tune the heartbeat interval:

- In C++, call *dfESPproject::setHeartbeatInterval(int number-of-seconds)* before you start the *project*.
- In XML, use the following attribute on the *project* element: *heartbeat-interval='number-of-seconds'*

Set the number-of-seconds as high as is practical.

### 3.18.8 Using Index Generation Functions

An index generation function selects fields in an event to sort them before patterns are applied. For example, consider the following EOIs and pattern logic:

```
<event name='e1'>(sym == symbol) and (price > 100)</event>  
<event name='e2'>(sym == symbol) and (price < 100)</event>  
<logic>fby(e1,e2)</logic>
```

Here, you are detecting events that have the same symbol and tracking them when the price first exceeds 100 and then falls below 100. Because *sym* is part of the EOI, every event that streams into the Pattern window is checked against every instance of this pattern. When applied to many events, that evaluation could degrade project performance.

Alternatively, you can generate an index of *sym* on the pattern:

```
<window-pattern...index='sym'>
```

In this case, every event is first sorted by *sym* because it is the index. After that, the pattern is applied to events. When *sym='IBM'*, only events that match the index are evaluated. Hence, you can simplify the EOIs:

```
<event name='e1'>price > 100</event>  
<event name='e2'>price < 100</event>
```

This can improve performance because the Pattern window has fewer events to evaluate.

## 3.19 Using Procedural Windows

### 3.19.1 Overview

Use a Procedural window to specify an arbitrary number of input windows and input handler functions for each input window. You can define Procedural window input handlers in one of two ways:

- Use a callable C++ function in a shared library. Use the *cxx-plugin* XML element to define this type of input handler.
- Use DATA step code that calls into an external SAS session that runs on the same server that produces one output row for each input row.

---

#### Note

You must configure Base SAS 9.4 to use UTF-8 before running DATA step code in a Procedural window.

---

**Note**

Support for SAS Micro Analytic Service (MAS) modules and stores has moved from the Procedural window to the Calculate window.

When an input event arrives, the input handler registered for the matching window is called. Then the events produced by that input handler function are generated.

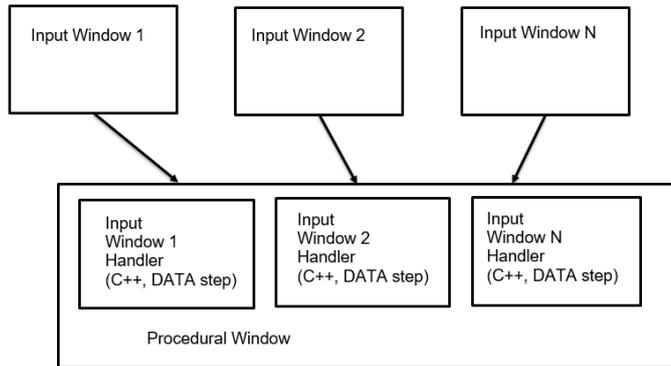


Figure 3-6 Procedural Window with Input Handlers

In order for the state of the Procedural window to be shared across handlers, an instance-specific context object (such as *dfESPpcontext*) is passed to the handler function. Each handler has full access to what is in the context object. The handler can store data in this context for use by other handlers, or by itself during future invocations. For information about the XML elements associated with Procedural windows, see ("**window-procedural**") and ("**XML Language Elements Relevant to Procedural Windows**")

### 3.19.2 Using C++ Window Handlers

Here is an example of the signature of a Procedural window handler written in C++.

```

typedef bool (*dfESPEvent_func)(dfESPpcontext *pc,
dfESPschema *is, dfESPEventPtr nep,
dfESPEventPtr oep, dfESPschema *os,
dfESPptrVect<dfESPEventPtr>&oe);
  
```

The procedural context is passed to the handler. The input schema, the new event, and the old event (in the case of an update) are passed to the handler when it is called. The final parameters are the schema of the output event (the structure of events that the Procedural window produces) and a reference to a vector of output events. It is this vector where the handler needs to push its computed events.

Only one input window is defined, so define only one handler function and call it when a record arrives.

```

// This handler functions simple counts inserts, updates,
  
```

```
// and deletes.
// It generates events of the form "1,#inserts,#updates,
// #deletes"
//
bool opcodeCount(dfESPpcontext *mc, dfESPschema *is,
dfESPeventPtr nep, dfESPeventPtr oep, dfESPschema *os, dfESPptrVect
<dfESPeventPtr>& oe) {
    derivedContext *ctx = (derivedContext *)mc;
    // Update the counts in the past context. switch (nep->getOpcode()) {
    case dfESPeventcodes::eo_INSERT: ctx->numInserts++;
    break;
    case dfESPeventcodes::eo_UPDATEBLOCK: ctx->numUpdates++;
    break;
    case dfESPeventcodes::eo_DELETE: ctx->numDeletes++;
    break;
    }

    // Build a vector of datavars, one per item in our output
    // schema, which looks like: "ID*:int32,insertCount:
    // int32,updateCount:int32,deleteCount:int32"
    dfESPptrVect<dfESPdatavarPtr> vect; os->buildEventDatavarVect(vect);

    // Set the fields of the record that we are going to produce.
    vect[0]->setI32(1); // We have a key of only 1, we keep updating one record. vect[1]-
    >setI32(ctx->numInserts);
    vect[2]->setI32(ctx->numUpdates); vect[3]->setI32(ctx->numDeletes);

    // events.
    dfESPeventPtr ev = new dfESPevent();
    ev->buildEvent(os, vect, dfESPeventcodes::eo_UPSERT, dfESPeventcodes::ef_NORMAL);
    oe.push_back(ev);

    // Free space used in constructing output record. vect.free();

    return true;
}
```

The following example shows how this fits together in a Procedural window:

```
<project name='project' pubsub='auto' threads='1'>
<contqueries>
<contquery name='contQuery' trace='proceduralWindow'>
<windows>
<window-source name='sourceWindow'>
<schema>
<fields>
<field name='ID' type='int32' key='true'/>
<field name='symbol' type='string'/>
<field name='quantity' type='int32'/>
<field name='price' type='double'/>
</fields>
</schema>
<connectors>
<connector class='fs' name='publisher'>
</properties>
```

```

<property name='type'>pub</property>
<property name='fstype'>csv</property>
<property name='fname'>input.csv</property>
<property name='transactional'>>true</property>
<property name='blocksize'>1</property>
</properties>
</connector>
</connectors>
</window-source>
<window-procedural name='proceduralWindow'>
<schema>
<fields>
<field name='ID' type='int32' key='true' />
<field name='insertCount' type='int32' />
<field name='updateCount' type='int32' />
<field name='deleteCount' type='int32' />
</fields>
</schema>
<cxx-plugin-context name='plugin' function='get_derived_context' />
<cxx-plugin source='sourceWindow' name='plugin' function='countOpcodes' />
</window-procedural>
</windows>
<edges>
<edge source='sourceWindow' target='proceduralWindow' />
</edges>
</contquery>
</contqueries>
</project>

```

---

**Note**

The `registerMethod` class consists of the following functions:

- `registerMethod_SO`
  - `registerMethod_DSEXT`
- 

For information about these functions, refer to the complete class and method documentation that is available at

**`$DFESP_HOME/doc/html/index.html`**.

Whenever the Procedural window sees an event from the Source window (sw), it calls the handler `opcodeCount` with the context `mc`, and produces an output event.

An application can use the `dfESPengine::logBadEvent()` member function from a Procedural window to log events that it determines are invalid. For example, you can use the function to permit models to perform data quality checks and log events that do not pass. There are two common reasons to reject an event:

- The event contains a null value in one of the key fields.
- The opcode that is specified conflicts with the existing window index (for example, two Inserts of the same key, or a Delete of a non-existing key).

### 3.19.3 Using DATA Step Window Handlers

#### Overview

When you write a handler using DATA step statements, the window:

- receives an incoming event
- executes DATA step code against the data in the event
- returns an output event

All fields in the input window are seen as variables by the DATA step.

Use a SET statement to receive the event and populate the DATA step variables. Use an OUTPUT statement to create an Upsert event, which is returned to the Procedural window. Both the SET and OUTPUT statements reference the event stream processing libref, which requires the sasioesp load module to be in the SAS search path.

#### Configuration

When you configure a model that contains a Procedural window that executes DATA step code, the project must contain the *<ds-initializer>* element:

```
<ds-initialize  
  sas-log-location='@SAS_LOG_DIR@' sas-connection-key='5555'  
  sas-command='sas -path @DFESP_HOME@/lib'  
>
```

- The *sas-log-location* is optional. If you do not specify it, the SAS log is placed in the directory where the event stream processing server was started.
- The *sas-connection-key* is optional. This key is used as the shared memory and semaphore key to communicate with Base SAS. It is a system-level resource (like a port) and needs to be unique per event stream processing server executing on the system. When there is only one event stream processing server running on the system, specify the default value of 5555.
- The *sas-command* starts a SAS session. It requires the **-path** option in order to find the access engine. Within the Procedural window itself, specify the *<ds-external>* element as follows:

```
<ds-external source='request'  
  trace='false'  
  code-file='@SAS_SOURCE_DIR@/score.sas'  
  connection-timeout='5'  
  max-string-length='32'  
>
```

- The *source* attribute designates the Source window to which the remaining attributes apply.
- The *trace* flag turns on output to the SAS log. Use this flag only during the model development phase with small amounts of test data.

- The *code-file* attribute identifies which SAS program executes on events that arrive from the Source window.
- The *connection-timeout* is measured in seconds. The default value is 60 seconds. Consider increasing the value under the following circumstances:
  - when your SAS code is complex
  - when your code takes a long time to compile
  - when Base SAS performs extensive one time initialization, such as loading hash tables:
- The *max-string-length* attribute communicates to Base SAS the maximum length of any string sent in an event from Edge Streaming Analytics to Base SAS.

**Referencing in a DATA Step**

Reference Edge Streaming Analytics in a DATA step as follows:

```
data esp.output;
set esp.input;
score = a * ranuni(104) + b; run;
```

- The DATA statement must designate esp.output as the output data set. When an observation is written to that data set, it actually is returned to the Procedural window as an Upsert event.
- The SET statement waits for the arrival of an event, and moves event data into DATA step variables.

**Supported Data Types**

The following mapping of event stream processor to DATA step data types is supported:

Event Streaming Type Processor	DATA Step Data Type
ESP_INT32	Numeric variable.
ESP_INT64	Numeric variable.
ESP_DOUBLE	Numeric variable.
ESP_TIMESTAMP ESP DATETIME	Numeric variable whose value is the number of seconds since Jan 1, 1960.
ESP_UTF8STR	Character Variable.
ESP_MONEY	Not supported.

**Known Limitations**

- Currently this functionality is supported only on Linux platforms.
- Some DATA step statements and options do not make sense when you use them in a real-time event processing context. For example, you should not use the END= option in the SET statement. In a real-time system, it is not known whether there are more records to come.

- The Procedural window uses shared memory and system semaphores to communicate with Base SAS. These are system wide resources, similar to sockets. Therefore, event stream processing servers that run on the same system cannot use the same set of keys to communicate with Base SAS. You can use the *sas-connection-key* attribute on the *ds-initialize* element to alter the starting key for one of the event stream processing servers.
- Edge Streaming Analytics supports mixed-case field names. Base SAS does not.
- Edge Streaming Analytics supports varying length strings. The SAS access engine interface does not. Use the *max-string-length* attribute on the Procedural window's *ds-external* element to declare the length of the maximum expected string value that is sent to Base SAS.

### 3.19.4 Converting DS2 Table Server Code to Run in SAS Micro Analytic Service Mode

Previous versions of Edge Streaming Analytics supported a Procedural window input handler that used DS2 code running in table server mode. Support for this functionality was deprecated at Release 5.2. To run DS2 code in a model, you now must define a SAS Micro Analytic Service module at the project level and a SAS Micro Analytic Service map in a Calculate window.

The key differences between running DS2 code in table server mode and running it in a SAS Micro Analytic Service map are as follows:

- The table server used a lazy binding on symbols. All input fields were always passed to the DS2 method. All declared variables in the DS2 method, provided that they matched a field name in the schema of the Procedural window, were exported to derived events.
- A SAS Micro Analytic Service module uses an interface similar to a function call, where only the input fields that you declare in the DS2 method signature are passed into the method. Only method parameters declared as *in\_out* are exported from the DS2 method and then assigned to correspondingly named output fields. Any input field that matches an output field in name and type is echoed through the Calculate window. You do not need to explicitly pass matched fields to the DS2 method.
- A DS2 method running under the SAS Micro Analytic Service supports a read/write shared vector that can be shared across SAS Micro Analytic Service threads and packages. This enables fast concurrent data sharing when you use multiple threads.

The following example shows how to convert code running in table server mode to code that can run in a SAS Micro Analytic Service module. Suppose you have the following Source window schema:

```
id*:int64,sensorName:string,sensorValue:double
```

Suppose you have the following output window schema:

```
id*:int64,sensorName:string,sensorValue:double,absValue:double,sqrtValue:double
```

Suppose you have this DS2 code in table server mode:

```
ds2_options cdump; data esp.out;  
  
dcl double absValue; dcl double sqrtValue; method run();  
  
set esp.in;
```

```
absValue = abs(sensorValue); sqrtValue = sqrt(sensorValue);
end; enddata;
```

You would use the following DS2 code in a SAS Micro Analytic Service module:

```
ds2_options sas;
package pl/overwrite=yes;
method compute(double sensorValue, in_out double absValue, in_out double_sqrtValue);
absValue = abs(sensorValue);
sqrtValue = sqrt(sensorValue); end;
endpackage;
```

When your DS2 code processes opcodes, you must convert opcode values from integers to strings before you use it in a SAS Micro Analytic Service module.

Opcode	Table Server Mode (integer)	SAS Micro Analytic Service module (string)
Insert	1	Insert
Update	2	Update
Delete	3	Delete
Upsert	4	Upsert
Safe Delete	5	Safeddelete

You must also convert event flags.

Flag	Table Server Mode(integer)	SAS Micro Analytic Service module (string)
Normal	1	N
Reyention	4	R

For example, consider the following DS2 code running in table server mode:

```
<ds2-tableserver source="Aggregate1">
<code>
<![CDATA[ds2_options cdump;
data esp.out;
method run();
set esp.in;
if _opcode = 2 or _opcode = 3 then_opcode = 1;
</code>
end;
enddata;]]>
</ds2-tableserver>
```

The following code performs the same evaluation and produces the same results:

```
<mas-module module="opcode_module" language="ds2" func-names="convert_opcode">  
<code>  
<![CDATA[ds2_options sas;  
package opcode_module/overwrite=yes;  
method convert_opcode(vvarchar(16) _inOpcode, in_out varchar _outOpcode); if  
(_inOpcode = 'delete' or _inOpcode = 'update') then_outOpcode = 'insert';  
else  
end;  
_outOpcode = _inOpcode;  
endpackage;]]>  
</code>  
</mas-module>
```

Although it is function call based, DS2 code used in a SAS Micro Analytic Service module can produce zero, one, or more than one output event for each input event. For more information, see the **(SAS Micro Analytic Service: Programming and Administration Guide.)**

## 3.20 Using Remove State Windows

Use a Remove State window to facilitate the transition of a stateful part of a model to a stateless part of a model. A Remove State window converts all events that it receives into Inserts and adds a field named eventNumber, which is a monotone-increasing sequential integer. This added field is the only key of the Remove State window.

For information about the XML elements associated with Remove State windows, see **(“window-remove-state”)**. Consider the following model:

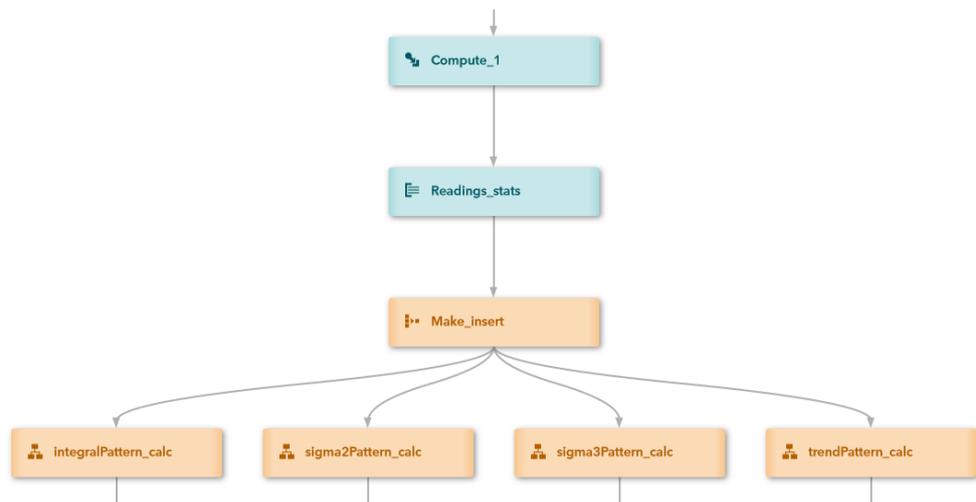


Figure 3-7 Stateful Windows Streaming into Stateless Pattern Window

Compute\_1, a Compute window, receives a stream of sensor and location data. It augments incoming events with new fields that are based on a manipulation of the incoming data. Compute\_1 streams augmented events into Readings\_stats, an Aggregate window. Readings\_stats groups events by a location and calculates values such as sum, average, standard deviation, and so on, for the group. Every time that a new event for a specific location streams into Reading\_stats, an Update event that contains these calculated values streams to Make\_insert, a Calculate window. Make\_insert contains a DS2 function that converts Update events into Insert events. Make\_insert streams those Insert events to a series of Pattern windows, which accept only Insert events. These Pattern windows further process the data. In this model, the only purpose of the Calculate window is to convert Update events into Insert events. You can replace Make\_insert with a Remove State window to accomplish the same result.

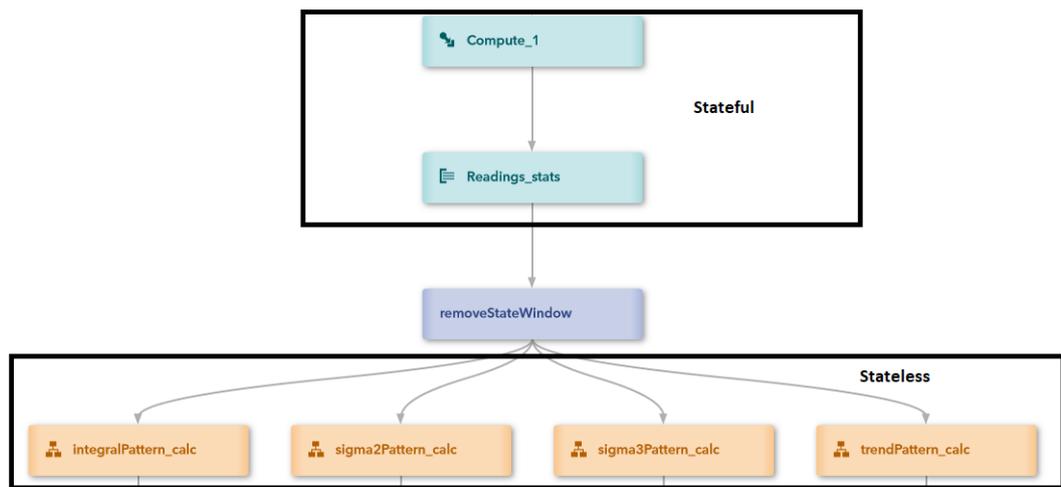


Figure 3-8 Using the Remove State Window

The generic form of the Remove State window schema is as follows:

```
eventNumber*:int64, [originalOC:string, originalFL:string,] <incoming_schema>
```

The following XML code shows a common case. The *incoming\_schema* is as follows:

```
symbol:string, true_test:integer
<window-remove-state name='removeStateWindow' add-log-fields='true' <!-- 1 -->
remove='retentionUpdates retentionDeletes'><!-- 2 -->
</window-remove-state>
```

1. Setting *add-log-fields='true'* adds the originalOC and originalFL fields specified in the schema.
2. You can configure the Remove State window to filter events by opcode or retention policy or combination of the two. Setting *remove='retentionDeletes retentionUpdates'* filters out Delete and Update events that have the retention flag set.

Sample output from this Remove State window is as follows. Three events are filtered:

```
<event opcode="insert" window="project/contQuery/removeStateWindow">
```

```

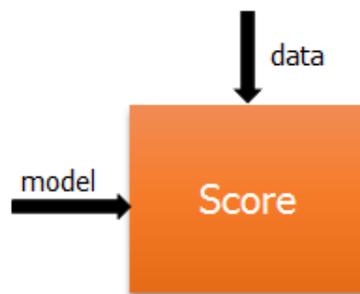
<value name="eventNumber">13</value> <!-- 1 -->
<value name="originalFL">N</value> <!-- 2 -->
<value name="originalOC">D</value> <!-- 3 -->
<value name="symbol">id</value>
<value name="true_test">21</value>
</event>
<event opcode="insert" window="project/contQuery/removeStateWindow">
<value name="eventNumber">14</value>
<value name="originalFL">N</value>
<value name="originalOC">D</value>
<value name="symbol">pd</value>
<value name="true_test">23</value>
</event>
<event opcode="insert" window="project/contQuery/removeStateWindow">
<value name="eventNumber">15</value>
<value name="originalFL">N</value>
<value name="originalOC">D</value>
<value name="symbol">pd</value>
<value name="true_test">23</value>
</event>

```

1. The eventNumber is inserted by the Remove State window into each event that passes through.
2. The original flag of the event number 13 was N (normal).
3. The original opcode of the event number 13 was D (Delete).

## 3.21 Using Score Windows

Score windows accept model events to make predictions for incoming data events. They generate scored data. You can use this score data to generate predictions based on the trained model. (No role is assigned to the outgoing edges, so they do not appear in the diagram.)



For more information, see [\(Edge Streaming Analytics: Using Streaming Analytics.\)](#)

## 3.22 Using Text Category Windows

Use a Text Category window to categorize a text field in incoming events. The text field can generate zero or more categories, with scores. Text Category windows are insert-only, so they require an index of *pi\_EMPTY*. For information about the XML elements associated with a Text Category window, see (“**window-textcategory**” .)

---

### Note

Without SAS Contextual Analysis, you do not have the MCO file that is required to initialize a Text Category window. For more information about SAS Contextual Analysis, see the *SAS Contextual Analysis: User's Guide*.

---

## 3.23 Using Text Context Windows

Use a Text Context window to abstract classified terms from an unstructured string field. Use this window to analyze a string field from an event's input to find classified terms. Events generated from the terms can be analyzed by other window types. For example, a Pattern window could follow a Text Context window to look for tweet patterns of interest. Text Context windows are insert-only, so they require an index of *pi\_EMPTY*.

For information about the XML elements associated with Text Context windows, see “**window-textcontext**”.

---

### Note

Without SAS Contextual Analysis, you do not have the language initialization files that are required to initialize a Text Context window. For more information about SAS Contextual Analysis, see the *SAS Contextual Analysis: User's Guide*.

---

## 3.24 Using Text Sentiment Windows

Use a Text Sentiment window to determine the sentiment of text in the specified incoming text field and the probability of its occurrence. The sentiment value is “positive,” “neutral,” or “negative.” The probability is a value between 0 and 1. Text Sentiment windows are insert-only, so they require the index type *pi\_EMPTY*.

For information about the XML elements associated with a Text Sentiment window, see (**"window-textsentiment" .**)

---

### Note

**Without SAS Sentiment Analysis Studio, you do not have the SAM file that is required to initialize a Text Sentiment window. For more information about SAS Sentiment Analysis Studio, see the SAS Sentiment Analysis Studio: User's Guide.**

---

The following Source window reads a CSV file named text\_sentiment\_data through a file and socket connector.

```
<window-source name='sourceWindow_01'>
<schema-string>ID*:int32,tstamp:date,msg:string</schema-string>
<connectors>
<connector class='fs'>
<properties>
<property name='type'>pub</property>
<property name='fstype'>csv</property>
<property name='fsname'>text_sentiment_data.csv</property>
<property name='dateformat'>%Y-%m-%d %H:%M:%S</property>
</properties>
</connector>
</connectors>
</window-source>
```

The data in the CSV file, which conforms to the schema specified by the Source window, looks something like this:

```
i n l 9/7/2010 16:09 I love my pickup truck
```

The Text Sentiment window specifies an empty primary index, the SAM file path, and the input string field or document to analyze. Change the samFile string to point to a specific SAM file.

```
<window-textsentiment name='textSentimentWindow' sam-file=samFile.sam' text-
field='msg' index='pi_EMPTY' />
```

Remember, you must have SAS Sentiment Analysis Studio to initialize a Text Sentiment window.

Place a copy window after the Text Sentiment window so that it can hold text sentiment events using a retention policy. Here, the retention policy is set to a sliding window of five minutes.

```
<window-copy name='copyWindow_01' index='pi_RBTREE'>
<retention type='bytime_sliding'>5 minutes</retention>
```

```
</window-copy>
```

Here are the edges that connect the windows.

```
<edges>
```

```
<edge source='sourceWindow_01' target='textSentimentWindow' />
```

```
<edge source='textSentimentWindow' target='copyWindow_01' />
```

```
</edges>
```

## 3.25 Using Text Topic Windows

### 3.25.1 Overview

Text topic windows run Text Topic models on events to score and identify themes in streaming text. Text Topic models are generated by SAS Visual Text Analytics or SAS Visual Data Mining and Machine Learning. Text Topic windows receive and process text from documents as string fields. Text Topic models enter a text topic window through an analytic store file. Text Topic windows are insert-only, so they require the index type `pi_EMPTY`.

For information about the XML elements associated with Text Topic windows, see ("**window-texttopic**").

For more information about creating text mining models and saving them to analytic store files, see *SAS Visual Data Mining and Machine Learning: Data Mining and Machine Learning Procedures*.

### 3.25.2 Example of Text Topic Window

Consider the following example. A single continuous query contains a Source window (*sourceWindow\_01*) and a Text Topic window (*textTopicWindow*). The Source window reads a file containing text that needs to be analyzed:

```
<window-source name="sourceWindow_01"
```

```
pubsub="true" collapse-updates="true"
```

```
index="pi_EMPTY" insert-only='true'>
```

```
<schema>
```

```
<fields>
```

```
<field name='did' type='string' key='true' />
```

```
<field name='text' type='string' />
```

```
</fields>
```

```
</schema>
```

```
<connectors>
```

```

<connector name="pub" class="fs">
<properties>
<property name="type"><![CDATA[pub]]>
</property>
  <property name="fstype"><![CDATA[csv]]></property>
<property name="fsname"><![CDATA[input/input.csv]]>
</property>
  <property name='dateformat'>%Y-%m-%d %H:%M:%S</property>
</properties>
</connector>
</connectors>
</window-source>

```

The Text Topic window receives the streaming events and analyzes the text according to the model in the analytic store file specified at the `window-texttopic` level:

```

<window-texttopic name="textTopicWindow"
  index="pi_EMPTY" pubsub="true"
  astore-file="../common/binary/astore"
  ta-path="../../../tools/textAnalytics/SASFoundation/9.4/misc/tktg"
  text-field="text"
  include-topic-name="true">
</window-texttopic>

```

Edges connect the source window to the Text Topic window at the end of the continuous query:

```

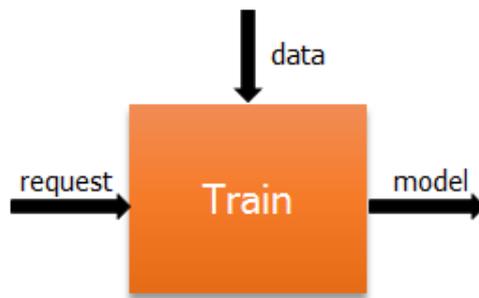
<edges>
<edge source="sourceWindow_01" target="textTopicWindow"/>
</edges>

```

## 3.26 Using Train Windows

*Train windows* receive data events and publish model events to Score windows. They use incoming data events to develop and adjust model parameters in real time. Often, the data is historical data from which to learn patterns. Incoming data should contain both the outcome that you are trying to predict and related variables.

Train windows can also receive request events. These events can adjust the learning algorithm while events continue to stream.



After a Train window has adjusted an algorithm, it writes the adjusted model to a Score window or a Model Supervisor window through a model event.

For more information, see *Edge Streaming Analytics: Using Streaming Analytics*.

### 3.27 Using Transpose Windows

You can conceptualize an event as a row that consists of multiple columns. Use a Transpose window to interchange an event’s rows as columns, or its columns as rows. Use attributes of the Transpose window to govern the rearrangement of data. For information about the XML elements associated with a Transpose window, see *window-transpose*.

The Transpose window accepts only Insert events, and it always has an index of `pi_EMPTY`. It provides two modes: wide and long.

Table 3- 9 Transpose Window Modes

Mode	Description
Wide	Produces one event per incoming event.
Description	Produces one or more event per incoming event.

Suppose that you are using wide mode to process information about the pitch, yaw, roll, and velocity of an aircraft in flight. The Source window uses the following schema:

```
<schema>
<fields>
<field name='ID' type='int64' key='true' />
<field name='PlaneID' type='string' /> <!-- 1 -->
<field name='TAG' type='string' /> <!-- 2 -->
<field name='value' type='double' /> <!-- 3 -->
<field name='time' type='stamp' />
<field name='lat' type='double' /> <!-- 4 -->
<field name='long' type='double' />
</fields>
</schema>
```

1. The value of the PlaneID field of the schema identifies which aircraft provides the data.
2. The value of the TAG field specifies whether the data contains the aircraft's pitch, yaw, roll, or velocity. The TAG field must be type string.
3. The value field records the value for the TAG and the specific time that the data is recorded.
4. The event captures the plane's latitude (lat) and longitude (long) when the pitch, yaw, roll, or velocity is recorded.

Now suppose that the following events stream through the Source window.

```
i,n,1,turboprop #1, pitch,1.1, 2017-08-30 15:21:00.000000,10,10
i,n,2,turboprop #2, velocity,-1.2, 2017-08-30 15:21:00.000001,20,20
i,n,3,turboprop #1, roll,-1.1, 2017-08-30 15:21:00.000002,11,11
i,n,4,turboprop #2, yaw,2.2, 2017-08-30 15:21:00.000003,21,21
i,n,5,turboprop #2, pitch,1.2, 2017-08-30 15:21:00.000004,22,22
i,n,6,turboprop #1, velocity,-1.1, 2017-08-30 15:21:00.000005,12,12
i,n,7,turboprop #2, roll,-1.2, 2017-08-30 15:21:00.000006,23,23
i,n,8,turboprop #1, yaw,2.1, 2017-08-30 15:21:00.000007,13,13
i,n,9,jet #1, pitch,1.3, 2017-08-30 15:21:00.000012,30,30
i,n,10,jet #2, velocity,-4.4, 2017-08-30 15:21:00.000013,40,40
i,n,11,jet #1, roll,-4.3, 2017-08-30 15:21:00.000014,31,31
i,n,12,jet #2, yaw,8.4, 2017-08-30 15:21:00.000015,41,41
i,n,13,jet #2, pitch,4.4, 2017-08-30 15:21:00.000016,42,42
i,n,14,jet #1, velocity,-4.3, 2017-08-30 15:21:00.000017,32,32
i,n,15,jet #2, roll,-4.4, 2017-08-30 15:21:00.000018,43,43
i,n,16,jet #1, yaw,8.3, 2017-08-30 15:21:00.000019,33,33
i,n,17,turboprop #1, pitch,23.1, 2017-08-30 15:21:06.000022,14,14
```

In these seventeen events, four planes stream data through the Source window: **turboprop #1**, **turboprop**

**#2**, **jet #1**, and **jet #2**. Each event that streams through contains either a pitch, velocity, roll, or yaw value at a specific time.

Now suppose you stream the input data through a Transpose window. Using mode='wide', you create a single event (row) that contains the pitch, velocity, roll, and yaw of each aircraft at a specific time.

```
<window-transpose name='transposeW'
```

```

mode='wide' <!-- 1 -->tag-name='
TAG'<!-- 2 -->tags-included='pitch,yaw,roll,velocity' <!-- 3 -->
tag-values='value,time' <!-- 4 -->
clear-timeout='never'
group-by='PlaneID' > <!-- 5 -->
<connectors>
...
</connectors>
</window-transpose>

```

1. Using wide mode produces output events that contain multiple columns, each based on data streamed through the input events.
2. The values of TAG in the input events determine the columns in output events.
3. Tags-included specify which specific values of TAG are to be included in the output events. Here, all four values of TAG are specified.
4. The value and time that are associated with pitch, yaw, roll, and velocity are included in the output event. The associated latitude and longitude are passed through, and not included.
5. Output events are grouped by the value of *PlaneID*.

Output columns are formed by taking the cross product of the following:

*{pitch, yaw, roll, velocity}* times *{value, time}*

This yields *pitch\_value*, *pitch\_time*, *yaw\_value*, *yaw\_time*, and so on.

Here are the first four events that stream from the Transpose window after processing events four events from the Source window:

```

<value name="pitch_time">1504106460000000</value>
<value name="pitch_value">1.100000</value>
<value name="roll_time">1504106460000002</value>
  <value name="roll_value">-1.100000</value>
</event> <!-- 3 -->
<event opcode="insert" window="newproject/cq/transposeW">
<value name="ID">4</value>
<value name="PlaneID">turboprop #2</value>
<value name="lat">21.000000</value>
<value name="long">21.000000</value>
<value name="velocity_time">1504106460000001</value>
<value name="velocity_value">-1.200000</value>
<value name="yaw_time">1504106460000003</value>
  <value name="yaw_value">2.200000</value>
</event> <!-- 4 -->

```

1. The first input event contains a pitch value of 1.1 for **turboprop #1**. In this output event and all subsequent output events, the Transpose window passes through *lat* and *long*

unchanged. Because *TAG* has the value **pitch** and time and value are the *tags-included*, the new variables *pitch\_time* and *pitch\_value* are generated. The values of *pitch\_time* and *pitch\_value* are inserted. There are no values of *roll*, *velocity*, or *yaw* to process in the first event.

2. The second input event contains a velocity value of -1.2 for **turboprop #2**. Because *TAG* has the value **velocity** and *time* and *value* are the *tags-included*, the new variables *velocity\_time* and *velocity\_value* are generated. There are no values of *pitch*, *roll*, or *yaw* to process in the second event.
3. The third input event contains a roll value of -1.1 for **turboprop #1**. The plane's *pitch\_time* and *pitch\_value* are retained from the first event. Because *TAG* has the value **roll** and *time* and *value* are the *tags-included*, the new variables *roll\_time* and *roll\_value* are generated. There are no values of *velocity* or *yaw* to process in the third event.
4. The fourth input event contains a yaw value of 2.2 for **turboprop #2**. The plane's *velocity\_time* and *velocity\_value* are retained from the second event. Because *TAG* has the value **yaw** and *time* and *value* are the *tags-included*, the new variables *yaw\_time* and *yaw\_value* are generated. There are no values of *pitch* or *roll* to process in the fourth event.

In this way, the Transpose window builds an event that contains every *TAG* value of interest per plane, because **PlaneID** is the value of group-by. By the time that all seventeen input events are processed, there are four events that capture each plane's pitch, roll, velocity, and yaw:

```
<value name="roll_value">-1.200000</value>
<value name="velocity_time">1504106460000001</value>
<value name="velocity_value">-1.200000</value>
<value name="yaw_time">1504106460000003</value>
<value name="yaw_value">2.200000</value>
</event>
<event opcode="insert" window="newproject/cq/transposeW">
<value name="ID">8</value>
<value name="PlaneID">turboprop #1</value>
<value name="lat">13.000000</value>
<value name="long">13.000000</value>
<value name="pitch_time">1504106460000000</value>
<value name="pitch_value">1.100000</value>
<value name="roll_time">1504106460000002</value>
<value name="roll_value">-1.100000</value>
<value name="velocity_time">1504106460000005</value>
<value name="velocity_value">-1.100000</value>
<value name="yaw_time">1504106460000007</value>
<value name="yaw_value">2.100000</value>
</event>
<event opcode="insert" window="newproject/cq/transposeW">
<value name="ID">15</value>
<value name="PlaneID">jet #2</value>
<value name="lat">43.000000</value>
<value name="long">43.000000</value>
<value name="pitch_time">1504106460000016</value>
<value name="pitch_value">4.400000</value>
<value name="roll_time">1504106460000018</value>
<value name="roll_value">-4.400000</value>
```

```

    <value name="velocity_time">1504106460000013</value>
    <value name="velocity_value">-4.400000</value>
    <value name="yaw_time">1504106460000015</value>
    <value name="yaw_value">8.400000</value>
</event>
<event opcode="insert" window="newproject/cq/transposeW">
<value name="ID">16</value>
<value name="PlaneID">jet #1</value>
<value name="lat">33.000000</value>
<value name="long">33.000000</value>
<value name="pitch_time">1504106460000012</value>
  <value name="pitch_value">1.300000</value>
  <value name="roll_time">1504106460000014</value>
  <value name="roll_value">-4.300000</value>
  <value name="velocity_time">1504106460000017</value>
  <value name="velocity_value">-4.300000</value>
  <value name="yaw_time">1504106460000019</value>
  <value name="yaw_value">8.300000</value>
</event>

```

The last input event updates the pitch of **turboprop #1**, which was collected at a later value of time.

```

<value name="pitch_value">23.100000</value>
<value name="roll_time">1504106460000002</value>
<value name="roll_value">-1.100000</value>
<value name="velocity_time">1504106460000005</value>
<value name="velocity_value">-1.100000</value>
<value name="yaw_time">1504106460000007</value>
<value name="yaw_value">2.100000</value>
</event>

```

Subsequent input events continue to update each plane's output event. Use the *clear-timeout* attribute of the Transpose window to specify a time after which output event values clear unless an input event value is received. Suppose that input data does not arrive from multiple devices or pieces of equipment (such as from two planes in the previous example). In that case, you do not need to use the *group-by* attribute. For example, consider the following input data. The Source window uses the same schema as before, but without *PlaneID*.

```

i,n,1,velocity, 234.0
i,n,2,roll,2.3
i,n,3,pitch,3.4
i,n,4,yaw,-1.1

```

You could transpose this data by specifying a Transpose window with the following attributes:

```
<window-transpose name='transposeW' mode='wide'  
tag-name='TAG'  
tags-included='pitch,yaw,roll,velocity' tag-values='value'  
clear-timeout='never'  
>  
</>
```

When you use long mode, you obtain the inverse results of wide mode. The Transpose window streams a number of events for each wide event that it receives. Input schema for the Source window must reflect combinations of fields.

For example, consider the schema of this Source window:

```
<schema>  
<fields>  
<field name='ID' type='int64' key='true' />  
<field name='PlaneID' type='string' />  
<field name='pitch_value' type='double' />  
<field name='pitch_time' type='stamp' />  
<field name='yaw_value' type='double' />  
<field name='yaw_time' type='stamp' />  
<field name='roll_value' type='double' />  
<field name='roll_time' type='stamp' />  
<field name='velocity_value' type='double' />  
<field name='velocity_time' type='stamp' />  
<field name='lat' type='double' />  
<field name='long' type='double' />  
</fields>  
</schema>
```

Suppose that you stream a wide event through that Source window.

```
I N 1 turboprop #2 1.2 21:00.0 2.2 21:00.0 -1.2 21:00.0 -1.2 21:00.0 20 20
```

A downstream Transpose window looks up combinations of tag-values and tags-included values in the incoming schema.

```
<window-transpose name='transposeL'  
mode='long' tag-name='TAG' tag-values='value,time'  
tags-included='pitch,yaw,roll,velocity'  
... </window-transpose>
```

Processing that input event, the Transpose window streams the following four output events:

```
I,N, 1,0,pitch,1.200000,2017-08-30 15:21:00.000004,turboprop #2,20.000000,20.000000  
I,N, 1,1,yaw,2.200000,2017-08-30 15:21:00.000003,turboprop #2,20.000000,20.000000  
I,N, 1,2,roll,-1.200000,2017-08-30 15:21:00.000006,turboprop #2,20.000000,20.000000  
I,N, 1,3,velocity,-1.200000,2017-08-30 15:21:00.000001,turboprop  
#2,20.000000,20.000000
```

If the downstream Transpose window does not find the appropriate combinations of *tag-values* and *tags-included* values, the window fails in finalization and does not permit the model to run.

## 3.28 Using Union Windows

A Union window unites two or more event streams using a strict policy or a loose policy. For information about the XML elements associated with a Union window, see `window-union`.

All input windows to a Union window must have the same schema. The default value of the strict flag is **true**, which means that the key merge from each window must semantically merge cleanly. In this case, you cannot send an Insert event for the same key using two separate input windows of the Union window.

Setting the strict flag set to **false** loosens the union criteria by replacing all incoming Inserts with Upserts. All incoming Deletes are replaced with safe Deletes. In this case, Deletes of a non-existent key fail without generating an error.

## 3.29 Understanding Retention

### 3.29.1 Introduction

Any Source or Copy window can set a retention policy. A window's retention policy governs how it introduces Deletes into the event stream. These Deletes work their way along the data flow, recomputing the model along the way. Internally generated Deletes are flagged with a retention flag, and all further window operations that are based on this Delete are flagged.

---

#### Note

Under some circumstances when you map a MAS method to a Procedural window, multiple derived events can be generated. In this case, retention flag propagation does not reliably occur.

---

For example, consider a Source window with a sliding volume-based retention policy of two. That Source window always contains at most two events. When an Insert arrives causing the

Source window to grow to three events, the event with the oldest modification time is removed. A Delete for that event is executed.

Retention Type	Description
time-based	Retention is performed as a function of the age of events. The age of an event is calculated as current time minus the last modification time of the event. Time can be driven by the system time or by a time field that is embedded in the event. A window with time-based retention uses current time set by the arrival of an event.
volume-based	Retention is based on a specified number of records. When the volume increases beyond that specification, the oldest events are removed.

Both time and volume-based retention can occur in one of two variants:

Retention Variant	Description
sliding	Specifies a continuous process of deleting events. Think of the retention window sliding continuously. For a volume-based sliding window, when the specified threshold is hit, one delete is executed for each insert that comes in.
jumping	Specifies a window that completely clears its contents when a specified threshold value is hit. Think of a ten-minute jumping window as one that deletes its entire contents every 10 minutes.

- *unit* (**minute, hour, day, week, month, or year**)
- *value*, which specifies the retention period. Use a positive integer that represents a multiple of the specified unit.

When an event streams into a window, it is rounded up to the specified UNIT.

For example, suppose that the *unit* is **month** and the first event arrives at **08-29-2017 17:21:00**. The rounded up time becomes **09-01-2017 00:00:00**. That time becomes the end of the first retention period. All other retention periods are intervals of the look back *unit*.

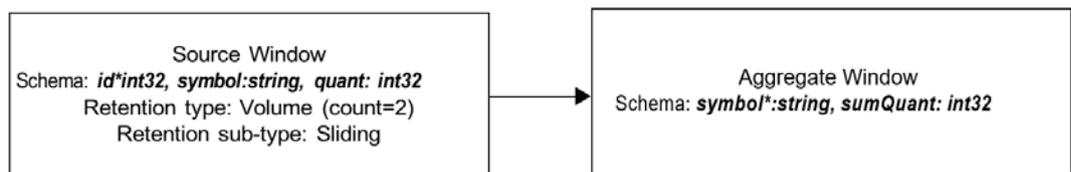
When the look back is 2, the next retention period would be [**09-01-2017 00:00:00, 11-01-2017 00:00:00**) and the next one after that would be: [**11-01-2017 00:00:00, 01-01-2018 00:00:00**).

A canonical set of events is a collapsed minimal set of events such that there is at most one event per key. Multiple updates for the same key and insert + multiple updates for the same key are collapsed. A window with retention generates a canonical set of changes (events). Then it appends retention-generated Deletes to the end of the canonical event set. At the end of the process, it forms the final output block.

Windows with retention produce output event blocks of the following form: *{<canonical output events>*,

*<canonical retention deletes>*. All other windows produce output blocks of the following form: *{<canonical output events>*.

Consider the following model:



The following notation is used to denote events [*<opcode>/<flags>: f1, ... ,fn*]

- Opcode
  - i — insert
  - d — delete
  - ub — update block — any event marked as ub is always followed by an event marked as d
- Flags
  - n — normal
  - r — retention generated

Suppose that the following events are streamed into the model:

Source In	Source Out — Aggregate In	Aggregate Out
[i/n: 1,ibm,10]	[i/n: 1,ibm,10]	[i/n: ibm,10]
Source In	Source Out — Aggregate In	Aggregate Out
[i/n: 2,ibm,11]	[i/n: 2,ibm,11]	[ub/n: ibm,21] [d/n: ibm,10]
Source In	Source Out — Aggregate In	Aggregate Out
[i/n: 3,sas,100]	[i/n: 3,sas,100] [d/r: 1,ibm,10]	[i/n: sas,100] [ub/r: ibm,11] [d/r: ibm,21]
Source In	Source Out — Aggregate In	Aggregate Out
[i/n: 4,ibm,12]	[i/n: 4,ibm,12] [d/r: 2,ibm,11]	[ub/r: ibm,12] [d/r: ibm,11]

When you run in retention-tracking mode, retention and non-retention changes are pulled through the system jointly. When the system processes a user event, the system generates a retention Delete. Both the result of the user event and the result of the retention Delete are pushed through the system. You can decide how to interpret the result. In normal retention mode, these two events can be combined to a single event by rendering its output event set canonical.

Source In	Source Out — Aggregate In	Aggregate Out
[i/n: 1,ibm,10]	[i/n: 1,ibm,10]	[i/n: ibm,10]
Source In	Source Out — Aggregate In	Aggregate Out
[i/n: 2,ibm,11]	[i/n: 2,ibm,11]	[ub/n: ibm,21] [d/n: ibm,10]
Source In	Source Out — Aggregate In	Aggregate Out
[i/n: 3,sas,100]	[i/n: 3,sas,100] [d/r: 1,ibm,10]	[i/n: sas,100] [ub/r: ibm,11] [d/r: ibm,21]

Source In	Source Out — Aggregate In	Aggregate Out
Source In	Source Out — Aggregate In	Aggregate Out
[i/n: 4,ibm,12]	[i/n: 4,ibm,12] [d/r: 2,ibm,11]	[ub: ibm,23] [d/n: ibm,11] [ub/r: ibm,12] [d/r: ibm,23]

Here, the output of the Aggregate window, because of the last input event, is non-canonical. In retention tracking mode, you can have two operations per key when the input events contain a user input for the key and a retention event for the same key.

**Note**

A window with pulsed mode set always generates a canonical block of output events. For the pulse to function as designed, the window buffers output events until a certain threshold time. The output block is rendered canonical before it is sent.

For examples of retention in practice, consider the following use cases for the four different retention type and variant combinations:

Retention Type and Variant	Use Case Examples
bytime_sliding	<ul style="list-style-type: none"> <li>Real-Time Dashboards</li> <li>Continuously analyzing the most recent events</li> </ul>
bytime_jumping	<ul style="list-style-type: none"> <li>Time Period Alarms</li> <li>"Notify us when average pressure hits 2 ATM within the current hour"</li> </ul>
bycount_sliding	<ul style="list-style-type: none"> <li>Real-Time Dashboards</li> <li>Continuously analyzing the n (100, 200, ...) most recent events</li> </ul>
bycount_jumping	<ul style="list-style-type: none"> <li>Analyze or Aggregate events in batches</li> <li>"Aggregate n events before loading to minimize impact on the DB"</li> </ul>

## 3.30 Understanding Primary and Specialized Indexes

### 3.30.1 Overview

In order to process events with opcodes, all windows must have a primary index. That index enables the rapid retrieval, modification, or deletion of events in the window.

Some windows have other indexes that serve specialized purposes.

- Source and Copy windows have an additional index to aid in retention.
- Join windows have left and right local indexes along with optional secondary indexes. These indexes help avoid locking and maintain data consistency.
- Aggregate windows have an aggregation index to maintain the group structure.

Window Type	Retention Index	Aggregation Index	Left Local Index	Right Local Index
Source Window Copy Window	Yes			
Join Window			Yes Optional secondary	Yes Optional secondary
Aggregate Window		Yes		

### 3.30.2 Fully Stateful Primary Indexes

Any window that uses a fully stateful primary index has a size equal to the cardinality of the unique set of keys. The exception is when a time or size-based retention policy is enforced. Events are absorbed, merged into a window’s index, and a canonical version of the change to the index is passed to all output windows.

Table 3- 10 Comparison of Fully Stateful Primary Index Types

Primary Index Type	Algorithm	Storage	Advantages	Drawbacks
pi_RBTREE	Red-Black Tree	In-memory	Ordered data and memory management provide smooth latencies	Slower than hash
pi_HASH	Open Hash	In-memory	Provides faster results than pi_RBTREE	Might lead to latency spikes when not properly sized.

Primary Index Type	Algorithm	Storage	Advantages	Drawbacks
pi_HLEVELDB	B-tree	Local Disk	Big Data <ul style="list-style-type: none"> <li>Index used for large source or Copy windows and for the left or right local dimension index in a join.</li> <li>Can be used when there is no retention or aggregation, or when you need a secondary index</li> </ul>	I/O performance <b>Note:</b> Do not use pi_HLEVELDB where a query, project, or any window name is written in MBCS.
pi_HLEVELDB_NC	B-tree	Local Disk	Offers the same advantages for big data storage as pi_HLEVELDB, with persistence across restarts.	I/O performance

When no retention policy is specified, a window that uses one of the fully stateful indices acts like a database table or materialized view. At any point, it contains the canonical version of the event log. Because common events are reference-counted across windows in a project, you should be careful that all retained events do not exceed physical memory. Use the Update and Delete opcodes for published events (as is the case with capital market orders that have a life cycle such as create, modify, and close order). However, for events that are Insert-only, you must use window retention policies to keep the event set bound below the amount of available physical memory.

### 3.30.3 Using pi\_HLEVELDB and pi\_HLEVELDB\_NC Primary Indexes

#### Overview

The pi\_HLEVELDB and pi\_HLEVELDB\_NC primary indexes store values on disk and maintain a most-recently used (MRU) in-memory cache. You can use these index types when there is no retention, aggregation, or need for a secondary index.

---

#### Note

**Do not use pi\_HLEVELDB or pi\_HLEVELDB\_NC when a query, project, or any window name is written with the variable-width encoding of MBCS.**

---

When you use pi\_HLEVELDB, you can stream millions of events into a window and consume just a few gigabytes of real memory. However, each time that you load the model (or restart the Streaming Server), the on-disk index is cleared. For long-lived servers or servers that have the ability to rebuild the index on restart, this might not be optimal usage of system resources. In that case, use the pi\_HLEVELDB\_NC index instead. That index essentially is pi\_HLEVELDB index that does not clear its on-disk locations upon initialization.

### Using a Large Lookup Table with Persistence Across Restarts

The following example shows how to enable an efficient no-regenerate lookup join with the following properties:

- The lookup table is stored on disk in a HyperLevel DB.
- A single copy of the data exists within the model. Only the in-memory cache is referenced during processing.
- The join is resilient to server restarts, that is, the lookup table does not need to be reloaded if the system bounces.

Suppose that you have a project named pr\_01 that contains a continuous query named cq\_01. The query contains two Source windows. Both of those windows are stateless, that is, they have indexes of pi\_EMPTY.

```
<window-source name="stream_source" index="pi_EMPTY" insert-only='true'>
<schema>
<fields>
<field name='ID' type='int64' key='true' />
<field name='matchID' type='int64' />
</fields>
</schema>
<connectors>
<connector class='fs' name='pub'>
<properties>
<property name='type'>pub</property>
<property name='fstype'>csv</property>
<property name='fname'>input/stream.csv</property>
<property name='transactional'>>true</property>
<property name='blocksize'>1</property>
<property name="dateFormat">%Y-%m-%d %H:%M:%S</property>
</properties>
</connector>
</connectors>
</window-source>
```

```

<window-source name='lookup_source' index='pi_EMPTY'>
<schema>
<fields>
<field name='ID' type='int64' key='true' />
<field name='Lookup' type='string' />
</fields>
</schema>
<connectors>
<connector class='fs' name='pub'>
<properties>
<property name='type'>pub</property>
<property name='fstype'>csv</property>
<property name='fsname'>input/500m.csv</property>
<property name='transactional'>>true</property>
<property name='blocksize'>64</property>
<property name="dateformat">%Y-%m-%d %H:%M:%S</property>
</properties>
</connector>
</connectors>
</window-source>

```

The Join window itself is also stateless. The right-index of the join has an index of pi\_HLEVELDB. Because this model performs the initial load of the lookup table, you want any old lookup data completely cleared out. Thus, you use pi\_HLEVELDB and not pi\_HLEVELDB\_NC.

```

<window-join name="join" index='pi_EMPTY'>
<join type="leftouter" no-regenerates='true' right-index='pi_HLEVELDB'>
<conditions>
<fields left="matchID" right="ID" />
</conditions>
</join>
<output>
<field-selection name="matchID" source="l_matchID" />
<field-selection name="lookup" source="r_Lookup" />
</output>
<connectors>
<connector class='fs' name='sub'>
<properties>
<property name='type'>sub</property>
<property name='fstype'>csv</property>
<property name='fsname'>join-out.csv</property>
<property name='snapshot'>>true</property>
<property name="dateformat">%Y-%m-%d %H:%M:%S</property>

```

```
</properties>  
</connector>  
</connectors>  
</window-join>
```

Edges connect the windows to yield the following:

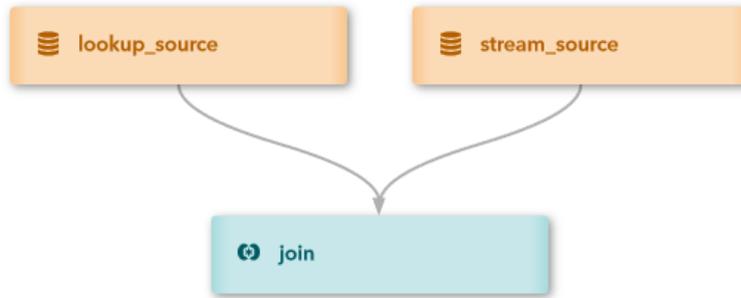


Figure 3-9 Continuous Query

Now suppose that you have 500 million rows of lookup data of the following form:

```
i,n,0, lookup string #0  
i,n,1, lookup string #1  
i,n,2, lookup string #2  
.  
.  
.  
i,n,499999997, lookup string #499999997  
i,n,499999998, lookup string #499999998  
i,n,499999999, lookup string #499999999
```

Running this model, which loads millions of rows of data, can take several minutes to run. After the lookup data is loaded, you use the following streaming data to test the lookup:

```
i,n,1,1  
i,n,2,2  
i,n,3,9999999
```

Running those Insert events yields the following results:

```
<event opcode='insert' window='pr_01/cq_01/join'>  
<value name='ID'>1</value>  
<value name='lookup'>lookup string #1</value>  
<value name='matchID'>1</value>  
</event>  
<event opcode='insert' window='pr_01/cq_01/join'>  
<value name='ID'>2</value>
```

```

<value name='lookup'>lookup string #2</value>
<value name='matchID'>2</value>
</event>
<event opcode='insert' window='pr_01/cq_01/join'>
<value name='ID'>3</value>
  <value name='lookup'>lookup string #9999999</value>
  <value name='matchID'>9999999</value>
</event>

```

Now suppose you stop the project. You run a new one identical to the previous one except that it uses `pi_HLEVELDB_NC` in the join's lookup index.

```
<join type="leftouter" no-regenerates='true' right-index='pi_HLEVELDB_NC'>
```

You process a few lookup side maintenance events.

```

p,n,14, NEW lookup string #14
u,n,499999999, NEW lookup string #499,999,999 p,n, 4999999, NEW lookup string
#4,999,999
d,n, 99999999

```

Then you process a few Insert events to verify that the lookups are performed correctly.

```

i,n,1,1
i,n,2,2
i,n,3, 4999999
i,n,4, 99999999
i,n,5,14 i,n,6, 3333333
i,n,7,499999999

```

The newly joined results are as follows:

```

<event opcode='insert' window='pr_01/cq_01/join'>
<value name='ID'>1</value>
<value name='lookup'>lookup string #1</value>
<value name='matchID'>1</value>
</event>
<event opcode='insert' window='pr_01/cq_01/join'>
<value name='ID'>2</value>
<value name='lookup'>lookup string #2</value>
<value name='matchID'>2</value>
</event>
<event opcode='insert' window='pr_01/cq_01/join'>
<value name='ID'>3</value>
<value name='lookup'>NEW lookup string #4</value>
<value name='matchID'>4999999</value>
</event>

```

```

<event opcode='insert' window='pr_01/cq_01/join'>
<value name='ID'>4</value>
<value name='matchID'>99999999</value>
</event>

<event opcode='insert' window='pr_01/cq_01/join'>
<value name='ID'>5</value>
<value name='lookup'>NEW lookup string #14</value>
<value name='matchID'>14</value>
</event>

<event opcode='insert' window='pr_01/cq_01/join'>
<value name='ID'>6</value>
<value name='lookup'>lookup string #3333333</value>
<value name='matchID'>3333333</value>
</event>

<event opcode='insert' window='pr_01/cq_01/join'>
<value name='ID'>7</value>
<value name='lookup'>NEW lookup string #499</value>
<value name='matchID'>499999999</value>
</event>

```

### 3.30.4 Non-Stateful Primary Index

Any window that uses a primary index type `pi_EMPTY` is non-stateful or stateless. It acts as a pass-through for all incoming events. This index does not store events. The following restrictions apply to Source windows that use the empty index type.

- No restrictions apply if the Source window is set to "Insert only."
- If the Source window is not Insert-only, then it must be followed by one of the following:
  - a Copy window with a stateful index
  - a Functional window or a Compute window
- When a source, compute, or Functional window in a linear chain with `pi_EMPTY` indexes start a model, one of the following must be true about the linear chain:
  - It must end with a functional window that converts its events to Insert only, and have the *produces-only-inserts* property set.
  - It must end in a stateful compute, functional, or Copy window that convert *Upserts* to Inserts. Alternatively, it must have updates that can propagate further through the model automatically through the stateful index.

Using empty indices and retention enables you to specify multiple retention policies from common event streams coming in through a Source window. The source window is used as an absorption point and pass-through to the copy windows, as shown in the following figure.

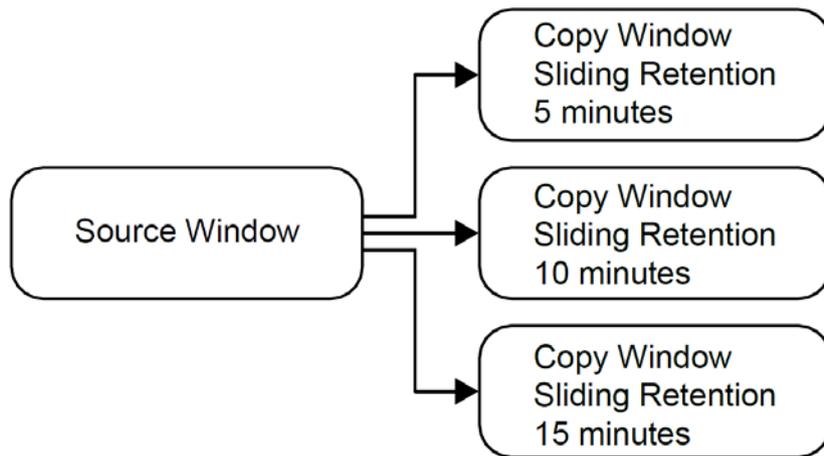


Figure 3-10 Copy Windows

### 3.30.5 Using a Stateful Local Join Index to Resolve the State

In the past, a streaming join lookup where the lookup table required maintenance through publishing Insert, Update, and Delete events required a large amount of memory. This lookup required a stateful Source window for the lookup side of the join. That stateful Source window fully resolved Insert, Update, and Delete events and fed them to the join. The Insert, Update, and Delete events were applied to the local lookup index in the join. A considerable amount of memory was wasted because of the dual stateful index maintenance (the Source window and the Join window's local lookup index).

Beginning with Edge Streaming Analytics 5.2, the local stateful index for the lookup side of the join can resolve Insert, Update, Upsert, and Delete events. This enables the windows that feed into the lookup side of the join to be `pi_EMPTY` but still contain Insert, Update, Upsert, and Delete events for lookup maintenance.

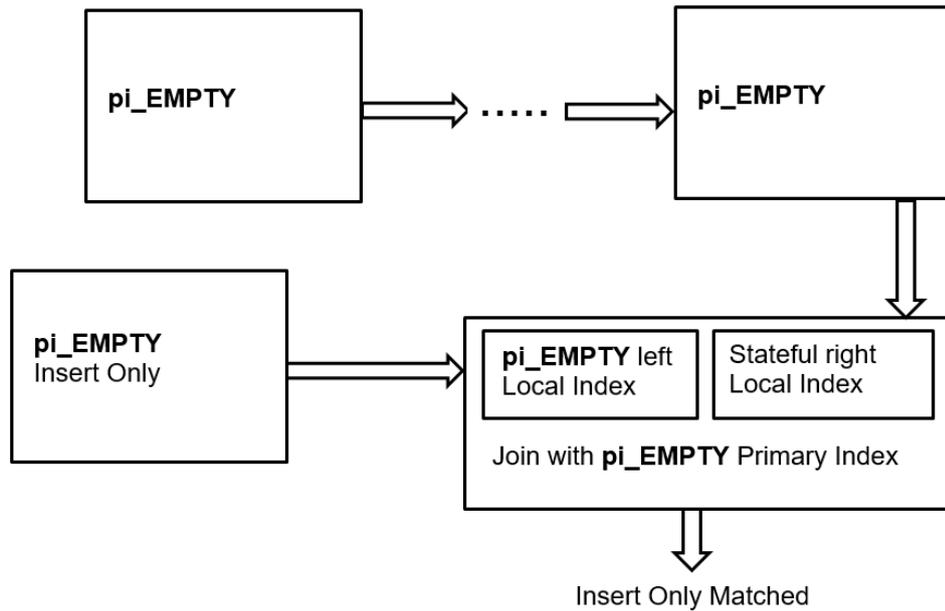


Figure 3-11 Using a Stateful Index for a Left Outer Join

The following code shows a low memory join:

**Note**

You must maintain a finite number of events to ensure a bounded memory footprint. The local lookup index for the join maintains all events (minus deleted events) that stream into the lookup side of the join.

Here are the Source windows:

```

<property name='fstype'>csv</property>
<property name='fname'>input/stream-1.csv</property>
<property name='transactional'>>true</property>
<property name='blocksize'>1</property>
<property name="dateformat">%Y-%m-%d %H:%M:%S</property>
</properties>
</connector>
</connectors>
</window-source>

<property name='fstype'>csv</property>
<property name='fname'>join-out.csv</property>
<property name='snapshot'>>true</property>
<property name="dateformat">%Y-%m-%d %H:%M:%S</property>
</properties>
</connector>
</connectors>
</window-join>
</windows>
  
```

```
<edges>
<edge source="lookup_source" target="join" role="right" />
<edge source="stream_source" target="join" role="left" />
</edges>
</contquery>
</contqueries>
```

Here are the connector groups and edges:

```
<project-connectors>
<connector-groups>
<connector-group name="group1">
<connector-entry connector='cq_01/join/sub' state='running' />
<connector-entry connector='cq_01/lookup_source/pub' state='finished' />
</connector-group>
<connector-group name="group2">
<connector-entry connector='cq_01/stream_source/pub' state='finished' />
</connector-group>
<connector-group name="group3">
<connector-entry connector='cq_01/lookup_source/publ' state='finished' />
</connector-group>
<connector-group name="group4">
<connector-entry connector='cq_01/stream_source/publ' state='finished' />
</connector-group>
</connector-groups>
<edges>
<edge source='group1' target='group2' />
<edge source='group2' target='group3' />
<edge source='group3' target='group4' />
</edges>
</project-connectors>
</project>
```

Four input data files are orchestrated in the following way:

1. The first file contains data that is fed to the lookup side of the join.

```
i,n,0, lookup string #0  
i,n,1, lookup string #1  
i,n,2, lookup string #2  
i,n,3, lookup string #3  
i,n,4, lookup string #4  
i,n,5, lookup string #5  
i,n,6, lookup string #6  
i,n,7, lookup string #7  
i,n,8, lookup string #8  
i,n,9, lookup string #9
```

2. The second file contains streaming data that is fed to the streaming side of the join, and join output is produced.

```
i,n,1,1  
i,n,2,2  
i,n,3,9
```

Here is the initial output of the join:

```
I,N, 1,1,lookup string #1  
I,N, 2,2,lookup string #2  
I,N, 3,9,lookup string #9
```

3. The third file contains a second set of data that is fed to the lookup side of the join. Join maintenance occurs; Inserts, Updates, Upserts, and Deletes are performed on the lookup table.

```
u,n,0, NEW lookup string #0
d,n,5,
p,n,9, NEW lookup string #9
d,n,17
```

4. The fourth file contains a second set of streaming data that is fed to the streaming side of the join. The lookups reflect the previously executed join maintenance.

```
i,n,1,1
i,n,2,2
i,n,3,9
i,n,4,5
i,n,5,0
```

Here is the output of the join after lookup table maintenance has been applied:

```
I,N, 1,1,lookup string #1
I,N, 2,2,lookup string #2
I,N, 3,9,NEW lookup string #9
I,N, 4,5, I,N, 5,0,NEW lookup string #0
```

## 3.31 Restrictions on a Window's Primary Index and Input Windows

### 3.31.1 Introduction

When you violate the following restrictions on a window's primary index or on its input windows, the Streaming Server returns a fatal error that is noted in the log. As a result, your model fails to start. This flags improper models before they process data and cause run-time problems.

Table 3- 11 Restrictions on a Window's Primary Index and Input Windows

Window	Index Restriction	Input Window Restriction	Opcodes Output
Aggregate	Use only stateful indexes	Input window cannot have pi_EMPTY index and cannot produce non-Inserts	All
Calculate	None	Input window cannot have pi_EMPTY index and produce non-Inserts	When algorithm='MAS', set produces-only-inserts='true'. Otherwise produce only Inserts when all input windows produce only Inserts
Compute	None	None	Produces only Inserts when the input window produces only Inserts
Copy	Use only stateful indexes; this requires that retention is set. See Note.	None	All
Counter	None	None	Produces only Inserts when the index is pi_EMPTY
Filter	None	Input window cannot be pi_EMPTY and produce non-Inserts	Produces only Inserts when input window produces only Inserts
Functional	None	None	Produces only Inserts when all input windows produce only Inserts
Geofence	None	None	When the event position input window always produces Inserts, only Inserts are produced

3.31 Restrictions on a Window's Primary Index and Input Windows

Window	Index Restriction	Input Window Restriction	Opcodes Output
Join	None	None	When the streaming side of the join produces only Inserts and the join is no-regenerates='true', only Inserts are produced
Model Reader	Use pi_EMPTY exclusively	All input windows must produce only Inserts	Always produces Inserts
Model Supervisor	Use pi_EMPTY exclusively	All input windows must produce only Inserts	Always produces Inserts
Object Tracking	Use pi_EMPTY exclusively	All input windows must produce only Inserts	Always produces Inserts
Pattern	Use pi_EMPTY exclusively	Should receive only Inserts, logs warning on non-Inserts	Always produces Inserts
Procedural	None	None	Set produces-only-inserts='true' when appropriate, which depends on the procedural code that executes within the window.
Remove State	Use pi_EMPTY exclusively	None	Always produces Inserts
Score	Use pi_EMPTY exclusively	All input windows must produce only Inserts	Always produces Inserts
Source	None	There are no input windows to a Source window.	Produces only Inserts when set to Insert-only
Text Category	Use pi_EMPTY exclusively	All input windows must produce only Inserts	Always produces Inserts
Text Context	Use pi_EMPTY exclusively	All input windows must produce only Inserts	Always produces Inserts
Text Sentiment	Use pi_EMPTY exclusively	All input windows must produce only Inserts	Always produces Inserts
Text Topic	Use pi_EMPTY exclusively	All input windows must produce only Inserts	Always produces Inserts

3.31 Restrictions on a Window's Primary Index and Input Windows

Window	Index Restriction	Input Window Restriction	Opcodes Output
Train	Use pi_EMPTY exclusively	All input windows must produce only Inserts	Always produces Inserts
Transpose	Use pi_EMPTY exclusively	All input windows must produce only Inserts	Always produces Inserts
Union	Depends on input windows and strict setting	Depends on index type	If any input window produces unresolved Updates or Deletes, then the index must be stateful. If all input windows always produce Inserts and the union is strict, then only Inserts are produced; the index can be pi_EMPTY.

---

**Note**

A Copy window that receives only Inserts and that uses (splitter expressions) can use a pi\_EMPTY index.

---

## 3.32 Understanding Design Patterns

### 3.32.1 Overview to Design Patterns

A design pattern is a reusable solution to a common problem within a specific context of software design. The combinations of windows that you use in your design pattern should enable fast and efficient event stream processing. Event stream processing models can be stateless, stateful, or mixed. The type of model that you choose affects how you design it. One challenge when you design a mixed model is to identify sections that must be stateful and those that can be stateless, and then connecting them properly. A stateless model is one where the indexes on all windows have the type `pi_EMPTY`. Events are not retained in any window, and are essentially transformed and passed through. Stateless models exhibit fast performance and use very little memory. They are well-suited to tasks where the inputs are inserts and when simple filtering, computation, text context analysis, or pattern matching are the only operations you require. A stateful model is one that uses windows with index types that store data, usually `pi_RBTREE` or `pi_HASH`. These models can fully process events with Insert, Update, or Delete opcodes. A stateful model facilitates complex relational operations such as joins and aggregations. Because events are retained in indexes, whenever all events are Inserts only, windows grow unbounded in memory. Thus, stateful models must process a mix of Inserts, Updates, and Deletes in order to remain bounded in memory. The mix of opcodes can occur in one of two ways:

- The data source and input events have bounded key cardinality. That is, there are a fixed number of unique keys (such as customer IDs) in the input stream. You can make many updates to these keys provided that the key cardinality is finite.
- A retention policy is enforced for the data flowing in, where the amount of data is limited by time or event count. The data is then automatically deleted from the system by the generation of internal retention delete events. A mixed model has stateless and stateful parts. Often it is possible to separate the parts into a stateless front end and a stateful back end.

A mixed model has stateless and stateful parts. Often it is possible to separate the parts into a stateless front end and a stateful back end.

### 3.32.2 Design Pattern That Links a Stateless Model with a Stateful Model

To control memory growth in a mixed model, link the stateless and stateful parts with copy windows that enforce retention policies. Use this design pattern when you have insert-only data that can be pre-processed in a stateless way. Pre-process the data before you flow it into a section of the model that requires stateful processing (using joins, aggregations, or both).

For example, consider the following model:

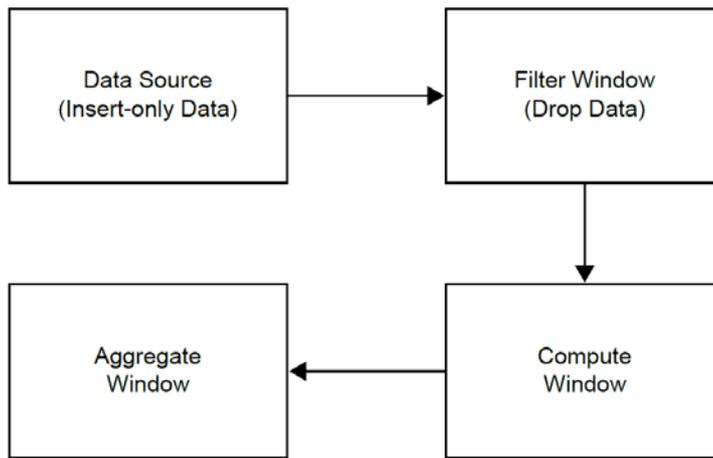


Figure 3-12 Event Stream Processing Model with Insert-Only Data

Here the data source is purely through Inserts. Therefore, the model can be made stateless by using an index type of `pi_EMPTY`. The filter receives inserts from the source, and drops some of them based on the filter criteria, so it produces a set of inserts as output. Thus, the filter can be made stateless also by using an index type of `pi_EMPTY`. The Compute window transforms the incoming inserts by selecting some of the output fields of the input events. The same window computes other fields based on values of the input event. It generates only inserts, so it can be stateless. After the Compute window, there is an Aggregate window. This window type needs to retain events. Aggregate windows group data and compress groups into single events. If an Aggregate window is fed a stream of Inserts, it would grow in an unbounded way. To control this growth, you can connect the two sections of the model with a Copy window with a retention policy.

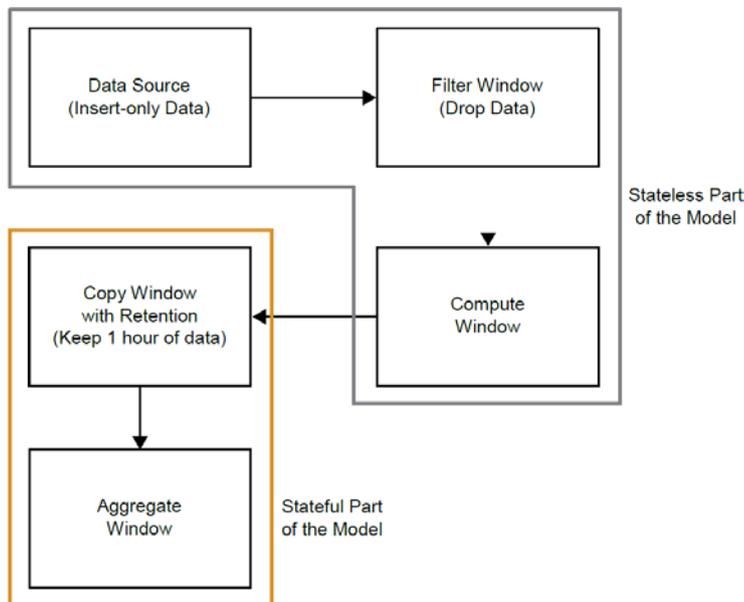


Figure 3-13 Modified Event Stream Processing Model with Stateless and Stateful Parts

The stateful part of the model is accurately computed, based on the rolling window of input data. This model is bounded in memory.

### 3.32.3 Controlling Pattern Window Matches

Pattern matches that are generated by Pattern windows are Inserts. Suppose you have a source window feeding a Pattern window. Because a Pattern window generate Inserts only, you must make it stateless by specifying an index type of `pi_EMPTY`. This prevents the Pattern window from growing infinitely. Normally, you want to keep some of the more recent pattern matches around. Because you do not know how frequent the pattern generates matches, follow the pattern window with a count-based Copy window. Suppose you specify to retain the last 1000 pattern matches in the Copy window.



Figure 3-14 Event Stream Processing Model with Copy with Retention

In cases like these, it is more likely that the Copy window is queried from the outside using adapters, or publish/ subscribe clients. The Copy window might also feed other sections of the model.

### 3.32.4 Augmenting Incoming Events with Rolling Statistics

Suppose you have an insert stream of events, and one or more values are associated with the events. You want to augment each input event with some rolling statistics and then produce the augmented events. Solving this problem requires using advanced features of the modeling environment.

For example, suppose you have a stream of stock trades coming in and you want to augment them with the average stock price in the past. You build the following model.

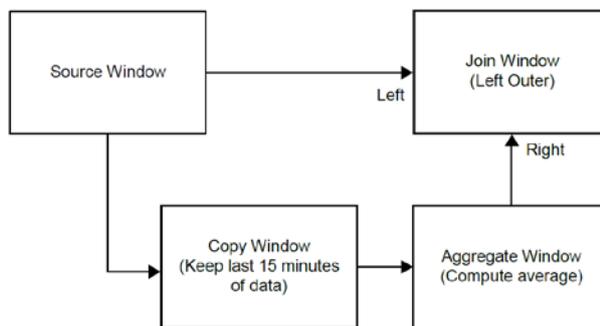


Figure 3-15 Event Stream Processing Model Using Advanced Features

To control the aggregate window:

- Put retention before it (the Copy window).
- Group it by symbol (which is bounded), and use the additive aggregation function `average (ESP_aAve)`, which does not need to store each event for its computation.

The Join window can be problematic. Ordinarily you think of a Join window as stateful. A join retains data for its fact window or dimension window and performs matches. In this case, you want a special, but frequently occurring behavior. When input comes in, pause it at the join until the aggregate corresponding to the input is processed. Then link the two together, and pass the augmented insert out.

To process input in this way:

1. Make the join a left outer join, with the source feed the left window, and the aggregate feeds the right window.
2. Set Tagged Token data flow model on for the projects. This turns on a special feature that causes a fork of a single event to wait for both sides of the fork to rejoin before generating an output event.
3. Set the index type of the join to **pi\_EMPTY**, making the join stateless. A stateless left outer join does not create a local copy of the left driving window (FACT window). It does not keep any stored results of the join. However, there is always a reference-counted copy the lookup window. In the case of a left outer join, this is the right window. The lookup window is controlled by retention in this case, so it is bounded.
4. Ordinarily, a join, when the dimension window is changed, tries to find all matching events in the fact window and then issue updates for those joined matches. You do not want this behavior, because you are matching events in lock step. Further, it is simply not possible because you do not store the fact data. To prevent this regeneration on each dimension window change, set the **no-regenerates** option on the Join window.

In this way you create a fast, lightweight join. This join stores only the lookup side, and produces a stream of inserts on each inserted fact event. A stateless join is possible for left and right outer joins. The following XML code implements this model.

```
<engine port='52525' dateformat='%d/%b/%Y:%H:%M:%S'>
<projects>
<project name='trades_proj' pubsub='auto'
use-tagged-token='true' threads='4'>
<contqueries>
<contquery name='trades_cq'>

<windows>
<window-source name='Trades'
insert-only='true' index='pi_EMPTY'>
<schema>
<fields>
<field name='tradeID' type='string' key='true' />
<field name='security' type='string' />
<field name='quantity' type='int32' />
<field name='price' type='double' />
<field name='traderID' type='int64' />
<field name='time' type='stamp' />
</fields>
</schema>
</window-source>

<window-copy name='TotalIn'>
<retention type='bycount_sliding'>5</retention>
</window-copy>
```

```
<window-aggregate name='RunTotal'>
  <schema>
    <fields>
      <field name='tradeID' type='string' />
      <field name='security' type='string' key='true' />
      <field name='quantityTotal' type='double' />
    </fields>
  </schema>
  <output>
    <field-expr>ESP_aLast(tradeID)</field-expr>
    <field-expr>ESP_aSum(quantity)</field-expr>
  </output>
</window-aggregate>

source='l_quantity' />
<field-selection name='price' source='l_price' />
<field-selection name='traderID' source='l_traderID' />
<field-selection name='time' source='l_time' />
<field-selection name='quantityTotal' source='r_quantityTotal' />
</output>
</window-join>
</windows>

<edges>
  <edge source='Trades' target='JoinTotal' />
  <edge source='Trades' target='TotalIn' />
  <edge source='TotalIn' target='RunTotal' />
  <edge source='RunTotal' target='JoinTotal' />
</edges>

</contquery>
</contqueries>
</project>
</projects>
</engine>
```

## 3.33 Advanced Window Operations

### 3.33.1 Writing Aggregate Functions to Embed in Applications

#### Overview

The most commonly used aggregate functions are one parameter functions with an input schema field name (for example, the aggregation function *ESP\_aMax(fieldname)*). For field names in the input schema, the field index into the input event is passed into the aggregate function, not the value of the field. This is important when you deal with groups. You might need to iterate over all events in the group and extract the values by event index from each input event.

After you write an aggregate function, embed it in C++ code in order to use it in your event stream processing application. Copy your function into **\$DFESP\_HOME/examples/cxx/aggregate\_userdef/src/ functions.cpp**. Suppose your function is named **My\_Aggregation\_Function**. At the bottom of **functions.cpp**, create a wrapper function for your aggregation function.

```
// the uMyFunction wrapper:
// every aggregation function must be wrapped like this.
// int dfESPaggrfunc_uMyFunctionWrapper(dfESPExpEngine::exp_engine_t *e,
//   dfESPExpEngine::exp_sym_value_t *returnval,
//   int parmcount, dfESP_EXPsym_value_t **parms)
// {
//   return dfESPaggrfunc_Wrapper((void *)my_aggreration_function, e,
//   returnval, parmcount, parms);
// }
```

Create an entry in the user-defined function list for the wrapper function.

```
// Get all user-defined aggregation functions during initialization
//
void add_user_aggrFunctions() {
dfESPengine *e = dfESPengine::getEngine();
dfESPptrList<aggr_function_t *> &uFuncs = e->getUDAFs();
// push back as many user defined functions as you like:
// the parameters are: <callable name>, <function pointer>,
// <num args>, <additive flag>, <additive flag for insert only>,
// <description>
uFuncs.push_back(new aggr_function_t("USER_uSum_add",
(void *)dfESPaggrfunc_uMyFunctionWrapper, "a", true,
true, "description"));
}
```

Adjust the number of arguments, additive flags, and the description field accordingly. The sample code **\$DFESP\_HOME/examples/cxx/aggregate\_userdef/src/functions.cpp** provides two complete examples. The makefile distributed with the sample code produces a shared library in the **aggregate\_userdef/plugins** directory. Copy this plug-in to **\$DFESP\_HOME/lib** and name it **libdfxesp\_udafD-major.minor**.

### 3.33.2 Writing Additive Aggregate Functions

Aggregate functions that compute themselves based on previous field state and a new field value are called additive aggregation functions. These functions provide computational advantages over aggregate functions.

An additive aggregate function can be complex for two reasons:

- They must look at the current state (for example, the last computed state).
- They must evaluate the type of incoming event to make proper adjustments.

Suppose that you keep the state of the last value of the group's summation of a field. When a new event arrives, you can conditionally adjust the state base on whether the incoming event is an Insert, Delete, or Update. For an Insert event, you simply increase the state by the new value. For a Delete, you decrease the state by the deleted value. For an Update, you increase and decrease by the new and old values respectively. Now the function never has to loop through all group values. It can determine the new sum based on the previous state and the latest event that affects the group.

The following code performs these basic steps:

1. Look at the input and output types and verify compatibility.
2. Initialize a return variable of the specified output type.
3. Determine whether the function has been called before. That is, is there a previous state value?
  - If so, retrieve it for use.
  - If not, create a new group with an arriving insert so that you can set the state to the incoming value.
4. Switch on the opcode and adjust the state value.
5. Check for computational errors and return the error value or the state value as the result.

```
// an additive summation function
//
// vgs is the groupstate object passed as a (void *) pointer
// fid is the filed ID in internal field order of the field on
// which we sum.
dfESPdatavarPtr uSum_add(void *vgs, dfESPExpEngine::exp_sym_value_t *fid) {
    dfESPdatavar *rdv;
    // placeholder for return value dfESPgroupstate *gs = (dfESPgroupstate *)vgs;
    // the passed groupstate cast back to dfESPgroupstate object.

    // get the 1) aggregate schema (output schema)
    // and 2) the schema of input events
    //
    dfESPschema *aSchema = gs->getAggregateSchema(); dfESPschema *iSchema = gs-
    >getInputSchema();

    // get the type of 1) the field we are computing in the aggregate schema and
    // 2) the input field we are summing.
    //
    dfESPdatavar::dfESPdatatype aType = aSchema->getTypeEO(gs->getOperField());
```

```

bool te=false;
int64_t fID = exp_param_getI64(fid, te); if (te) {
cerr << "could not obtain the integer field offset (field ID)" << endl; rdv = new
dfESPdatavar(aType); rdv->null();
return rdv;
}
dfESPdatavar::dfESPdatatype iType = iSchema->getTypeIO(fID); dvn_error_t retCode =
dvn_noError;
// return code for using the datavar numerics package.

// If the input fields or the output field is non-numeric,
// flag an error.
//
if ( (!isNumeric(aType)) || (!isNumeric(iType)) ) {
cerr << "summation must work on numeric input, produce numeric output."
<< endl; return NULL;
}
}

// fetch the input event from the groupstate object (nev)
// and, in the case of an update, the old event that
// is being updated (oev)
//
dfESPEventPtr nev = gs->getNewEvent(); dfESPEventPtr oev = gs->getOldEvent();

// Get the new value out of the input record
//
dfESPdatavar iNdv(iType);
// a place to hold the input variable. dfESPdatavar iOdv(iType);
// a place to hold the input variable (old in upd case). nev->copyByIntID(fID, iNdv);
// extract input value (no copy) to it (from new record)

// Get the old value out of the input record (update)
//
if (oev) {
oev->copyByIntID(fID, iOdv);
// extract input value to it (old record)
}

// Note: getStateVector() returns a reference to the state vector for
// the field we are computing inside the group state object.
//
dfESPptrVect<dfESPdatavarPtr> &state = gs->getStateVector();

// create the datavar to return, of the output type and set to zero.
//
rdv = new dfESPdatavar(aType); // NULL by default. rdv->makeZero();

// If the state has never been set, we set it and return.
//
if (state.empty()) { dv_assign(rdv, &iNdv);
// result = input
state.push_back(new dfESPdatavar(rdv));
// make a copy and push as state return rdv;
}

```

```
// at this point we have a state,  
// so lets see how we should adjust it based on opcode.  
//  
break;  
case dfESPEventcodes::eo_DELETE: if (!iNdv.isNull())  
retCode = dv_subtract(state[0], state[0], &iNdv); break;  
case dfESPEventcodes::eo_UPDATEBLOCK: retCode = dv_compare(c, &iNdv, &iOdv); if  
(retCode != dvn_noError) break;  
if (c == 0) // the field value did not change. break;  
if (!iNdv.isNull())  
// add in the update value  
retCode = dv_add(state[0], state[0], &iNdv); if (retCode != dvn_noError) break;  
if (!iOdv.isNull())  
// subtract out the old value  
retCode = dv_subtract(state[0], state[0], &iOdv); break;  
default:  
cerr << "got a bad opcode when running uSum_add()" << endl; badOpcode = true;  
}  
  
if ( badOpcode || (retCode != dvn_noError) ) { rdv->null();  
// return a null value.  
cerr << "uSum() got an arithmetic error summing up values" << endl;  
} else  
dv_assign(rdv, state[0]);  
// return the adjusted state value  
  
return rdv;  
}
```

You can use the *Sum()* aggregate function to iterate over the group and compute a new sum when a new group changes. Faster results are obtained when you maintain the *Sum()* in a *dfESPdatavar* in the *dfESPgroupstate* object and increment or decrement the object by the incoming value, provided the new event is an Insert, Update, or Delete. The function then adjusts this field state so that it is up-to-date and can be used again when another change to the group occurs.

### 3.33.3 Writing Non-Additive Aggregate Functions

You can write an aggregate sum function that does not maintain state and is not additive. The function iterates through each event in a group to aggregate. It requires the aggregation window to maintain a copy of every input event for all groups.

The following code performs these basic steps:

1. Look at the input and output types and verify compatibility.
2. Initialize a return variable of the specified output type.
3. Loop across all events in the group and perform the aggregation function.
4. Check for computational errors and return the error or the result .

```

// fid is the filed ID in internal field order of the field on
// which we sum.
dfESPdatavarPtr uSum_nadd(void *vgs, dfESPexpEngine::exp_sym_value_t *fid) {
dfESPdatavar *rdv;
// placeholder for return value dfESPgroupstate *gs = (dfESPgroupstate *)vgs;
// the passed groupstate cast back to dfESPgroupstate object.

// get the 1) aggregate schema (output schema)
// and 2) the schema of input events
//
dfESPschema *aSchema = gs->getAggregateSchema(); dfESPschema *iSchema = gs-
>getInputSchema();

// get the type of 1) the field we are computing in the aggregate schema and
// 2) the input field we are summing.
//
dfESPdatavar::dfESPdatatype aType = aSchema->getTypeEO(gs->getOperField());

bool te=false;
int64_t fID = exp_param_getI64(fid, te); if (te) {
cerr << "could not obtain the integer field offset (field ID)" << endl; rdv = new
dfESPdatavar(aType); rdv->null();
return rdv;
}

dfESPdatavar::dfESPdatatype iType = iSchema->getTypeIO(fID); dvn_error_t retCode =
dvn_noError;
// return code for using the datavar numerics package.

// If the input fields or the output field is non-numeric,
// flag an error.
//
if ( (!isNumeric(aType)) || (!isNumeric(iType)) ) {
cerr << "summation must work on numeric input, produce numeric output."
<< endl; return NULL;
}

// in the ESP type system, INT32 < INT64 < DOUBLE < DECSECT.
// This checks compatibility. The output type must be greater
// equal the input type. i.e. we cannot sum a column of int64
// and puit them into an int32 variable.
//
if (iType > aType) {
cerr << "output type is not precise enough for input type" << endl; return NULL;
}

```

### 3.33.4 Implementing Periodic (or Pulsed) Window Output

In most cases, the Edge Streaming Analytics API is fully event driven. That is, windows continuously produce output as soon as they transform input. But there might be times when you want a window to hold data and then write a canonical batch of updates. In this case, operations to common key values are collapsed into a single operation.

Here are two cases where batched output might be useful:

- Visualization clients might want to get updates once a second because they cannot visualize changes any faster than this. When the event data is pulsed, the clients take advantage of the reduction of event data to visualize through the collapse around common key values.
- A window that follows the pulsed window is interested in comparing the deltas between periodic snapshots from that window.

Use the following call to add output pulsing to a window:

```
dfESPwindow::setPulseInterval(size_t us);
```

---

#### Note

Periodicity is specified in microseconds. However, given the clock resolution of most non-real-time operating systems, the minimum value that you should specify for a pulse period is 100 milliseconds. In your XML code, use the pulse-interval attribute of the window. The value defaults to milliseconds.

---

### 3.33.5 Splitting Generated Events across Output Slots

#### Overview

All window types can register a splitter function or expression to determine what output slot or slots should be used for a newly generated event. This enables you to send generated events across a set of output slots. Most windows send all generated events out of output slot 0 to zero or more downstream windows. For this reason, it is not standard for most models to use splitters. Using window splitters can be more efficient than using Filter windows off a single output slot. This is especially true, for example, when you are performing an alpha-split across a set of trades or a similar task. When adding edges between a window with a splitter function and downstream windows, specify the slot number of the parent window where the downstream window receives its input events. If the slot number is -1, the downstream window receives all the data produced by the parent window regardless of the splitter function. Using window splitters is more efficient than using two or more subsequent Filter windows. This is because the filtering is performed a single time at the window splitter rather than multiple times for each filter. This results in less data movement and processing.

#### Using Splitter Functions in XML

Here is an example with one Source window, one Compute window, and three Copy windows. The Compute window includes a user-defined function as a splitter to determine what slot an event should go to. Each slot directs to a different Copy window.

Here is the Source window:

```
<window-source name='sourceWindow' index='pi_RBTREE'>
<schema>
<fields>
<field name='ID' type='int32' key='true' />
<field name='symbol' type='string' />
<field name='price' type='double' />
</fields>
</schema>
<connectors>
<connector class='fs'>
<properties>
<property name='type'>pub</property>
<property name='fstype'>csv</property>
<property name='fname'>input.csv</property>
<property name='blocksize'>1</property>
<property name='transactional'>>true</property>
</properties>
</connector>
</connectors>
</window-source>
```

The Source window uses a file and socket publisher connector to receive input events from a CSV file named input.csv.

```
l n 1 IB 121.48
      M
      2 1593.41
i n  A
      M
      ZN
      2 1588.2
u n  A
      M
      ZN
```

```
      3  179.55
p n  AP
    PL
      3  198
p n  AP
    PL
      5  1094.58
i n  G
    O
    O
    GL
      1
d n  IB 120.5
    M
      1 FB
i n 1  138.68
      1 FB
u n 1  137.5
```

It streams those events to a computer window:

```
<edge source='sourceWindow' target='computeWindow' />
```

The Compute window includes a user-defined function as a splitter. The function calculates a slot number to determine what Copy window should receive the incoming event:

```
<window-compute name='computeWindow' collapse-updates='true'>
  <splitter-expr>
  <expr-initialize>
  <udfs>
  <udf name='udf1' type='int32'>
  <![CDATA[private integer p
    p = parameter(1);
return p%2]]>
  </udf>
  </udfs>
  </expr-initialize>
  <expression>udf1(ID)</expression>
  </splitter-expr>
  <expr-initialize>
  <initializer type='int32'>
  <![CDATA[integer counter
counter=0]]>
  </initializer>
  </expr-initialize>
  <schema>
  <fields>
  <field name='ID' type='int32' key='true' />
  <field name='counter' type='int32' />
```

```
<field name='symbol' type='string' />
<field name='price' type='double' />
</fields>
</schema>
<output>
<field-expr>counter=counter+1 return counter</field-expr>
<field-expr>symbol</field-expr>
<field-expr>price</field-expr>
</output>
</window-compute>
```

Here are the edges between the Compute and Copy windows, specifying the slot numbers of the Compute window where the Copy windows receive their input events.

```
<edge source='computeWindow' slot='0' target='computeWindowSlot_01' />
<edge source='computeWindow' slot='1' target='computeWindowSlot_02' />
<edge source='computeWindow' slot='-1' target='computeWindowSlot_03' />
<window-copy name='computeWindowSlot_01' />
<window-copy name='computeWindowSlot_02' />
<window-copy name='computeWindowSlot_03' />
```

After streaming the incoming events, here are the events processed by computeWindowSlot\_01:

```
I,N, 2,2,AMZN,1593.410000
UB,N, 2,3,AMZN,1588.200000
D,N, 2,2,AMZN,1593.410000
```

Here are the events processed by computeWindowSlot\_02:

```
I,N, 1,1,IBM,121.480000
I,N, 3,4,APPL,179.550000
UB,N, 3,5,APPL,198.000000
D,N, 3,4,APPL,179.550000
I,N, 5,6,GOOGL,1094.580000
D,N, 1,1,IBM,121.480000
I,N, 11,7,FB,138.680000
UB,N, 11,8,FB,137.500000
D,N, 11,7,FB,138.680000
```

Here are the events processed by computeWindowSlot\_03:

```
I,N, 1,1,IBM,121.480000
I,N, 2,2,AMZN,1593.410000
UB,N, 2,3,AMZN,1588.200000
D,N, 2,2,AMZN,1593.410000
```

```
I,N, 3,4,APPL,179.550000
UB,N, 3,5,APPL,198.000000
D,N, 3,4,APPL,179.550000
I,N, 5,6,GOOGL,1094.580000
D,N, 1,1,IBM,121.480000
I,N, 11,7,FB,138.680000
UB,N, 11,8,FB,137.500000
D,N, 11,7,FB,138.680000
```

### Splitter Functions in C++

Here is a prototype for a C++ splitter function.

```
size_t splitterFunction(dfESPschema *outputSchema, dfESPEventPtr nev, dfESPEventPtr
oev);
```

This splitter function receives the schema of the events supplied, the new and old event (only non-null for update block), and it returns a slot number.

Here is how you use the splitter for the Source window (*sw\_01*) to split events across three Copy windows:

```
sw_01, cw_02, cw_03.
sw_01->setSplitter(splitterFunction); cq_01->addEdge(sw_01, 0, cw_01);
cq_01->addEdge(sw_01, 1, cw_02);
cq_01->addEdge(sw_01, -1, cw_03);
```

The *dfESPwindow::setSplitter()* member function is used to set the user-defined splitter function for the Source window. The *dfESPcontquery::addEdge()* member function is used to connect the Copy windows to different output slots of the Source window. When no splitter function is registered with the parent window, the slots specified are ignored, and each child window receives all events produced by the parent window.

---

### Note

Do not write a splitter function that randomly distributes incoming records. Also, do not write a splitter function that relies on a field in the event that might change. The change might cause the updated event to generate a different slot value than what was produced prior to the update. This can cause an Insert to follow one path and a subsequent Update to follow a different path. This generates inconsistent results, and creates indices in the window that are not valid.

---

### Splitter Expressions in C++

When you define splitter expressions, you do not need to write the function to determine and return the desired slot number. Instead, the registered expression does this using the splitter expression engine. Applying expressions to the previous example would look as follows, assuming that you split on the field name "splitField", which is an integer:

```
sw_01->setSplitter("splitField%2"); cq_01->addEdge(sw_01, 0, cw_01);
```



values in a single event block with transactional properties. This attempt fails and is logged because transactional event blocks are treated atomically. All operations in that block are checked against an existing window state before the transactional block is applied as a whole.

### Publishing Partial Events into a Source Window

Consider these three points when you publish partial events into a Source window.

- In order to construct the partial event, you must represent all the fields in the event. Specify either the field type and value or a placeholder field that indicates that the field value and type are missing. In this way, the existing field value for this key field combination remains for the updated event. These field values and types can be provided as datavars to build the event. Alternatively, they can be provided as a comma-separated value (CSV) string.
- If you use CSV strings, then use '^U' (such as, control-U, decimal value 21) to specify that the field is a placeholder field and should not be updated. On the other hand, if you use datavars to represent individual fields, then those fully specified fields should be valid. Enter them as datavars with values (non-null or null). Specify the placeholder fields as empty datavars of type *dfESPdatavar::ESP\_LOOKUP*.
- No matter what form you use to represent the field values and types, the representation should be included in a call for the partial update to be published. In addition to the fields, use a flag to indicate whether the record is a normal or partial update. If you specify partial update, then the event must be an Update or an Upsert that is resolved to an Update. Using partial-update fields makes sense only in the context of updating an existing or retained Source window event. This is why the opcode for the event must resolve to Update. If it does not resolve to Update, an event merge error is generated. If you use an event constructor to generate this binary event from a CSV string, then the beginning of that CSV string contains "u,p" to show that this is a partial-update. If instead, you use *event->buildEvent()* to create this partial update event, then you need to specify the event flag parameter as *dfESPeventcodes::ef\_PARTIALUPDATE* and the event opcode parameter as *dfESPeventcodes::eo\_UPDATE*.
- One or more events are pushed onto a vector and then that vector is used to create the event block. The event block is then published into a Source window. For performance reasons, each event block usually contains more than a single event. When you create the event block, you must specify the type of event block as transactional or atomic using *dfESPeventblock::ebt\_TRANS* or as normal using *dfESPeventblock::ebt\_NORMAL*.

Do not use transactional blocks with partial updates. Such usage treats all events in the event block as atomic. If the original Insert for the event is in the same event block as a partial Update, then it fails. The events in the event block are resolved against the window index before the event block is applied atomically. Use normal event blocks when you perform partial Updates

### Examples

Here are some sample code fragments for the variations on the three points described in the previous section. Create a partial Update *datavar* and push it onto the *datavar* vector.

```
// Create an empty partial-update datavar.  
dfESPdatavar* dvp = new dfESPdatavar(dfESPdatavar::ESP_LOOKUP);  
  
// Push partial-update datavar onto the vector in the appropriate  
// location.
```

```
// Other partial-update datavars might also be allocated and pushed to the
// vector of datavars as required. dvVECT.push_back(dvp); // this would be done for
// each field in the update event
```

Create a partial Update using partial-update and normal datavars pushed onto that vector.

```
// Using the datavar vector partially defined above and schema,
// create event.
```

```
dfESPEventPtr eventPtr = new dfESPEvent();

eventPtr->buildEvent(schemaPtr, dvVECT, dfESPEventcodes::eo_UPDATE,
dfESPEventcodes::ef_PARTIALUPDATE);
```

Define a partial update event using CSV fields where '^U' values represent partial-update fields. Here you are explicitly showing '^U'. However, in actual text, you might see the character representation of Ctrl-U because

individual editors show control characters in different ways. Here, the event is an Update (due to 'u'), which is partial-update (due to 'p'), key value is 44001, "ibm" is the instrument that did not change. The instrument is included in the field. The price is 100.23, which might have changed, and 3000 is the quantity, which might have changed, so the last three of the fields are not updated.

```
p = new dfESPEvent(schema_01,
(char *) "u,p,44001,ibm,100.23,3000,^U,^U,^U");
```

### 3.33.7 Implementing Persist and Restore Operations

Edge Streaming Analytics enables you to do the following:

- persist a complete model state to a file system
- restore a model from a persist directory that had been created by a previous persist operation
- persist and restore an entire engine
- persist and restore a project

To create a persist object for a model, provide a pathname to the class constructor: *dfESPersist(char\*baseDir)*; The *baseDir* parameter can point to any valid directory, including disks shared among multiple running event stream processors.

After providing a pathname, call either of these two public methods:

```
bool persist();
bool restore(bool dumpOnly=false);
// dumpOnly = true means do not restore, just walk and print info
```

The *persist()* method can be called at any time. Be aware that it is expensive. Event block injection for all projects is suspended, all projects are quiesced, persist data is gathered and written to disk, and all projects are restored to normal running state.

The *restore()* method should be invoked only before any projects have been started. If the persist directory contains no persist data, the *restore()* call does nothing.

The persist operation is also supported by the C, Java, and Python publish/subscribe APIs. These API functions require a host:port parameter to indicate the target engine.

The C publish/subscribe API method is as follows: *int C\_dfESPpubsubPersistModel(char \*hostportURL, const char \*persistPath)*

The Java publish/subscribe API method is as follows: *boolean persistModel(String hostportURL, String persistPath)*

One application of the persist and restore feature is saving state across event stream processor system maintenance. In this case, the model includes a call to the restore() function described previously before starting any projects. To perform maintenance at a later time on the running engine:

1. Pause all publish clients in a coordinated fashion.
2. Make one client execute the publish/subscribe persist API call described previously.
3. Bring the system down, perform maintenance, and bring the system back up.
4. Restart the event stream processor model, which executes the restore() function and restores all windows to the states that were persisted in step 2.
5. Resume any publishing clients that were paused in step 1.

To persist an entire engine, use the following functions: *bool dfESPengine::persist(const char \*path);*

The path that you specify for persist can be the same as the path that you specify for set\_restorePath. To persist a project, use the following functions:

```
bool dfESPproject::persist(const char *path)
```

```
bool dfESPproject::restore(const char *path);
```

Start an engine and publish data into it before you persist it. It can be active and receiving data when you persist it. To persist an engine, call *dfESPengine::persist(path);*. The system does the following:

1. pauses all incoming messages (suspends publish/subscribe)
2. finish processing any queued data
3. after all queued data has been processed, persist the engine state to the specified directory, creating the directory if required
4. after the engine state is persisted, resume publish/subscribe and enable new data to flow into the engine

To restore an engine, initialize it and call *dfESPengine::set\_restorePath(path);* After the call to *dfESPengine::startProjects()* is made, the entire engine state is restored.

To persist a project call *dfESPproject::persist(path);*. The call turns off publish/subscribe, quiesces the system, persists the project, and then re-enables publish/subscribe. The path specified for restore is usually the same as that for persist.

To restore the project, call `dfESPproject::restore(path)`; before the project is started. Then call `dfESPEngine::startProject(project)`;

---

**Note**

When you persist a model that trains online projects, note that the Train window immediately starts training a new model upon model restore. No user intervention is required. For more information about online training, see( "Overview")

---

**Note**

When you persist a model that performs offline training (whereby the model is published to the Streaming Server through a Model Reader window) and then restore the model, you must republish the model to the Model Reader window to enable the Score window to score new events. For more information about offline training, see ("Overview").

---

### 3.33.8 Gathering and Saving Latency Measurements

The `dfESPLatencyController` class supports gathering and saving latency measurements on an event stream processing model. Latencies are calculated by storing 64-bit microsecond granularity timestamps inside events that flow through windows enabled for latency measurements.

In addition, latency statistics are calculated over fixed-size aggregations of latency measurements. These measurements include average, minimum, maximum, and standard deviation. The aggregation size is a configurable parameter. You can use an instance of the latency controller to measure latencies between any Source window and some downstream window that an injected event flows through.

The latency controller enables you to specify an input file of event blocks and the rate at which those events are injected into the Source window. It buffers the complete input file in memory before injecting to ensure that disk reads do not skew the requested inject rate.

Specify an output text file that contains the measurement data. Each line of this text file contains statistics that pertain to latencies gathered over a bucket of events. The number of events in the bucket is the configured aggregation size. Lines contain statistics for the next bucket of events to flow through the model, and so on. Each line of the output text file consists of three tab-separated columns. From left to right, these columns contain the following:

- the maximum latency in the bucket
- the minimum latency in the bucket
- the average latency in the bucket

You can configure the aggregation size to any value less than the total number of events. A workable value is something large enough to get meaningful averages, yet small enough to get several samples at different times during the run.

If publish/subscribe clients are involved, you can also modify publisher/subscriber code or use the file/socket adapter to include network latencies as well.

To measure latencies inside the model only:

1. Include "*int/dfESPlatencyController.h*" in your model, and add an instance of the *dfESPlatencyController* object to your *main()*.
2. Call the following methods on your *dfESPlatencyController* object to configure it:

Method	Description
<code>void set_playbackRate(int32_t r)</code>	Sets the requested inject rate.
<code>void set_bucketSize(int32_t bs)</code>	Sets the <code>bucketSize</code> parameter previously described.
<code>void set_maxEvents(int32_t me)</code>	Sets the maximum number of events to inject.
<code>void set_oFile(char *ofile)</code>	Sets the name of the output file containing latency statistics.
<code>void set_iFile(char *ifile)</code>	Sets the name of the input file containing binary event block data.
<code>void set_stampBlkSize(int64_t stampsize)</code>	Specifies the block size (in number of events) of memory to allocate for storing timestamps when in latency mode. Additional blocks are allocated as required.
<code>void set_skipSize(int32_t ss)</code>	Specifies the number of beginning and ending aggregation blocks to ignore in latency calculations.

3. Add a subscriber callback to the window where you would like the events to be timestamped with an ending timestamp. Inside the callback add a call to this method on your *dfESPlatencyController* object: `void record_output_events(dfESPeventblock *ob)`. This adds the ending timestamp to all events in the event block.
4. After starting projects, call these methods on your *dfESPlatencyController* object:

Method	Description
<code>void set_injectPoint(dfESPwindow_source *s)</code>	Sets the Source window in which you want events time stamped with a beginning timestamp.
<code>void read_and_buffer()</code>	Reads the input event blocks from the configured input file and buffers them.
<code>void playback_at_rate()</code>	Time stamps input events and injects them into the model at the configured rate, up to the configured number of events.

5. Quiesce the model and call this method on your *dfESPlatencyController* object: `void generate_stats()` and pass 4 and 0 for the *metaHigh* and *metaLow* parameters respectively. This gathers the start and end timestamps from the correct metadata locations in each event and writes the latency statistics to the configured output file.

To measure model and network latencies by modifying your publish/subscribe clients:

1. In the model, call the *dfESPengine setLatencyMode()* function before starting any projects.
2. In your publisher client application, immediately before calling *C\_dfESPpublisherInject()*, call *C\_dfESPLibrary\_getMicroTS()* to get a current timestamp. Loop through all events in the event block and for each one call *C\_dfESPevent\_setMeta(event, 0, timestamp)* to write the timestamp to the event. This records the publish/subscribe inject timestamp to meta location 0.
3. The model inject and subscriber callback timestamps are recorded to meta locations 2 and 3 in all events automatically because latency mode is enabled in the engine.
4. Add code to the inject loop to implement a fixed inject rate. See the latency publish/subscribe client example for sample rate limiting code.
5. In your subscriber client application, include *"int/dfESPlatencyController.h"* and add an instance of the *dfESPlatencyController* object.
6. Configure the latency controller *bucketSize* and *playbackRate* parameters as described previously.
7. Pass your latency controller object as the context to *C\_dfESPsubscriberStart()* so that your subscriber callback has access to the latency controller.
8. Make the subscriber callback pass the latency controller to *C\_dfESPlatencyController\_recordOutputEvents()*, along with the event block. This records the publish/subscribe callback timestamp to meta location 4.
9. When the subscriber client application has received all events, you can generate statistics for latencies between any pair of the four timestamps recorded in each event. First call *C\_dfESPlatencyController\_setOFile()* to set the output file. Then write the statistics to the file by calling *C\_dfESPlatencyController\_generateStats()* and passing the latency controller and the two timestamps of interest. The list of possible timestamp pairs and their time spans are as follows:
  - (0, 2) – from inject by the publisher client to inject by the model
  - (0, 3) – from inject by the publisher client to subscriber callback by the model
  - (0, 4) – from inject by the publisher client to callback by the subscriber client (full path)
  - (2, 3) – from inject by the model to subscriber callback by the model
  - (2, 4) – from inject by the model to callback by the subscriber client
  - (3, 4) – from subscriber callback by the model to callback by the subscriber client
10. To generate further statistics for other pairs of timestamps, reset the output file and call *C\_dfESPlatencyController\_generateStats()* again.

To measure model and network latencies by using the file/socket adapter, run the publisher and subscriber adapters as normal but with these additional switches:

Publisher	
<code>--r <i>rate</i></code>	Specifies the requested transmit rate in events per second.
<code>-m <i>maxevents</i></code>	Specifies the maximum number of events to publish.
<code>-p</code>	Specifies to buffer all events prior to publishing.
<code>-n</code>	Enables latency mode.

Subscriber	
<code>-r <i>rate</i></code>	Specifies the requested transmit rate in events per second.
<code>-a <i>aggrsize</i></code> <code>&lt;<i>aggr_blocks_to_ignore</i>&gt;</code>	Specifies the aggregation bucket size. Can be followed by a comma-separated value that specifies the beginning and ending aggregation blocks to ignore in latency calculations.
<code>-n</code>	Enables latency mode.
<code>-N <i>latencyblksize</i></code>	Specifies the block size (in number of events) of memory to allocate for storing timestamps when in latency mode. Additional blocks are allocated as required.

The subscriber adapter gathers all four timestamps described earlier for the windows specified in the respective publisher and subscriber adapter URLs. At the end of the run, it writes the statistics data to files in the current directory. These files are named "latency\_ **transmit rate** \_high timestamp\_low timestamp", where the high and low timestamps correspond to the timestamp pairs listed earlier.

### 3.33.9 Enabling Finalized Callback

Some data structures are fully created when windows and edges are made, but are finalized just before the project is started. These data structures include derived schema and certain types of window indexes. The finalized callback function is called when all data structures are completely initialized, but before any events start to flow into the window. The finalized callback function can initialize some state or connection information that is required by an application or XML model.

Enable finalized callback as follows:

- Use the (**finalized-callback**) element in XML. Specify the name of the library that contains the window callback function and the name of the function that the window calls.

```
<finalized-callback name='library' function='fin_callback'>□
```

- Use the following function in C++:

```
dfESPwindow::addFinalizeCallback(dfESPwindowCB_func cbf)
```

### 3.34 Functional Window and Notification Window Support Functions

#### 3.34.1 Functions for Event Stream Processing

#### 3.34.2 CONTQUERYNAME

Returns the name of the continuous query that contains the current event.

**contqueryName**( )

```
contqueryName( )=cq_1
contqueryName( )=myquery
```

#### 3.34.3 ENGINEMETADATA

Return the value of the engine metadata item specified by the argument.

**engineMetadata**( *argument* )

Table 3- 12 Arguments for ENGINEMETADATA

Argument	Description
<i>argument</i>	Specifies a string whose value is an engine metadata item (for example, version or model).

#### 3.34.4 ESPCONFIGVALUE

Return the value of the configuration value specified by the argument.

**espConfigValue**( *argument* )

Table 3- 13 Arguments to ESPCONFIGVALUE

Argument	Description
<i>argument</i>	Specifies a configuration value. For more information about configuration values, see "Configuring the Streaming Server".

When the following value is specified in **esp-properties.yaml**:

```
modelvalues:
  magicnumber: 11
```

The function returns the following:

```
espConfigValue('meta.meteringhost')=espsrv01
product(espConfigValue('modelvalues.magicnumber'),10)=110
```

### 3.34.5 EVENTCOUNTER

Return the 0-based number of events generated by a functional window.

**eventCounter()**

```
string($id, '-', eventCounter())=eventid-0
string($id, '-', eventCounter())=eventid-1
string($id, '-', eventCounter())=eventid-2
```

### 3.34.6 EVENTSPROCESSED

Returns the total number of events processed by the output window.

**eventsProcessed()**

```
eventsProcessed()=10000
```

### 3.34.7 EVENTTIMESTAMP

Returns the timestamp of the current event.

**eventTimestamp()**

```
eventTimestamp()=1506347669000000
```

### 3.34.8 INPUT

Returns the name of the event stream processing input window.

**input()**

```
input()=sourceWindow
```

### 3.34.9 ISLASTEVENTINBLOCK

Returns true when the event being processed is the last event in the event block. Otherwise, return false.

**isLastEventInBlock()**

```
isLastEventInBlock()=true
```

### 3.34.10 ISNOTRETENTION

Returns true when this event is not generated by a retention policy. Returns false when this event is generated by a retention policy.

**isNotRetention()**

```
isNotRetention()=true
```

### 3.34.11 ISRETENTION

Returns true when the event is generated by a retention policy. Returns false when the event is not generated by a retention policy.

**isRetention()**

```
isRetention()=true
```

### 3.34.12 OPCODE

Returns the opcode of the current event.

**opcode()**

```
opcode()=insert  
opcode()=upsert  
opcode()=delete
```

### 3.34.13 OUTPUT

Returns the name of the event stream processing output window.

**output()**

```
output()=myFunctionalWindow
```

### 3.34.14 PROJECTNAME

Returns the name of the project containing the current event.

**projectName()**

```
projectName()=project_1
projectName()=myproject
```

### 3.34.15 General Functions

### 3.34.16 ABS

Returns the absolute floating-point value of the supplied argument.

**abs(argument)**

Table 3- 14 Argument for ABS

Argument	Description
<i>argument</i>	<p>Specifies one of the following:</p> <ul style="list-style-type: none"> <li>• a literal value, either string or numeric. Enclose string values in single or double quotation marks.</li> <li>• a value to be resolved from an event field. Precede field names with the \$ character.</li> <li>• a value that refers to a resource. For example, when you use the <code>xpath</code> function to refer to an XML object named myXML, you specify this: <b>#myXML</b>.</li> <li>• a function.</li> </ul>

```
abs (-55) =55
abs (44) =44
```

### 3.34.17 AND

When both arguments are true, returns true. Otherwise, returns false.

**and**(*argument1*, *argument2*)

Table 3- 15 Arguments for AND

Argument	Description
<i>argument1</i> , <i>argument2</i>	<p>Specifies one of the following:</p> <ul style="list-style-type: none"> <li>• a literal value, either string or numeric. Enclose string values in single or double quotation marks.</li> <li>• a value to be resolved from an event field. Precede field names with the \$ character.</li> <li>• a value that refers to a resource. For example, when you use the <code>xpath</code> function to refer to an XML object named <code>myXML</code>, you specify this: <code>#myXML</code>.</li> <li>• a function.</li> </ul>

If the argument returns a numeric value, then the function returns true when the numeric value is nonzero. Otherwise it returns false.

If the argument returns a string value:

- When the string value is 'true', the function returns true.
- When the string value is 'false', the function returns false.
- Else, when the length of the string is > 0, the function returns true, otherwise false.

```
and(gt(3,2),gt(2,5))=0
and(gt(3,2),gt(12,5))=1
and(0,1)=0
and('non empty string',55)=1
and('true',55)=1
and('',4)=0
```

### 3.34.18 BASE64DECODE

Decodes the supplied base64-encoded string.

**base64Decode**(*string*)

Table 3- 16 Arguments for BASE64DECODE

Argument	Description
<i>string</i>	Specifies a base64-encoded string. There is no length limit to this string.

```
base64Decode('dGhpcyBpcyBhIHRIc3Q=')=this is a test
```

### 3.34.19 BASE64DECODEBINARY

Decodes the supplied base64-encoded string and sets the result to the binary representation of that data.

**base64DecodeBinary**(*string*)

Table 3- 17 Arguments for BASE64DECODEBINARY

Argument	Description
<i>string</i>	Specifies a base64-encoded string. There is no length limit to this string.

```
base64DecodeBinary('dGhpcyBpcyBhIHRlc3Q=')=<binary data>
```

### 3.34.20 BASE64ENCODE

Encodes the supplied string.

**base64Encode**(*string*)

Table 3- 18 Argument for BASE64ENCODE

Argument	Description
<i>string</i>	Specifies a text string. There is no length limit to this string.

```
base64Encode('this is a test')=dGhpcyBpcyBhIHRlc3Q=
```

### 3.34.21 BASE64ENCODEBINARY

Encodes the supplied binary data and returns an encoded string

**base64Encode**(*binary\_data*)

```
base64EncodeBinary(<binary data>)=dGhpcyBpcyBhIHRlc3Q=
```

### 3.34.22 BETWEEN

When the first argument is greater than the second and less than the third, returns true. Otherwise, returns false.

Table 3- 19 Arguments for BETWEEN

Argument	Description
<i>argument1, argument2, argument3</i>	Specifies one of the following: <ul style="list-style-type: none"> <li>• a literal value, either string or numeric. Enclose string values in single or double quotation marks.</li> <li>• a value to be resolved from an event field. Precede field names with the \$ character.</li> <li>• a value that refers to a resource. For example, when you use the <code>xpath</code> function to refer to an XML object named <code>myXML</code>, you specify this: <code>#myXML</code>.</li> <li>• a function.</li> </ul>

`between(20,17,30)=1`

`between(20,17,15)=0`

### 3.34.23 BOOLEAN

When the supplied argument is a string, returns true when the string has length greater than 0. When the supplied argument is numeric, returns true when value is not equal to 0. When the supplied argument is a Boolean expression, returns true when the value is true. Otherwise, returns false.

`boolean(argument)`

Table 3- 20 Arguments to BOOLEAN

Argument	Description
<i>argument</i>	Specifies one of the following: <ul style="list-style-type: none"> <li>• a literal value, either string or numeric. Enclose string values in single or double quotation marks.</li> <li>• a value to be resolved from an event field. Precede field names with the \$ character.</li> <li>• a value that refers to a resource. For example, when you use the <code>xpath</code> function to refer to an XML object named <code>myXML</code>, you specify this: <code>#myXML</code>.</li> <li>• a function.</li> </ul>

**Note**

The special string values 'true' and 'false' are handled outside the string length > 0. If 'true', the function returns 1. If 'false', the function returns 0.

```
boolean('my string')=1
boolean('')=0
boolean(10)=1
boolean(0)=0
boolean(gt(4,7))=0
boolean(gt(7,5))=1
```

**3.34.24 CEILING**

Returns the integer value above the numeric value of the supplied argument.

**ceiling**(*argument*)

Table 3- 21 Argument for CEILING

Argument	Description
<i>argument</i>	Specifies one of the following: <ul style="list-style-type: none"> <li>• a literal value, either string or numeric. Enclose string values in single or double quotation marks.</li> <li>• a value to be resolved from an event field. Precede field names with the \$ character.</li> <li>• a value that refers to a resource. For example, when you use the <code>xpath</code> function to refer to an XML object named <code>myXML</code>, you specify this: <code>#myXML</code>.</li> <li>• a function.</li> </ul>

```
ceiling(product(4,4.1))=17
```

**3.34.25 COMPARE**

Compares the first argument to the second. If the first argument is less than the second, then it returns -1. If the first is greater than the second, then it returns 1. If the first is equal to the second, then it returns 0.

**compare**(*argument1*, *argument2*)

Table 3- 22 Arguments to COMPARE

Argument	Description
<i>argument1</i> , <i>argument2</i>	<p>Specifies one of the following:</p> <ul style="list-style-type: none"> <li>• a literal value, either string or numeric. Enclose string values in single or double quotation marks.</li> <li>• a value to be resolved from an event field. Precede field names with the \$ character.</li> <li>• a value that refers to a resource. For example, when you use the <code>xpath</code> function to refer to an XML object named <code>myXML</code>, you specify this: <code>#myXML</code>.</li> <li>• a function.</li> </ul>

The type of the first argument determines whether equality is determined by a string or numeric comparison.

```
compare('bears', 'lions')=-1
compare('lions', 'bears')=1
compare('bears', 'bears')=0
compare(10, 20)=-1
compare(20, 10)=1
compare(10, 10)=0
```

**3.34.26 CONCAT**

Returns a string that is the concatenation of the string values of the supplied arguments.

**concat**(*argument1*, *argument2*,...<*argumentN*>)

Table 3- 23 Arguments to CONCAT

Argument	Description
<i>argument1</i> , ... <i>argumentN</i>	<p>Specifies one of the following:</p> <ul style="list-style-type: none"> <li>• a literal value, either string or numeric. Enclose string values in single or double quotation marks.</li> <li>• a value to be resolved from an event field. Precede field names with the \$ character.</li> <li>• a value that refers to a resource. For example, when you use the <code>xpath</code> function to refer to an XML object named <code>myXML</code>, you specify this: <code>#myXML</code>.</li> <li>• a function.</li> </ul>

A minimum of two arguments is required.

```
concat('Name: ', 'Joe', ', Age: ', floor(sum(25,10)), '.') = Name: Joe, Age: 35.
```

### 3.34.27 CONCATDELIM

Returns a string that is the concatenation of the supplied values separated by the specified delimiter.

**concatDelim**(*'delimiter'*, *argument1*, *argument2*, ...<*argumentN*>)

Table 3- 24 Arguments to CONCATDELIM

Argument	Description
'delimiter'	Specifies a character used as a delimiter.
<i>argument1</i> , ... <i>argumentN</i>	Specifies one of the following: <ul style="list-style-type: none"> <li>• a literal value, either string or numeric. Enclose string values in single or double quotation marks.</li> <li>• a value to be resolved from an event field. Precede field names with the \$ character.</li> <li>• a value that refers to a resource. For example, when you use the <code>xpath</code> function to refer to an XML object named <code>myXML</code>, you specify this: <b>#myXML</b>.</li> <li>• a function.</li> </ul>

```
concatDelim('.', 'www', 'sas', 'com') = www.sas.com
```

### 3.34.28 CONTAINS

When the string value of the first argument contains the string value of the second, it returns true. Otherwise, it returns false.

**contains**(*argument1*, *argument2*)

Table 3- 25 Arguments to CONTAINS

Argument	Description
<i>argument1, argument2</i>	Specifies one of the following: <ul style="list-style-type: none"> <li>• a literal value, either string or numeric. Enclose string values in single or double quotation marks.</li> <li>• a value to be resolved from an event field. Precede field names with the \$ character.</li> <li>• a value that refers to a resource. For example, when you use the <code>xpath</code> function to refer to an XML object named <code>myXML</code>, you specify this: <code>#myXML</code>.</li> <li>• a function.</li> </ul>

```
contains('www.sas.com','sas') = true
contains('www.google.com','sas') = false
```

### 3.34.29 DECREMENT

Returns the numeric value of the supplied argument minus 1. This function supports only integers.

**decrement**(*argument*)

Table 3- 26 Arguments to DECREMENT

Argument	Description
<i>argument</i>	Specifies an integer.

```
decrement(10)=9
```

### 3.34.30 DIFF

Returns the value of the first argument minus the second.

`diff(argument1, argument2)`

Table 3- 27 Arguments to DIFF

Argument	Description
<i>argument1, argument2</i>	<p>Specifies one of the following:</p> <ul style="list-style-type: none"> <li>• a literal value, either string or numeric. Enclose string values in single or double quotation marks.</li> <li>• a value to be resolved from an event field. Precede field names with the \$ character.</li> <li>• a value that refers to a resource. For example, when you use the <code>xpath</code> function to refer to an XML object named <code>myXML</code>, you specify this: <code>#myXML</code>.</li> <li>• a function.</li> </ul>

`diff(sum(8,7),22)=-7.0`

### 3.34.31 EQUALS

Returns true when the first argument is equal to the second. Otherwise, it returns false.

`equals(argument1, argument2)`

Table 3- 28 Arguments to EQUALS

Argument	Description
<i>argument1, argument2</i>	<p>Specifies one of the following:</p> <ul style="list-style-type: none"> <li>• a literal value, either string or numeric. Enclose string values in single or double quotation marks.</li> <li>• a value to be resolved from an event field. Precede field names with the \$ character.</li> <li>• a value that refers to a resource. For example, when you use the <code>xpath</code> function to refer to an XML object named <code>myXML</code>, you specify this: <code>#myXML</code>.</li> <li>• a function.</li> </ul>

The type of the comparison depends on the type of the first argument.

`equals('sas.com',string('sas','.com'))=1`

`equals('sas.com','google.com')=0`

`equals(10,sum(5,5))=1`

### 3.34.32 FALSE

Returns true when the Boolean value of the argument is false. Otherwise, it returns true.

Table 3- 29 Arguments to FALSE

Argument	Description
<i>argument</i>	Specifies one of the following: <ul style="list-style-type: none"> <li>• a literal value, either string or numeric. Enclose string values in single or double quotation marks.</li> <li>• a value to be resolved from an event field. Precede field names with the \$ character.</li> <li>• a value that refers to a resource. For example, when you use the <code>xpath</code> function to refer to an XML object named <code>myXML</code>, you specify this: <code>#myXML</code>.</li> <li>• a function.</li> </ul>

```
false(0)=1
false(equals(10,10))=0
```

### 3.34.33 FLOOR

Returns the integer value below the numeric value of the supplied argument.

**floor**(*argument*)

Table 3- 30 Arguments to FLOOR

Argument	Description
<i>argument</i>	Specifies one of the following: <ul style="list-style-type: none"> <li>• a literal value, either string or numeric. Enclose string values in single or double quotation marks.</li> <li>• a value to be resolved from an event field. Precede field names with the \$ character.</li> <li>• a value that refers to a resource. For example, when you use the <code>xpath</code> function to refer to an XML object named <code>myXML</code>, you specify this: <code>#myXML</code>.</li> <li>• a function.</li> </ul>

The type of the comparison depends on the type of the first argument.

```
floor(product(3.5,7))=24
```

### 3.34.34 GT

Returns true when the first argument is greater than the second. Otherwise, it returns false.

`gt(argument, argument2)`

Table 3- 31 Arguments to GT

Argument	Description
<i>argument, argument2</i>	<p>Specifies one of the following:</p> <ul style="list-style-type: none"> <li>• a literal value, either string or numeric. Enclose string values in single or double quotation marks.</li> <li>• a value to be resolved from an event field. Precede field names with the \$ character.</li> <li>• a value that refers to a resource. For example, when you use the <code>xpath</code> function to refer to an XML object named myXML, you specify this: <code>#myXML</code>.</li> <li>• a function.</li> </ul>

The type of the comparison depends on the type of the first argument.

```
gt(sum(10,4),13)=true
gt('internet explorer','internet explorer')=false
gt('internet explorer','netscape')=false
```

### 3.34.35 GTE

The type of the comparison depends on the type of the first argument.

`gte(argument, argument2)`

Table 3- 32 Arguments to GTE

Argument	Description
<i>argument, argument2</i>	<p>Specifies one of the following:</p> <ul style="list-style-type: none"> <li>• a literal value, either string or numeric. Enclose string values in single or double quotation marks.</li> <li>• a value to be resolved from an event field. Precede field names with the \$ character.</li> <li>• a value that refers to a resource. For example, when you use the <code>xpath</code> function to refer to an XML object named myXML, you specify this: <code>#myXML</code>.</li> <li>• a function.</li> </ul>

The type of the comparison depends on the type of the first argument.

```
gte(sum(10,4),13)=true  
gte('internet explorer','internet explorer')=true  
gte('netscape','internet explorer')=true
```

**3.34.36 GUID**

Returns a globally unique identifier.

**guid()**

```
guid()=46ca7b9e-b11d-41be-a3eb-be8bc8553aed  
guid()=319cd2a6-1b30-4c1b-8ac7-55e9465ea066
```

**3.34.37 INCREMENT**

Returns the numeric value of the first argument + 1. This function only supports integers.

**increment(*argument*)**

Table 3- 33 Arguments to INCREMENT

Argument	Description
<i>argument</i>	Specifies an integer value or a function that returns an integer value.

```
increment(10)=11
```

**3.34.38 IF**

When the Boolean value of the first argument is true, returns the second argument. Otherwise, it returns the third argument if specified.

`if(argument1 , argument2, <argument3>)`

Table 3- 34 Arguments to IF

Argument	Description
<i>argument1</i>	Specifies a Boolean expression.
<i>argument2</i>	Specifies one of the following: <ul style="list-style-type: none"> <li>• a literal value, either string or numeric. Enclose string values in single or double quotation marks.</li> <li>• a value to be resolved from an event field. Precede field names with the \$ character.</li> <li>• a value that refers to a resource. For example, when you use the <code>xpath</code> function to refer to an XML object named <code>myXML</code>, you specify this: <code>#myXML</code>.</li> <li>• a function.</li> </ul>
<i>argument3</i>	Specifies one of the following: <ul style="list-style-type: none"> <li>• a literal value, either string or numeric. Enclose string values in single or double quotation marks.</li> <li>• a value to be resolved from an event field. Precede field names with the \$ character.</li> <li>• a value that refers to a resource. For example, when you use the <code>xpath</code> function to refer to an XML object named <code>myXML</code>, you specify this: <code>#myXML</code>.</li> <li>• a function.</li> </ul>

The type of the comparison depends on the type of the first argument.

```
if(equals('x','x'),'one','two')=one
if(equals('x','y'),'one','two')=two
if(equals('x','y'),'one')=
```

### 3.34.39 IFNEXT

Evaluates the first argument in a pair. When the argument evaluates to true, the function returns the value of the second argument in the pair.

**ifNext**(*argument* , *argument2*, ...<*argumentN*>, <*argumentN+1*>)

Table 3- 35 Arguments to IFNEXT

Argument	Description
<i>argument1</i>	Specifies a Boolean expression.
<i>argument2</i>	Specifies one of the following: <ul style="list-style-type: none"> <li>• a literal value, either string or numeric. Enclose string values in single or double quotation marks.</li> <li>• a value to be resolved from an event field. Precede field names with the \$ character.</li> <li>• a value that refers to a resource. For example, when you use the <code>xpath</code> function to refer to an XML object named <code>myXML</code>, you specify this: <code>#myXML</code>.</li> <li>• a function.</li> </ul>

```
ifNext(gt(20,10),'value 1',lt(20,10),'value 2')=value 1
ifNext(gt(20,100),'value 1',lt(20,100),'value 2')=value 2
```

### 3.34.40 IN

Returns true when `expr0` matches `expr1`, or any expression between `expr1` and `exprN`, inclusive. Otherwise, returns false.

**in**(*expr0*, *expr1*<, ...*exprN*>)

Table 3- 36 Arguments to IN

Argument	Description
<i>expr0</i> , <i>expr1</i>	Specifies the expressions to be evaluated. The minimum number of expressions is 2.
<i>exprN</i>	Specifies additional expressions to be evaluated.

### 3.34.41 INDEX

Returns the value of `argumentN`, where `N` is the numeric value of specified index.

**index**(*index* , *argument0*, ...<*argumentN*>)

Table 3- 37 Arguments to INDEX

Argument	Description
<i>index</i>	Specifies an integer or a function that returns an integer.
<i>argument0</i> , <i>argument1</i> , ... <i>argumentN</i>	Specifies one of the following: <ul style="list-style-type: none"> <li>• a literal value, either string or numeric. Enclose string values in single or double quotation marks.</li> <li>• a value to be resolved from an event field. Precede field names with the \$ character.</li> <li>• a value that refers to a resource. For example, when you use the <code>xpath</code> function to refer to an XML object named <code>myXML</code>, you specify this: <code>#myXML</code>.</li> <li>• a function.</li> </ul>

The minimum number of arguments is 2.

```
index(1, 'larry', 'moe', 'curly')=moe
index(random(0, 4) ,10, 20, 30, 40, 50)=40
index(random(0, 4) ,10, 20, 30, 40, 50)=20
```

### 3.34.42 INDEXOF

Returns the 0-based index of the string value of the first argument in the string value of the second argument. Returns -1 when the value is not found.

**indexOf**(*argument1* , *argument2*)

Table 3- 38 Arguments to INDEXOF

Argument	Description
<i>argument1</i> , <i>argument2</i>	Specifies a string.

The minimum number of arguments is 2.

```
indexOf('Edge Streaming Analytics', 'Stream')=10
indexOf('Edge Streaming Analytics', 'Google')=-1
```

### 3.34.43 INTEGER

Returns the integer value of the argument.

**integer**(*argument*)

Table 3- 39 Arguments to INTEGER

Argument	Description
<i>argument</i>	Specifies one of the following: <ul style="list-style-type: none"> <li>• a literal value, either string or numeric. Enclose string values in single or double quotation marks.</li> <li>• a value to be resolved from an event field. Precede field names with the \$ character.</li> <li>• a value that refers to a resource. For example, when you use the <code>xpath</code> function to refer to an XML object named <code>myXML</code>, you specify this: <code>#myXML</code>.</li> <li>• a function.</li> </ul>

```
integer('88.45')=88
integer(111.23)=111
```

**3.34.44 INTERVAL**

Takes the numeric value of the first argument and compares it to the numeric values of all remaining arguments. If the numeric value of the first argument is less than one of the arguments that follow, then the value of the argument that follows that one is returned.

**interval**(*argument*, *argument2*, *argument3*, ...<*argumentN*>)

Table 3- 40 Arguments to INTERVAL

Argument	Description
<i>argument</i>	Specifies one of the following: <ul style="list-style-type: none"> <li>• a literal value, either string or numeric. Enclose string values in single or double quotation marks.</li> <li>• a value to be resolved from an event field. Precede field names with the \$ character.</li> <li>• a value that refers to a resource. For example, when you use the <code>xpath</code> function to refer to an XML object named <code>myXML</code>, you specify this: <code>#myXML</code>.</li> <li>• a function.</li> </ul>

```
interval(85,60,'F',70,'D',80,'C',90,'B','A')=B
interval(90,60,'F',70,'D',80,'C',90,'B','A')=A
```

### 3.34.45 ISNULL

Returns true when the argument is not set, otherwise it returns false.

**isNull(argument)**

Table 3- 41 Arguments to ISNULL

Argument	Description
<i>argument</i>	Specifies one of the following: <ul style="list-style-type: none"> <li>• a literal value, either string or numeric. Enclose string values in single or double quotation marks.</li> <li>• a value to be resolved from an event field. Precede field names with the \$ character.</li> <li>• a value that refers to a resource. For example, when you use the <code>xpath</code> function to refer to an XML object named <code>myXML</code>, you specify this: <code>#myXML</code>.</li> <li>• a function.</li> </ul>

```
isNull($unresolved)=1
isNull('my string')=0
```

### 3.34.46 ISSET

Returns true when the argument is not set, otherwise it returns false.

**isSet(argument)**

Table 3- 42 Arguments to ISSET

Argument	Description
<i>argument</i>	Specifies one of the following: <ul style="list-style-type: none"> <li>• a literal value, either string or numeric. Enclose string values in single or double quotation marks.</li> <li>• a value to be resolved from an event field. Precede field names with the \$ character.</li> <li>• a value that refers to a resource. For example, when you use the <code>xpath</code> function to refer to an XML object named <code>myXML</code>, you specify this: <code>#myXML</code>.</li> <li>• a function.</li> </ul>

```
isSet($unresolved)=0
isSet('my string')=1
```

### 3.34.47 JSON

Parses the JSON object specified in the first argument and returns a value as a function of the second argument.

**json**(*argument* , *argument2*)

Table 3- 43 Arguments to ISSET

Argument	Description
<i>argument</i>	Specifies a JSON object.
<i>argument2</i>	Specifies an evaluation string. In a name value pair, specify the name of the object whose value you want to return.

```

json(' {first:"john",last:"smith",hobbies:["running","reading","golf"]} ',
'first')=john
json(' {first:"john",last:"smith",hobbies:["running","reading","golf"]} ',
'last')=smith
json(' {first:"john",last:"smith",hobbies:["running","reading","golf"]} ',
'hobbies[1]')=reading
json(#myJson,'hobbies[1]')=reading
    
```

### 3.34.48 LASTINDEXOF

Returns the last index of the string value of the second argument in the string value of the first, or -1 when the value is not found.

**lastIndexOf**(*argument* , *argument2*)

Table 3- 44 Arguments to LASTINDEXOF

Argument	Description
<i>argument</i> , <i>argument2</i>	Specifies one of the following: <ul style="list-style-type: none"> <li>• a literal value, either string or numeric. Enclose string values in single or double quotation marks.</li> <li>• a value to be resolved from an event field. Precede field names with the \$ character.</li> <li>• a value that refers to a resource. For example, when you use the <code>xpath</code> function to refer to an XML object named <code>myXML</code>, you specify this: <code>#myXML</code>.</li> <li>• a function.</li> </ul>

```
lastIndexOf('http://www.sas.com/products/webanalytics','/')=27
```

### 3.34.49 LISTITEM

This function has two uses.

- 1) Parses the first argument using the specified delimiter and then returns the value at the specified index.
- 2) References an existing list in the specified function context and returns its value at the specified index.

**listItem(argument , delimiter, index)**

**listItem(reference , index)**

Table 3- 45 Arguments to LISTITEM

Argument	Description
<i>delimiter</i>	Specifies a character used as a delimiter.
<i>index</i>	Specifies a numeric value used as an index.
<i>reference</i>	Specifies a reference to a function context.
<i>argument</i>	Specifies a delimited string.

```
listItem('one,two,three,four',' ',2)=three
```

```
listItem(#myList,0)=one
```

### 3.34.50 LISTSIZE

This function has two uses.

- 1) Parses the first argument using the specified delimiter and then returns its size.
- 2) References an existing list in the specified function context and returns its size.

**listSize(argument , delimiter)**

**listSize(reference )**

Table 3- 46 Arguments to LISTSIZE

Argument	Description
argument	Specifies a delimited string.
<i>delimiter</i>	Specifies a character used as a delimiter.
reference	Specifies a reference to a function context.

```
listSize('one,two,three,four',' ',')=4
```

```
listSize(#myList)=4
```

### 3.34.51 LONG

Returns the long value of the argument.

**long(argument)**

Table 3- 47 Arguments to LONG

Argument	Description
<i>argument</i>	Specifies one of the following: <ul style="list-style-type: none"><li>• a literal value, either string or numeric. Enclose string values in single or double quotation marks.</li><li>• a value to be resolved from an event field. Precede field names with the \$ character.</li><li>• a value that refers to a resource. For example, when you use the <code>xpath</code> function to refer to an XML object named <code>myXML</code>, you specify this: <b>#myXML</b>.</li><li>• a function.</li></ul>

`long('88.45')=88`

`long(111.23)=111`

**3.34.52 LT**

Returns true when the first argument is less than the second. Otherwise, returns false.

`lt(argument, argument2)`

Table 3- 48 Arguments to LT

Argument	Description
<i>argument</i>	Specifies one of the following: <ul style="list-style-type: none"> <li>• a literal value, either string or numeric. Enclose string values in single or double quotation marks.</li> <li>• a value to be resolved from an event field. Precede field names with the \$ character.</li> <li>• a value that refers to a resource. For example, when you use the <code>xpath</code> function to refer to an XML object named <code>myXML</code>, you specify this: <code>#myXML</code>.</li> <li>• a function.</li> </ul>
<i>argument2</i>	Specifies one of the following: <ul style="list-style-type: none"> <li>• a literal value, either string or numeric. Enclose string values in single or double quotation marks.</li> <li>• a value to be resolved from an event field. Precede field names with the \$ character.</li> <li>• a value that refers to a resource. For example, when you use the <code>xpath</code> function to refer to an XML object named <code>myXML</code>, you specify this: <code>#myXML</code>.</li> <li>• a function.</li> </ul>

```
lt(sum(10,4),13)=false
lt('internet explorer','internet explorer')=true
lt('internet explorer','netscape')=true
```

### 3.34.53 LTE

Returns true when the first argument is less than or equal to the second. Otherwise, returns false.

**lte**(*argument* , *argument2*)

Table 3- 49 Arguments to LTE

Argument	Description
<i>argument</i> , <i>argument2</i>	Specifies one of the following: <ul style="list-style-type: none"> <li>• a literal value, either string or numeric. Enclose string values in single or double quotation marks.</li> <li>• a value to be resolved from an event field. Precede field names with the \$ character.</li> <li>• a value that refers to a resource. For example, when you use the <code>xpath</code> function to refer to an XML object named <code>myXML</code>, you specify this: <code>#myXML</code>.</li> <li>• a function.</li> </ul>

```
lte(sum(10,3),13)=true
lt('internet explorer','internet explorer')=true
lt('internet explorer','netscape')=true
```

### 3.34.54 MAPVALUE

This function has two uses. 1) Parses name-value pairs from the first argument using the specified outer delimiter and the specified inner delimiter, and then extracts the value for the specified name. 2) References an existing value map in the referenced function context, and then extracts the value for the name.

**mapValues**(*argument* , *outerdelimiter*, *innerdelimiter*, *delimiter*, *name*)

**mapValues**(*#reference*, *name*)

Table 3- 50 Arguments to MAPVALUE

Argument	Description
<i>argument</i>	Specifies a delimited string of name-value pairs
<i>outerdelimiter</i> , <i>innerdelimiter</i> <i>delimiter</i>	Specifies characters used as delimiters.
<i>name</i>	Specifies the name in the name-value pair specified in argument.
<i>#reference</i>	Specifies a reference to a function context.

```
mapValue('first:John;last:Doe;occupation:plumber',';',':', 'occupation')=plumber
mapValue(#myMap, 'occupation')=plumber
```

### 3.34.55 MAPVALUES

This function has two uses. 1) Parses name-value pairs from the first argument using the specified outer delimiter and the specified inner delimiter, and then extracts the values for each specified name. 2) References an existing value map in the referenced function context and extracts the values for each specified name.

**mapValues**(*argument*, *outerdelimiter*, *innerdelimiter*, *delimiter*, *name1*,...<*nameN*>)

**mapValues**(#*reference*, *name1*, ...<*nameN*>))

Table 3- 51 Arguments to MAPVALUES

Argument	Description
<i>argument</i>	Specifies a delimited string of name-value pairs
<i>outerdelimiter</i> , <i>innerdelimiter</i> , <i>delimiter</i>	Specifies characters used as delimiters.
<i>name1</i> , ... <i>nameN</i>	Specifies the name in the name-value pair specified in argument.
# <i>reference</i>	Specifies a reference to a function context.

```
mapValues('first=John,last=Doe',' ','=' , ':' , 'first', 'last')=John:Doe
mapValues(#myMap, 'first', 'last')=John:Doe
```

### 3.34.56 MAX

Returns the largest numeric value of all specified arguments.

**max**(*argument*, *argument2*, ...<*argumentN*>)

Table 3- 52 Arguments for MAX

Argument	Description
<i>argument</i> , <i>argument2</i> , ... <i>argumentN</i>	<p>Specifies one of the following:</p> <ul style="list-style-type: none"> <li>• a literal value, either string or numeric. Enclose string values in single or double quotation marks</li> <li>• a value to be resolved from an event field. Precede field names with the \$ character.</li> <li>• a value that refers to a resource. For example, when you use the <code>xpath</code> function to refer to an XML object named <code>myXML</code>, you specify this: <b>#myXML</b>.</li> <li>• a function.</li> </ul>

The minimum number of arguments is 1.

```
max(33, 44.2, sum(1, 3, 2, 12), -33.21)=44.2
```

### 3.34.57 MEAN

Returns the mean value of all specified arguments.

**mean**(*argument* , *argument2*, ...<*argumentN*>)

Table 3- 53 Arguments for MEAN

Argument	Description
<i>argument</i> , <i>argument2</i> , ... <i>argumentN</i>	Specifies one of the following: <ul style="list-style-type: none"> <li>• a literal value, either string or numeric. Enclose string values in single or double quotation marks</li> <li>• a value to be resolved from an event field. Precede field names with the \$ character.</li> <li>• a value that refers to a resource. For example, when you use the <code>xpath</code> function to refer to an XML object named <code>myXML</code>, you specify this: <code>#myXML</code>.</li> <li>• a function.</li> </ul>

`mean( 33 , 44 . 2 , sum( 1 , 3 , 2 , 12 ) , -33 . 21 ) = 15 . 4975`

### 3.34.58 MIN

Returns the smallest numeric value of all specified arguments.

**min**(*argument* , *argument2*, ...<*argumentN*>)

Table 3- 54 Arguments for MIN

Argument	Description
<i>argument</i> , <i>argument2</i> , ... <i>argumentN</i>	Specifies one of the following: <ul style="list-style-type: none"> <li>• a literal value, either string or numeric. Enclose string values in single or double quotation marks</li> <li>• a value to be resolved from an event field. Precede field names with the \$ character.</li> <li>• a value that refers to a resource. For example, when you use the <code>xpath</code> function to refer to an XML object named <code>myXML</code>, you specify this: <code>#myXML</code>.</li> <li>• a function</li> </ul>

The minimum number of arguments is 1.

```
min(33,44.2,sum(1,3,2,12),-33.21)=-33.21
```

### 3.34.59 MOD

Returns the remainder of the first argument divided by the second.

```
mod(argument, argument2)
```

Table 3- 55 Arguments for MOD

Argument	Description
<i>argument, argument2</i>	Specifies one of the following: <ul style="list-style-type: none"> <li>• a literal value, either string or numeric. Enclose string values in single or double quotation marks</li> <li>• a value to be resolved from an event field. Precede field names with the \$ character.</li> <li>• a value that refers to a resource. For example, when you use the <code>xpath</code> function to refer to an XML object named <code>myXML</code>, you specify this: <code>#myXML</code>.</li> <li>• a function.</li> </ul>

```
mod(10,3)=1.0
```

### 3.34.60 NEG

Returns the negative numeric value of the specified argument.

```
neg(argument)
```

Table 3- 56 Arguments for NEG

Argument	Description
<i>argument</i>	Specifies one of the following: <ul style="list-style-type: none"> <li>• a literal value, either string or numeric. Enclose string values in single or double quotation marks</li> <li>• a value to be resolved from an event field. Precede field names with the \$ character.</li> <li>• a value that refers to a resource. For example, when you use the <code>xpath</code> function to refer to an XML object named <code>myXML</code>, you specify this: <code>#myXML</code>.</li> <li>• a function.</li> </ul>

`neg(55)=-55`

### 3.34.61 NORMALIZESPACE

Returns a string that is created by replacing any extra white space in the specified argument with a single space.

`normalizeSpace(argument)`

Table 3- 57 Arguments for NORMALIZESPACE

Argument	Description
<i>argument</i>	Specifies a string.

`normalizeSpace('Sentence with many spaces')=Sentence with many spaces`

### 3.34.62 NEQUALS

Returns true when the first argument is not equal to the second. Otherwise, returns false.

`nequals(argument, argument2)`

Table 3- 58 Arguments for NEQUALS

Argument	Description
<i>argument, argument2</i>	<p>Specifies one of the following:</p> <ul style="list-style-type: none"> <li>• a literal value, either string or numeric. Enclose string values in single or double quotation marks</li> <li>• a value to be resolved from an event field. Precede field names with the \$ character.</li> <li>• a value that refers to a resource. For example, when you use the <code>xpath</code> function to refer to an XML object named <code>myXML</code>, you specify this: <code>#myXML</code>.</li> <li>• a function.</li> </ul>

```
nequals('sas.com',string('sas','.com'))=0
nequals('sas.com','google.com')=1
nequals(10,sum(5,5))=0
```

### 3.34.63 NOT

Returns true when the Boolean value of the argument is false. Otherwise, returns false.

`not(argument)`

Table 3- 59 Arguments for NOT

Argument	Description
<i>argument</i>	<p>Specifies one of the following:</p> <ul style="list-style-type: none"> <li>• a literal value, either string or numeric. Enclose string values in single or double quotation marks</li> <li>• a value to be resolved from an event field. Precede field names with the \$ character.</li> <li>• a value that refers to a resource. For example, when you use the <code>xpath</code> function to refer to an XML object named <code>myXML</code>, you specify this: <code>#myXML</code>.</li> <li>• a function.</li> </ul>

```
not(0)=1
not(equals(10,10))=0
```

### 3.34.64 NUMBER

Returns the numeric value of the argument.

`number(argument)`

Table 3- 60 Arguments for NUMBER

Argument	Description
<i>argument</i>	Specifies one of the following: <ul style="list-style-type: none"><li>• a literal value, either string or numeric. Enclose string values in single or double quotation marks</li><li>• a value to be resolved from an event field. Precede field names with the \$ character.</li><li>• a value that refers to a resource. For example, when you use the <code>xpath</code> function to refer to an XML object named <code>myXML</code>, you specify this: <code>#myXML</code>.</li><li>• a function.</li></ul>

```
number('88.45')=88.45  
number(111.23)=111.23  
number(gt(2,1))=1
```

### 3.34.65 OR

Returns true when any of the supplied arguments are true. Otherwise, returns false.

`or(argument, argument2, ...<argumentN>)`

Table 3- 61 Arguments for OR

Argument	Description
<i>argument, argument2, ...argumentN</i>	<p>Specifies one of the following:</p> <ul style="list-style-type: none"> <li>• a literal value, either string or numeric. Enclose string values in single or double quotation marks</li> <li>• a value to be resolved from an event field. Precede field names with the \$ character.</li> <li>• a value that refers to a resource. For example, when you use the <code>xpath</code> function to refer to an XML object named <code>myXML</code>, you specify this: <code>#myXML</code>.</li> <li>• a function.</li> </ul>

The minimum number of arguments is 1.

```
or(equals('a','b'),nequals('a','b'))=1
or(equals('a','b'),nequals('a','a'))=0
```

### 3.34.66 OUTSTR

When the argument contains a string or one of a group of strings, returns the associated value. When there are no matches, returns a specified default value.

```
outstr(argument, string1, value_associated_with_string1,
...<stringN>, <value_associated_with_stringN>, default)
```

Table 3- 62 Arguments for OUTSTR

Argument	Description
<i>argument</i>	Specifies a string.
<i>string1...stringN</i>	Specifies a string or group of strings.
<i>value_associated_with_string1...value_associated_with_stringN</i>	Specifies a value associated with a string or values associated with a group of strings.
<i>default</i>	Specifies a string or group of strings.

```
outstr('government spending','govern','Government','Other')=Government
outstr('spending',('govern','spend'),
'Government or Spending','Other')=Government or Spending
outstr('bob',('govern','spend'),'Government or Spending',
('john','jack','bob'),'Names','Other')=Names
outstr('stream processing',('govern','spend'),'Government or Spending',
('john','jack','bob'),'Names','Other')=Other
```

### 3.34.67 PRECISION

Sets the decimal point precision of the first argument to the second argument.

**precision**(*argument* , *argument2*)

Table 3- 63 Arguments for PRECISION

Argument	Description
<i>argument</i> , <i>argument2</i>	Specifies a numeric value.

`precision(123.44567,2)=123.45`

### 3.34.68 PRODUCT

Returns the product of the supplied arguments.

**product**(*argument* , *argument2*...<*argumentN*>)

Table 3- 64 Arguments for PRODUCT

Argument	Description
<i>argument</i> , <i>argument2</i> ...< <i>argumentN</i> >	Specifies a numeric value or a function that returns a numeric value.

`product(3,sum(2,4),2)=36`

### 3.34.69 QUOTIENT

Returns the quotient of the supplied arguments.

**quotient**(*argument* , *argument2*...<*argumentN*>)

Table 3- 65 Arguments for QUOTIENT

Argument	Description
<i>argument</i> , <i>argument2</i> ...< <i>argumentN</i> >	Specifies a numeric value or a function that returns a numeric value.

`quotient(3,sum(2,4),2)=0.25`

### 3.34.70 RANDOM

Returns a random number between the first argument and the second.

**random**(*argument* , *argument2*)

Table 3- 66 Arguments for RANDOM

Argument	Description
<i>argument</i> , <i>argument2</i>	Specifies a numeric value.

```
random(100,1000)=741
random(100,1000)=356
random(100,1000)=452
precision(random(0,.5),2)=0.17
```

### 3.34.71 RGX

Runs the specified regular expression on a supplied string and returns the result. If a group is specified, the result is the content of the specified numeric regular expression group.

**rgx**(*regular\_expression* , *string...<group>*)

Table 3- 67 Arguments for RGX

Argument	Description
<i>regular_expression</i>	Specifies a regular expression or a reference to a regular expression in the function context.
<i>string</i>	Specifies a string.
<i>group</i>	Specifies a numeric reference to the regular expression.

```
rgx('.*\/view\/([0-9]*)\/([0-9]*)',
'http://cistore-dev.unx.sas.com/products/view/23/4', 1)=23
rgx(#myExpr, 'http://cistore-dev.unx.sas.com/products/view/23/4' ,2)=4
```

### 3.34.72 RGXINDEX

Runs the specified regular expression on a supplied string. When a match is found, returns the index of the match. When no match is found, returns -1.

**rgxIndex**(*regular\_expression* , *string...<group>*)

Table 3- 68 Arguments for RGXINDEX

Argument	Description
<i>regular_expression</i>	Specifies a regular expression or a reference to a regular expression in the function context.
<i>string...stringN</i>	Specifies a string.

```
rgxIndex('developer','larry - manager','moe - tester','curly - developer')=2
```

### 3.34.73 RGXLASTTOKEN

Uses the regular expression in the first argument as a delimiter within the regular expression of the second to find all strings separated by that expression.

```
rgxLastToken(regular_expression1 , regular_expression2...<index_value>)
```

Table 3- 69 Arguments for RGXLASTTOKEN

Argument	Description
<i>regular_expression1</i>	Specifies a regular expression or a reference to a regular expression in the function context.
<i>regular_expression2</i>	Specifies a regular expression
<i>index_value</i>	Specifies an index value (defaults to 0) that counts from the last token in the expression. When this value is greater than 0 and less than or equal to the number of tokens in the regular expression, the token at the value is returned. Otherwise, null is returned.

```
rgxLastToken('/', 'data/opt/sas/dataflux')=dataflux
rgxLastToken('/', 'data/opt/sas/dataflux', 2)=opt
rgxLastToken('\.', 'www.sas.com')=com
```

### 3.34.74 RGXMATCH

Compares the regular expression in the first argument to the second argument and returns a Boolean value that indicates whether a match is found.

```
rgxMatch(regular_expression1 , string...<group>)
```

Table 3- 70 Arguments for RGXMATCH

Argument	Description
<i>regular_expression1</i>	Specifies a regular expression or a reference to a regular expression in the function context.
<i>string</i>	Specifies a string.
<i>group</i>	Specifies a numeric reference to the regular expression. When specified, the result is the content of the specified numeric regular expression group.

```
rgxMatch(' (google|yahoo|bing)', 'http://www.google.com')=1
rgxMatch(' (google|yahoo|bing)', 'http://www.sas.com')=0
```

### 3.34.75 RGXREPLACE

Parses the regular expression in the first argument against the string in the second argument and replaces the first match with the string in the third argument.

**rgxReplace**(*regular\_expression1* , *string1*, *string2*)

Table 3- 71 Arguments for RGXREPLACE

Argument	Description
<i>regular_expression1</i>	Specifies a regular expression or a reference to a regular expression in the function context.
<i>string</i>	Specifies a string.
<i>string2</i>	Specifies a string.

```
rgxReplace(' (google|yahoo|bing)', 'http://www.google.com', 'sas')=http://www.sas.com
```

### 3.34.76 RGXREPLACEALL

Parses the regular expression in the first argument against the string in the second argument and replaces any match with the string in the third argument.

**rgxReplaceAll**(*regular\_expression1* , *string1*, *string2*)

Table 3- 72 Arguments for RGXREPLACEALL

Argument	Description
<i>regular_expression1</i>	Specifies a regular expression or a reference to a regular expression in the function context.
<i>string</i>	Specifies a string.
<i>string2</i>	Specifies a string.

```
rgxReplaceAll(' (google|yahoo|bing)', 'http://www.google.com/google/products', 'sas')
= http://www.sas.com/sas/products
```

### 3.34.77 RGXTOKEN

Uses the first argument as a delimiter within the second argument to find all strings separated by that delimiter.

```
rgxToken(delimiter, string, <index>)
```

Table 3- 73 Arguments for RGXTOKEN

Argument	Description
<i>delimiter</i>	Specifies a regular expression or a reference to a regular expression in the function context.
<i>string</i>	Specifies a string.
<i>index</i>	Specifies a numeric value that serves as an index when parsing the string. When the index is less than or equal to the number of tokens in the string, the token at the index value is returned. Otherwise, null is returned. The default value is 0.

```
rgxToken('/', 'data/opt/sas/dataflux')=data
rgxToken('/', 'data/opt/sas/dataflux', 2)=sas
rgxToken('\.', 'www.sas.com')=www
```

### 3.34.78 RGXV

Parses the regular expression in the first argument against the string in the second argument and returns all matches delimited by the specified delimiter.

```
rgxV(regular_expression, string, delimiter<group>)
```

Table 3- 74 Arguments for RGXV

Argument	Description
regular_expression	Specifies a regular expression or a reference to a regular expression in the function context.
<i>string</i>	Specifies a string.
delimiter	Specifies a character value that serves as a delimiter when parsing string.
<i>index</i>	Specifies a numeric reference to the regular expression. When specified, the result is the content of the specified numeric regular expression group.

```
rgxV('(jerry|scott|vince)',
'The ESP product has jerry, scott, and vince working on it', ' : ')=jerry : scott :
vince
```

### 3.34.79 ROUND

Returns the rounded numeric value of the argument.

**round(argument)**

Table 3- 75 Arguments for ROUND

Argument	Description
<i>argument</i>	<p>Specifies one of the following:</p> <ul style="list-style-type: none"> <li>• a literal value, either string or numeric. Enclose string values in single or double quotation marks.</li> <li>• a value to be resolved from an event field. Precede field names with the \$ character.</li> <li>• a value that refers to a resource. For example, when you use the <code>xpath</code> function to refer to an XML object named <code>myXML</code>, you specify this: <b>#myXML</b>.</li> <li>• a function.</li> </ul>

```
round(34.56)=35
round(34.46)=34
```

### 3.34.80 SETCONTAINS

This function has two uses.

- 1) Parses a specified set of tokens containing the specified delimiter to check whether a specified string appears within the set.
- 2) References an existing set of tokens in the function context to check whether a specified string appears in the set.

**setContains**(*set\_of\_tokens* , *delimiter*, *string*)  
**setContains**(#*reference string*)

Table 3- 76 Arguments for SETCONTAINS

Argument	Description
set_of_tokens	Specifies a string of tokens.
delimiter	Specifies a character value used as a delimiter.
string	Specifies a string.
reference	Specifies a reference to a function context.

```
setContains('one,two,three,four',' ','two')=1
setContains(#mySet,'five')=0
```

### 3.34.81 STARTSWITH

Returns true when the first argument starts with the second. Otherwise, returns false.

**startsWith**(*argument1* , *argument2*)

Table 3- 77 Arguments for STARTSWITH

Argument	Description
<i>argument1</i> , <i>argument2</i>	Specifies one of the following: <ul style="list-style-type: none"> <li>• a literal value, either string or numeric. Enclose string values in single or double quotation marks.</li> <li>• a value to be resolved from an event field. Precede field names with the \$ character.</li> <li>• a value that refers to a resource. For example, when you use the <code>xpath</code> function to refer to an XML object named <code>myXML</code>, you specify this: <code>#myXML</code>.</li> <li>• a function.</li> </ul>

```
startsWith('www.sas.com','www.')=1
startsWith('www.sas.com','sww.')=0
```

### 3.34.82 STRING

Returns a concatenated single string value from one or more arguments.

**string**(*argument*<, *argument*2,...*argument*N>)

Table 3- 78 Arguments to STRING

Argument	Description
<i>argument</i> (s)	<p>Specifies one of the following:</p> <ul style="list-style-type: none"> <li>• a literal value, either string or numeric. Enclose string values in single or double quotation marks.</li> <li>• a value to be resolved from an event field. Precede field names with the \$ character.</li> <li>• a value that refers to a resource. For example, when you use the <code>xpath</code> function to refer to an XML object named <code>myXML</code>, you specify this: <code>#myXML</code>.</li> <li>• a function.</li> </ul>

```
string(33.9)=33.9
string($id, '-', eventNumber())=eventid-0
```

### 3.34.83 STRINGLENGTH

Returns the length of the string value of the argument.

**stringLength**(*argument*)

Table 3- 79 Arguments to STRINGLENGTH

Argument	Description
<i>argument</i>	<p>Specifies one of the following:</p> <ul style="list-style-type: none"> <li>• a literal value, either string or numeric. Enclose string values in single or double quotation marks.</li> <li>• a value to be resolved from an event field. Precede field names with the \$ character.</li> <li>• a value that refers to a resource. For example, when you use the <code>xpath</code> function to refer to an XML object named <code>myXML</code>, you specify this: <code>#myXML</code>.</li> <li>• a function.</li> </ul>

```
stringLength('SAS ESP XML')=11
```

### 3.34.84 STRIP

Returns the string created after removing leading or trailing white space from the argument.

**strip**(*argument*)

Table 3- 80 Arguments to STRIP

Argument	Description
<i>argument</i>	Specifies a string.

```
strip(' SAS ESP XML ')=SAS ESP XML
```

### 3.34.85 SUBSTRING

Returns the string created by taking the substring of the value of the first argument at the specified index.

**substring**(*argument* , *index*, <*length*>)

Table 3- 81 Arguments to SUBSTRING

Argument	Description
<i>argument</i>	Specifies a string.
<i>index</i>	Specifies a numeric value that defines an index with which to parse the <i>argument</i> .
<i>length</i>	Specifies a numeric value. When specified, the substring is <i>length</i> size, otherwise it contains all characters to the end of the string.

```
substring('www.sas.com', 4, 3)=sas
substring('www.sas.com', 4)=sas.com
```

### 3.34.86 SUBSTRINGAFTER

Returns the string that results from taking the value of the first argument after an occurrence of the value of the second argument.

**substringAfter**(*argument1* , *argument2*, <*index*>)

Table 3- 82 Arguments to SUBSTRINGAFTER

Argument	Description
<i>argument1</i> , <i>argument2</i>	Specifies a string.
<i>index</i>	Specifies a numeric value that defines an index with which to parse the <i>argument1</i> . When specified, the content after that occurrence is returned.

```
substringAfter('www.sas.com', '.')=sas.com
substringAfter('www.sas.com', '.', 2)=com
```

### 3.34.87 SUBSTRINGBEFORE

Returns the string that results from taking the value of the first argument before an occurrence of the value of the second argument.

**substringBefore**(*argument1* , *argument2*, <*index*>)

Table 3- 83 Arguments to SUBSTRINGBEFORE

Argument	Description
<i>argument1</i> , <i>argument2</i>	Specifies a string.
<i>index</i>	Specifies a numeric value that defines an index with which to parse the <i>argument1</i> . When specified, the content after that occurrence is returned.

```
substringBefore('www.sas.com', '.')=www
substringBefore('www.sas.com', '.', 2)=www.sas
```

### 3.34.88 SUM

Returns the sum of the numeric values of all arguments.

**sum**(*argument* , *argument2*...<*argumentN*>)

Table 3- 84 Arguments to SUM

Argument	Description
<i>argument1</i> , <i>argument2</i> ,... <i>argumentN</i>	Specifies one of the following: <ul style="list-style-type: none"> <li>• a literal value, either string or numeric. Enclose string values in single or double quotation marks</li> <li>• a value to be resolved from an event field. Precede field names with the \$ character.</li> <li>• a value that refers to a resource. For example, when you use the <code>xpath</code> function to refer to an XML object named <code>myXML</code>, you specify this: <code>#myXML</code>.</li> <li>• a function.</li> </ul>

```
sum(33, 22, 55.4, 34, min(0, 4, -9))=135.4
```

### 3.34.89 SWITCH

Parses arguments beginning with the second one. When an argument matches the first, it returns the following argument. If no match is found, it returns null.

**switch**(*argument* , *argument2*, ... <*argumentN*>, <*argumentN+1*>)

Table 3- 85 Arguments to SWITCH

Argument	Description
<i>argument1</i> , <i>argument2</i> ,... <i>argumentN+1</i>	Specifies one of the following: <ul style="list-style-type: none"> <li>• a literal value, either string or numeric. Enclose string values in single or double quotation marks</li> <li>• a value to be resolved from an event field. Precede field names with the \$ character.</li> <li>• a value that refers to a resource. For example, when you use the <code>xpath</code> function to refer to an XML object named <code>myXML</code>, you specify this: <code>#myXML</code>.</li> <li>• a function.</li> </ul>

```
switch('bob','jerry','manager','bob','developer')=developer
switch('jerry','jerry','manager','bob','developer')=manager
switch('moe','jerry','manager','bob','developer')=
```

### 3.34.90 SYSTEMMICRO

Returns the number of microseconds since Jan 1, 1970.

**systemMicro()**

```
systemMicro()=1420557039483912
```

### 3.34.91 SYSTEMMILLI

Returns the number of microseconds since Jan 1, 1970.

**systemMilli()**

```
systemMilli()=1420557039483
```

### 3.34.92 TIMECURRENT

Returns the current time.

**timeCurrent()**

```
timeCurrent()=1421157236
timeString(timeCurrent())=Tue Jan 13 08:53:56 2015
```

### 3.34.93 TIMEDAYOFMONTH

Returns the day of the month of the current or specified time.

**timeDayOfMonth**(<argument>)

Table 3- 86 Arguments to TIMEDAYOFMONTH

Argument	Description
<i>argument1</i>	Specifies an expression that defines a specific time, or a function that returns a specific time.

```
timeDayOfMonth()=13
timeDayOfMonth(timeParse('06/21/2015 00:00:00','%m/%d/%Y %H:%M:%S'))=21
```

### 3.34.94 TIMEDAYOFWEEK

Returns the day of the week of the current or specified time.

**timeDayOfWeek**(<argument>)

Table 3- 87 Arguments to TIMEDAYOFWEEK

Argument	Description
<i>argument1</i>	Specifies an expression that defines a specific time, or a function that returns a specific time. Returns a value of 0 through 6 (Sunday through Saturday).

```
timeDayOfWeek()=2
timeDayOfWeek(timeParse('06/21/2015 00:00:00','%m/%d/%Y %H:%M:%S'))=0
```

### 3.34.95 TIMEDAYOFYEAR

Returns the day of the year of the current or specified time.

**timeDayOfYear**(<argument>)

Table 3- 88 Arguments to TIMEDAYOFYEAR

Argument	Description
<i>argument1</i>	Specifies an expression that defines a specific time, or a function that returns a specific time.

```
timeDayOfYear()=12
timeDayOfYear(timeParse('06/21/2015 00:00:00','%m/%d/%Y %H:%M:%S'))=171
```

### 3.34.96 TIMEGMTTOLOCAL

Converts the GMT that is specified in the argument to local time.

Table 3- 89 Arguments to TIMEGMTTOLOCAL

Argument	Description
<i>argument</i>	Specifies a time value or a function that returns a time value.

```
timeString(timeGmtToLocal(timeCurrent()))=Tue Jan 13 02:10:08 2015
```

### 3.34.97 TIMEGMTSTRING

Writes the GMT time represented by the first argument.

**timeGmtString**(*argument*, <*argument2*>)

Table 3- 90 Arguments to TIMEGMTSTRING

Argument	Description
<i>argument</i>	Specifies a time value or a function that returns a time value.
<i>argument2</i>	Specifies a time format.

```
timeString(timeCurrent(), '%Y-%m-%d %H:%M:%S %Z')=2015-02-20 07:57:18 EST
timeGmtString(timeCurrent(), '%Y-%m-%d %H:%M:%S %Z')=2015-02-20 12:57:18 GMT
```

### 3.34.98 TIMEHOUR

Returns the hour of the day of the current or specified time.

**timeHour**(<*argument*>)

Table 3- 91 Arguments to TIMEHOUR

Argument	Description
<i>argument</i>	Specifies an expression that defines a specific time, or a function that returns a specific time.

```
timeHour()=9
timeHour(timeParse('06/21/2015 13:45:15', '%m/%d/%Y %H:%M:%S'))=14
```

### 3.34.99 TIMEMICRO

Returns the number of microseconds of the supplied argument, or the number of microseconds since Jan 1, 1970 when an argument is not specified.

**timeMicro**(*argument*)

Table 3- 92 Argument for TIMEMICRO

Argument	Description
<i>argument</i>	<p>Specifies one of the following:</p> <ul style="list-style-type: none"> <li>• a literal value, either string or numeric. Enclose string values in single or double quotation marks.</li> <li>• a value to be resolved from an event field. Precede field names with the \$ character.</li> <li>• a value that refers to a resource. For example, when you use the <code>xpath</code> function to refer to an XML object named <code>myXML</code>, you specify this: <code>#myXML</code>.</li> <li>• a function.</li> </ul>

```
timeMicro()=1553283757000000
timeMicro(timeParse('06/21/2015 00:00:00','%m/%d/%Y %H:%M:%S'))=143485920000000
```

### 3.34.100 TIMEMILLI

Returns the number of milliseconds of the supplied argument, or the number of microseconds since Jan 1, 1970 when an argument is not specified.

**timeMilli**(*argument*)

Table 3- 93 Argument for TIMEMILLI

Argument	Description
<i>argument</i>	<p>Specifies one of the following:</p> <ul style="list-style-type: none"> <li>• a literal value, either string or numeric. Enclose string values in single or double quotation marks.</li> <li>• a value to be resolved from an event field. Precede field names with the \$ character.</li> <li>• a value that refers to a resource. For example, when you use the <code>xpath</code> function to refer to an XML object named <code>myXML</code>, you specify this: <code>#myXML</code>.</li> <li>• a function.</li> </ul>

```
timeMilli()=1553283876000
timeMilli(timeParse('06/21/2015 00:00:00','%m/%d/%Y %H:%M:%S'))=1434859200000
```

### 3.34.101 TIMEMINUTE

Returns the minute of the hour of the current or specified time.

**timeMinute(<argument>)**

Table 3- 94 Argument for TIMEMILLI

Argument	Description
<i>argument</i>	Specifies an expression that defines a specific time, or a function that returns a specific time.

```
timeMinute()=22
timeMinute(timeParse('06/21/2015 13:45:15','%m/%d/%Y %H:%M:%S'))=45
```

### 3.34.102 TIMEMINUTEOFDAY

Returns the minute of the day of the current or specified time.

**timeMinuteOfDay(<argument>)**

Table 3- 95 Argument for TIMEMINUTEOFDAY

Argument	Description
<i>argument</i>	Specifies an expression that defines a specific time, or a function that returns a specific time.

```
timeMinuteOfDay()=563
timeMinuteOfDay(timeParse('06/21/2015 13:45:15','%m/%d/%Y %H:%M:%S'))=885
```

### 3.34.103 TIMEPARSE

Returns a string that represents the time specified in the first argument.

**timeParse(*time*,<format>)**

Table 3- 96 Argument for TIMEPARSE

Argument	Description
<i>time</i>	Specifies a time specification or a function that returns a time specification.
<i>format</i>	Specifies a time format that is supported by the UNIX <code>strftime</code> function.

```
timeParse(timeString())=1421159135
timeParse('01/01/2015 00:00:00','%m/%d/%Y %H:%M:%S')=1420088400
```

### 3.34.104 TIMESECOND

Returns the second of the minute of the current or specified time.

**timeSecond**(*<argument>*)

Table 3- 97 Argument for TIMESECOND

Argument	Description
<i>argument</i>	Specifies an expression that defines a specific time, or a function that returns a specific time.

```
timeSecond()=37
timeSecond(timeParse('06/21/2015 13:45:15','%m/%d/%Y %H:%M:%S'))=15
```

### 3.34.105 TIMESTAMP

Returns the current time as a string.

**timeStamp**(*<format>*)

Table 3- 98 Argument for TIMESTAMP

Argument	Description
<i>format</i>	Specifies a time format that is supported by the UNIX <code>strftime</code> function.

```
timeStamp()=Thu Feb 19 15:09:35 2015
timeStamp('%m-%d-%Y')=02-19-2015
```

### 3.34.106 TIMESTRING

Returns the time represented by the first argument.

**timeString**(*time, <format>*)

Table 3- 99 Argument for TIMESTRING

Argument	Description
<i>time</i>	Specifies a time specification or a function that returns a time specification.
<i>format</i>	Specifies a time format. When you do not specify <i>format</i> , the system default time format is used.

```
timestring(timeCurrent())=Thu Feb 19 15:09:35 2015
timeString(timeCurrent(), '%m-%d-%Y')=02-19-2015
```

### 3.34.107 TIMESTRING

Returns the time represented by the first argument.

**timeString**(*time*,<*format*>)

Table 3- 100 Argument for TIMESTRING

Argument	Description
<i>time</i>	Specifies a time specification or a function that returns a time specification.
<i>format</i>	Specifies a time format. When you do not specify <i>format</i> , the system default time format is used.

```
timeString(timeCurrent())=Thu Feb 19 15:09:35 2015
timeString(timeCurrent(),'%m-%d-%Y')=02-19-2015
```

### 3.34.108 TIMESECONDOFDAY

Returns the second of the day of the current or specified time.

**timeSecondofDay**(<*argument*>)

Table 3- 101 Argument for TIMESECONDOFDAY

Argument	Description
<i>argument</i>	specifies an expression that defines a specific time, or a function that returns a specific time.

```
timeSecondOfDay()=34035
timeSecondOfDay(timeParse('06/21/2015 13:45:15','%m/%d/%Y %H:%M:%S'))=53115
```

### 3.34.109 TIMETODAY

Returns a value that represents the first second of the current day relative to local time.

**timeToday**()

```
timeToday()=1421125200
```

### 3.34.110 TIMEYEAR

Returns the number of years since 1900 of the current or specified time.

**timeYear(<argument>)**

Table 3- 102 Argument for TIMEYEAR

Argument	Description
<i>argument</i>	specifies an expression that defines a specific time, or a function that returns a specific time.

```
timeYear()=115
timeYear(timeParse('06/21/2013 13:45:15','%m/%d/%Y %H:%M:%S'))=113
```

**3.34.111 TOLOWER**

Converts the value of the argument to lowercase.

**toLower(*string*)**

Table 3- 103 Argument for TOLOWER

Argument	Description
<i>string</i>	Specifies a string.

```
toLower('Http://Www.Sas.Com/Products/Esp')=http://www.sas.com/products/esp
```

**3.34.112 TOUPPER**

Converts the value of the argument to uppercase.

**toUpper(*string*)**

Table 3- 104 Argument for TOUPPER

Argument	Description
<i>string</i>	Specifies a string.

```
toUpper('Http://Www.Sas.Com/Products/Esp')=HTTP://WWW.SAS.COM/PRODUCTS/ESP
```

**3.34.113 TRANSLATE**

For each character in the second argument, finds the corresponding characters in the first argument and replaces them with the corresponding characters in the third.

**translate**(*argument1*, *argument1*, *argument3*)

Table 3- 105 Argument for TRANSLATE

Argument	Description
<i>argument1</i> , <i>argument2</i> , <i>argument3</i>	Specifies a string. The length of <i>argument2</i> and <i>argument3</i> must be identical

```
translate('replace all vowels with its capital equivalent','aeiou','AEIOU') =rEplAcE
All vOwEls wIth Its cApItAl EqUIvAlEnt
```

### 3.34.114 TRUE

Returns true when the Boolean value of the argument is true. Otherwise, it returns false.

**true**(*argument*)

Table 3- 106 Argument for TRUE

Argument	Description
<i>argument</i>	Specifies one of the following: <ul style="list-style-type: none"> <li>• a literal value, either string or numeric. Enclose string values in single or double quotation marks.</li> <li>• a value to be resolved from an event field. Precede field names with the \$ character.</li> <li>• a value that refers to a resource. For example, when you use the <code>xpath</code> function to refer to an XML object named <code>myXML</code>, you specify this: <code>#myXML</code>.</li> <li>• a function.</li> </ul>

```
true('testing')=1
true('')=0
true(gt(10,5))=1
```

### 3.34.115 URLDECODE

Decodes the URL represented by the argument.

**urlDecode**(*argument*)

Table 3- 107 Argument for URLDECODE

Argument	Description
<i>argument</i>	Specifies a string.

For more information to encoding and decoding URLs, see this reference. [\(Insert Link\)](#)

```
urlDecode('http%3A%2F%2Fwww%2Esas%2Ecom%2Fproducts%2Fevent%20stream%20processing')
=http://www.sas.com/products/event stream processing
```

### 3.34.116 URLENCODE

Encodes the URL represented by the argument.

**urlEncode**(*argument*)

Table 3- 108 Argument for URLENCODE

Argument	Description
<i>argument</i>	Specifies a string.

For more information to encoding and decoding URLs, see this reference. [\(Insert Link\)](#)

```
urlEncode('http://www.sas.com/products/event stream processing')
=http%3a%2f%2fwww%2esas%2ecom%2fproducts%2fevent%20stream%20processing
```

### 3.34.117 XPATH

Parses the XML in the first argument, evaluating the second argument in the XML context.

**xpath**(*argument1*, *argument2*, <*argument3*>)

Table 3- 109 Argument for XPATH

Argument	Description
<i>argument1</i>	Specifies an instance of XML, represented by valid XML textual context or by a reference to XML elsewhere.
<i>argument2</i>	Specifies an evaluation string.
<i>argument3</i>	Specifies a separator used when the function returns multiple results.

```
xpath('<info><name>john smith</name><hobby>running</hobby>
<hobby>reading</hobby><hobby>golf</hobby></info>',
'./name/text()')=john smith
xpath(#myXml,'./hobby/text()','')=running,reading,golf
```



# Using Streaming Analytics

## 4.1 Overview

Edge Streaming Analytics Analytics enables you to use advanced analytical algorithms and machine learning techniques within an event stream processing project. Streaming analytics enables you to address common challenges with data from the Internet of Things (IoT):

- Lots of disparate variables
- Noisy or missing data
- Redundancy in the data
- Prediction of rare events

Common use cases for streaming analytics include the following:

- Preprocessing, transforming, or filtering data — determining how much and what data to send from the edge to the data center
- Detecting anomalies
- Monitoring system stability or degradation
- Processing unstructured text, audio, video, or image data in order to discern patterns or trends

Several analytical algorithms are packaged with Edge Streaming Analytics Analytics for use within event stream processing projects. Edge Streaming Analytics Analytics also supports a variety of offline models from various sources to use.

## 4.2 Streaming Analytics Window Types

### 4.2.1 Overview

Use the following window types within your event stream processing project to implement streaming analytics:

Table 4- 1 Streaming Analytics Window Types

Window Type	Description
Score	Score events with online analytical algorithms that are packaged with Edge Streaming Analytics or with algorithms in offline models.
Train	Refines the algorithm parameters of online models based on streaming event data.
Calculate	Transform data events using a variety of analytical algorithms.

Window Type	Description
Model Reader	Read models brought into Edge Streaming Analytics as analytic store files or as a combination of non-binary files.
Model Supervisor	Manage models received from Model Reader windows.

When you execute a streaming analytical algorithm within a Score, Train, or Calculate window, you must specify its name, parameter properties, and input and output mapping properties. To obtain this information for a particular window type and algorithm combination:

- use the **dfesp\_analytics** command-line utility.
- use HTTP requests to the Streaming Server through the RESTful API. For more information, see “Using the RESTful API”.

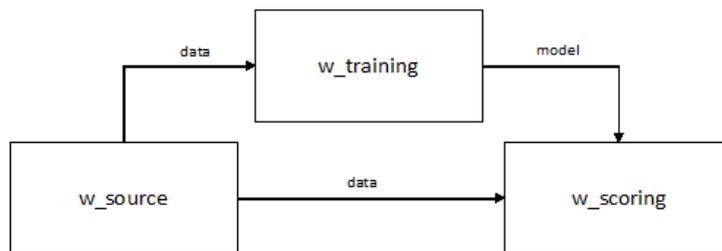
### 4.2.2 Edge Roles

Edges to or from any of the streaming analytics window types must specify a role that corresponds to the event type.

Table 4-2 Edge Roles

Role	Description
data	Sends data events between windows. A data event streams data to be processed by a streaming analytical algorithm into the receiving window.
model	Sends model events between windows. A model event, which has a fixed schema, streams model details into the receiving window.
request	Sends request events between windows. A request event, which has a fixed schema, requests that a specific action be performed within the receiving window.

Consider the following arrangement of windows, which is common to many training models:



The following code specifies roles for the edges between the windows:

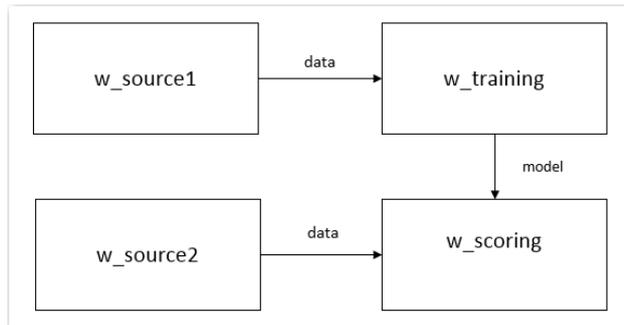
```

<edges>
<edge source='w_source' target='w_training' role='data' />
<edge source='w_source' target='w_scoring' role='data' />
<edge source='w_training' target='w_scoring' role='model' />
    
```

```
</edges>
```

- The first edge specifies that a data event originating from the Source window stream into the Train window.
- The second edge specifies that a data event originating from the Source window stream into the Score window.
- The third edge specifies that a model event originating from the Train window stream into the Score window.

Now consider an alternative arrangement of windows.



Here, one Source window streams data for training and a different Source window streams data for scoring. The Train window sends a refined model to the Score window. The following code specifies roles for the edges between the windows:

```

<edges>
  <edge source='w_source1' target='w_training' role='data' />
  <edge source='w_source2' target='w_scoring' role='data' />
  <edge source='w_training' target='w_scoring' role='model' />
</edges>

```

### 4.2.3 Determining Algorithm Availability and Properties

To determine what algorithms are available to a streaming analytics window type and what properties you set to use them, use the **dfesp\_analytics** command-line utility. The utility is located in **\$DFESP\_HOME/bin** in a UNIX environment and **%DFESP\_HOME%\bin** in a Windows environment.

Table 4- 3 Analytics Utility Commands

Command	Result
<b>dfesp_analytics</b>	Returns a list of the utility's available options
<b>dfesp_analytics --window_type</b>	Lists the algorithms available to the specified <i>window_type</i> : <b>train</b> , <b>score</b> , or <b>calculate</b> .
<b>dfesp_analytics --reader --model-Type recommender  astore</b>	Lists the properties that you can set within a Model Reader window for a recommender system or for a model provided in an analytic store file.
<b>dfesp_analyticswindow --algorithm algorithm</b>	For a specified algorithm, lists a window type's properties in text format.

Command	Result
<code>dfesp_analytics --window --algorithm algorithm -- xml</code>	For a specified algorithm, lists a window type's properties in XML format.
<code>dfesp_analytics --score --type recommender</code>	For <b>recommender</b> , lists the input-map and output-map of the model.
<code>dfesp_analytics --score --type astore --reference astore_file</code>	For a specified <code>astore_file</code> , lists the input-map and output-map of the trained model stored as a binary file with the analytic store format.
<code>dfesp_analytics --score --type astore --reference astore_file --options option(s)</code>	Set the specified option(s) for the trained model in the specified <code>astore_file</code> . It then lists the input- map and output-map associated with those specified options. You can specify a comma-separated list of key- value pairs for <i>options(s)</i> : <b>-options key1=value1:key2=value2...</b>

For example, when you submit `$DFESP_HOME/bin/dfesp_analytics --train` on the UNIX command line, you obtain the following output:

```
train algorithms:
DBSCAN
KMEANS
LinearRegression
SVM
LogisticRegression
```

When you submit `$DFESP_HOME/bin/dfesp_analytics --train --algorithm DBSCAN` on the UNIX command line, you obtain the following output:

```
parameters:
epsilon: double: (3.0)
mu: int64: (4)
beta: double: (0.3)
lambda: double: (0.02)
recluster: boolean: (1)
reclusterFactor: double: (2.0)
nInit: int64: (50)
velocity: int64: (1)
commitInterval: int64: (25)
input-map:
inputs: varlist: double ()
```

When you submit `$DFESP_HOME/bin/dfesp_analytics --train --algorithm DBSCAN --xml` on the UNIX command line, you obtain the following output:

```
<parameters>
<properties>
<property name='epsilon'>3.0</property>
<property name='mu'>4</property>
<property name='beta'>0.3</property>
<property name='lambda'>0.02</property>
<property name='recluster'>1</property>
<property name='reclusterFactor'>2.0</property>
<property name='nInit'>50</property>
<property name='velocity'>1</property>
<property name='commitInterval'>25</property>
</properties>
</parameters>
<input-map>
<properties>
<property name='inputs'></property>
```

```
</properties>
</input-map>
```

Suppose that you have created an analytic store file named **svddsstate.sasast** that contains a trained Support Vector Data Description (SVDD) model. When you submit **\$DFESP\_HOME/bin/dfesp\_analytics — score —type astore —reference svddsstate.sasast** on the UNIX command line from the directory that contains the analytic store file, you obtain output that looks like this:

```
input-map:
x1: double
x2: double
x3: double
x4: double
x5: double
x6: double
x7: double
x8: double
x9: double
x10: double
x11: double
x12: double
x13: double
x14: double
x15: double
x16: double
x17: double
x18: double
x19: double
x20: double
x21: double
x22: double
x23: double
output-map:
SVDDdistance : double (SVDD Distance)
SVDDscore : double (Label)
```

You can use information from the input map to help you code the input schema of the Source window that streams data to be scored. You can use information from the output map to help you code the schema for the Score window.

When you submit **\$DFESP\_HOME/bin/dfesp\_analytics —reader —modelType recommender** on the UNIX command line, you obtain this type of output:

```
parameters:
n: int32: (-1)
method: string: (SME)
itemTable: string: ()
itemTableDelimiter: string: (COMMA)
itemTableLineBreak: string: (LF)
userTable: string: ()
userTableDelimiter: string: (COMMA)
userTableLineBreak: string: (LF)
userRateInfo: string: ()
userRateInfoDelimiter: string: (COMMA)
userRateInfoLineBreak: string: (LF)
itemRateInfo: string: ()
itemRateInfoDelimiter: string: (COMMA)
itemRateInfoLineBreak: string: (LF)
ratingsByUser: string: ()
ratingsByUserDelimiter: string: (COMMA)
ratingsByUserLineBreak: string: (LF)
similarUsers: string: ()
similarUsersDelimiter: string: (COMMA)
similarUsersLineBreak: string: (LF)
```

These are the properties that you can set for recommender scoring within the Model Reader window.

When you submit `$DFESP_HOME/bin/dfesp_analytics --reader --modelType astore` on the UNIX command line, you obtain this type of output:

```
parameters:  
reference: string: ()
```

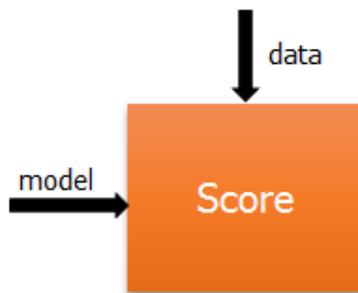
The *reference* property specifies the name and location of the analytic store file to use. No other parameters are listed in this output because those parameters cannot be predicted. They are specific to the model contained in the analytic store file.

Assume that you have an analytic store file named `rpca.sasast` that contains a trained Robust Principal Components Analysis model. The following command sets projection into the low rank space for the RPCA algorithm.

```
dfesp_analytics -score -type astore -reference rpca.sasast -options  
RPCA_PROJECTION_TYPE=1
```

### 4.3 Using Score Windows

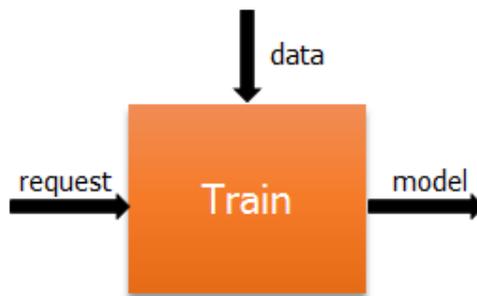
Score windows accept model events to make predictions for incoming data events. They generate scored data. You can use this score data to generate predictions based on the trained model. (No role is assigned to the outgoing edges, so they do not appear in the diagram.)



### 4.4 Using Train Windows

Train windows receive data events and publish model events to Score windows. They use incoming data events to develop and adjust model parameters in real time. Often, the data is historical data from which to learn patterns. Incoming data should contain both the outcome that you are trying to predict and related variables.

Train windows can also receive request events. These events can adjust the learning algorithm while events continue to stream.

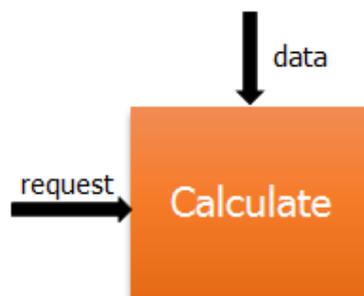


After a Train window has adjusted an algorithm, it writes the adjusted model to a Score window or a Model Supervisor window through a model event.

## 4.5 Using Calculate Windows

### 4.5.1 Overview to Calculate Windows

*Calculate windows* create real time, running statistics that are based on established analytical techniques. They receive data events and publishes newly transformed score data into output events. (No role is assigned to the outgoing edges, so they do not appear in the diagram.) Calculate windows can also receive request events.



Calculate windows are designed for data normalization and transformation methods, as well as for learning models that bundle training and scoring together.

### 4.5.2 Migrating from a Procedural Window to a Calculate Window

Support for SAS Micro Analytic Service (MAS) modules and stores has moved from the Procedural window to the Calculate window.

To migrate from a Procedural Window to a Calculate Window requires minimal change to your XML code.

Example Code A.1 Procedural Window

```
<window-procedural name='pw_01'>
```

```
...  
</window-procedural>  
<edge source='w_source' target='pw_01'/>  
Example Code A.2 Calculate Window  
<window-calculate name='pw_01' algorithm='MAS'>  
...  
</window-calculate>  
<edge source='w_source' target='pw_01' role='data'/>
```

You can use the `dfesp_xml_migrate` command to convert Procedural windows to Calculate windows in your model code. However, you must manually convert DS2 code in table server mode to code that uses SAS Micro Analytic Service modules before running the tool. For more information about this command, see "Migrating XML Code across Product Releases".

## 4.6 Using Model Reader Windows

In most cases, *Model Reader windows* receive request events that include the location and type of an offline model. Offline models are specified, developed, trained, and stored separately from the Streaming Server. Model Reader windows publish a model event that contains the model to Score windows or to Model Supervisor windows.



For more information, see "Using Offline Models".

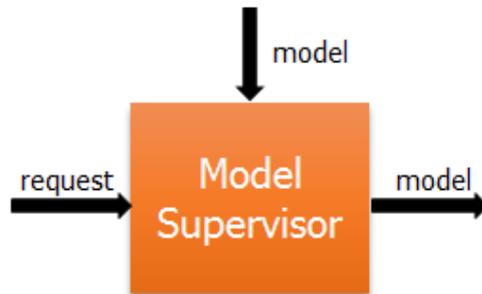
For recommender systems, you can specify offline model property values within the Model Reader window itself. The window publishes a model event based on those values.



For more information, see "Using Recommender Systems".

## 4.7 Using Model Supervisor Windows

*Model Supervisor windows* manage the flow of model events. Through input request events, you can control what model to deploy and when and where to deploy it. Model events are published to Score windows.



A Model Supervisor window can receive any number of model events. In a streaming analytics project, model events are typically sent by a Train window or a Model Reader window. After receiving a model event, a Model Supervisor window processes and publishes events to other streaming analytics windows based on the Model Supervisor window's deployment mode and on user requests.

The `<window-model-supervisor>` element has three properties:

- `name=` specifies the name of the window
- `deployment-policy=`

### **immediate**

sends model events to any receiving window immediately after receiving them

### **on-demand**

sends model events according to the requests specified at the command line

`capacity=` specifies the number of current model events to keep. After the capacity is met, older model events are discarded.

## 4.8 Understanding Event Types

### 4.8.1 Data Events

*Data events* stream input data to be processed by a receiving window. Data that feeds into a Score window is processed with machine learning algorithms specified by the incoming model event. The Score window produces data events consisting of scored data. Data that feeds into a Train window is applied to train the model specified by the incoming model event to produce a trained model. The data that feeds into a Calculate window serves as input into analytical techniques to produce real-time running statistics.

### 4.8.2 Model Events

*Model events* stream model metadata into a Score window or into a Model Supervisor window.

- A Train window adjusts the parameters of a model based on incoming data events. A Train window then streams the outcome of those changes through model events into a Score window.
- A Model Reader window publishes models to Score windows or Model Supervisor windows as specified by a request event.
- The Model Supervisor window controls how and when models are deployed to Score windows as specified by a request event.

You do not explicitly specify fixed model event schema; it is implicit for the query:

Table 4- 4 Fields in the Model Event Schema

Field	Description
model_id	Specifies a unique ID assigned by a Train window, Model Reader window, or a Model Supervisor window.
model_addr	Specifies the address of the model descriptor in memory.
model_origin	Specifies the window name from which the model was first created.
model_token	Specifies a token to determine the receiver of the model event.
timestamp	Specifies the timestamp when the model was first created.
model_perf	Specifies the performance metric of a model.

For example, with k-means clustering, a Train window receives data streams and computes and adjusts the centroids of the clusters. When the Score window receives the model event, it recovers the centroid information and applies it to the model. The Score window uses the centroid information to find the closest centroid for each of its incoming events, and assigns the cluster label accordingly.

### 4.8.3 Request Events

#### 4.8.4 Overview

Events that are transferred through request edges are called request events. You can use request events to initiate an action (for example, reconfigure a model). You can inject request events into a Source window with an adapter, or you can specify that they come from another window in the continuous query. Request events are generally slower than data events.

Specify the schema of a request event in the originating window. Request events have a fixed schema that consists of a `req_id` field, a `req_key` field, and a `req_val` field.

```
<schema>
  <fields>
    <field name='req_id' type='int64' />
    <field name='req_key' type='string' />
    <field name='req_val' type='string' />
  </fields>
</schema>
```

Request events can perform the following actions:

Table 4- 5 Actions Performed by Request Events

Action	Window	Description
<b>list</b>	Model Supervisor	Lists all available models currently stored.
<b>send</b>	Model Supervisor	Sends a model to a downstream window.
<b>remove</b>	Model Supervisor	Removes a model.
<b>load</b>	Source Model Reader	Loads an offline analytic store file into the event stream processing engine.
<b>reconfig</b>	Calculate, Train	Changes the value of a property.

The first request event should always specify **action** as the `req_key`. It should set the desired action (**list**, **send**, and so on) of the request as the `req_val`.

The last request event should always specify an empty `req_key`, which indicates the end of the request.

A request consists of a series of request events. The parameters of a request's action are specified in the request events between the first and last. In the middle request events, the values of `req_key` specify the names of parameters, and the values of `req_val` specify their values.

### 4.8.5 List Requests

Here is an example of a **list** request:

```
i,n,1,"action","list"
i,n,2,,
```

All events are Insert (Normal). After receiving the request, the Model Supervisor window sends multiple model events. Each model event represents an available model. The `model_addr` field of those model events are masked (that is, set to -1). Downstream windows (for example, Score windows) ignore these events.

After returning all available models, the model supervisor sends an event with `model_id` set to -1 to indicate the end of results. Thus, to get a list of available models, subscribe to the Model Supervisor window and capture the `model_id` fields.

### 4.8.6 Send Requests

The structure of a **send** request is as follows:

```
i,n,1,"action","send"
i,n,2,"modelId","USER_SPECIFIED_MODEL_ID"
i,n,3,"target","USER_SPECIFIED_TARGET"
i,n,4,,
```

All events are Insert (Normal). The `USER_SPECIFIED_MODEL_ID` is one previously obtained through a **list** request. The `USER_SPECIFIED_TARGET` can be the name of any window downstream of the Model Supervisor window.

### 4.8.7 Remove Requests

The structure of a **remove** request is as follows:

```
i,n,1,"action","remove"
i,n,2,"modelId","USER_SPECIFIED_MODEL_ID"
i,n,3,,
```

All events are Insert (Normal). The `USER_SPECIFIED_MODEL_ID` is one previously obtained through a **list** request. When you request to remove a model that has already been deployed to one window, the model is still valid and can continue to be used. However, the model can no longer be deployed to a new window. The model is removed permanently after no window uses it.

### 4.8.8 Load Requests

Send a **load** request to instruct a Model Reader window to load an analytic store file into an engine. All events are Insert (Normal).

```
i,n,1,"action","load"
i,n,2,"type","astore"
i,n,3,"reference","YOUR_ASTORE_FILE"
i,n,4,,
```

### 4.8.9 Processing Loaded Analytic Store Files with GPUs

To process a loaded analytic store file with a graphical processing unit (GPU), specify events before the **load** request that use special keywords.

```
i,n,1,USEGPUESP,1
i,n,2,NDEVICES,1
i,n,3,DEVICE0,1
i,n,4,"action","load"
i,n,5,"type","astore"
i,n,6,"reference","YOUR_ASTORE_FILE"
i,n,7,,
```

The following keywords control GPU usage.

Table 4- 6 Keywords for GPU Usage

Keyword	Description
USEGPUESP	Tell the Streaming Server to use GPUs to perform the calculation. GPUs must physically reside on the same computer system running the Streaming Server. Specify 1 to use GPUs, 0 not to use GPUs.
TENSORRTESP	Tell the Streaming Server to enable GPUs using Nvidia's high-performance TensorRT engine. When you specify this keyword, the analytic store mode is deployed in TensorRT format. After initially analyzing the network architecture, a reduced network is used to score with the GPUs. Note: Use TENSORRTESP instead of USEGPUESP only on Nvidia Jetson TX2 Modules.

Keyword	Description
NDEVICES	Specify how many GPUs to use to perform the calculation. When you do not specify NDEVICES, the Streaming Server uses all GPUs on the system. Important: It is recommended to assign a separate GPU per Score window to avoid contention. Note: If your project uses multiple deep learning models, then you should distribute each model to a separate GPU.
DEVICEn	Use with NDEVICES. Specify which GPU to use to perform the calculation. When using multiple GPUs, specify a list starting from 0. For example, suppose that your system has eight GPUs. You want to use two of them, with device IDs 3 and 5. <pre>i,n,4,USEGPUESP,1 i,n,5,NDEVICES,2 i,n,6,DEVICE0,3 l,n,7,DEVICE1,5 l,n,8,"action","load" ...</pre>

**Note**

Only deep learning models support the use of GPUs.

**4.8.10 Reconfig Requests**

Here is an example of a **reconfig** request:

```
i,n,1,"action","reconfig"
i,n,2,"arg1","val1"
i,n,3,"arg2","val2"
i,n,4,,
```

All events are Insert (Normal):

1. The first request event specifies **action** as the `req_key` and **reconfig** as the action to perform in the request.
2. The second request event specifies **arg1** as the `req_key` and **val1** as the `req_val`.
3. The third request event specifies **arg2** as the `req_key` and **val2** as the `req_val`.
4. The fourth request event has an empty `req_key`. This submits the `reconfig` request with `arg1=val1` and `arg2=val2`.

Callbacks that handle requests (for example, **reconfig** in the Calculate and Train windows and **read** in the Model Reader window) are not invoked by each request event. They are invoked by each request.

## 4.8.11 Example

The following model specifies **request** for the edge role between a Source window named `w_request` and a Calculate window named `w_calculate`:

```
<windows>
...
<window-source name='w_request'>
<schema>
<fields>
<field name='req_id' type='int64' key='true' />
<field name='req_key' type='string' />
<field name='req_val' type='string' />
</fields>
</schema>
</window-source>

<window-calculate name='w_calculate' algorithm='Summary'>
<schema>
...
</schema>
<parameters>
<properties>
<property name="windowLength">4</property>
</properties>
</parameters>
...
</window-calculate>
</windows>
...
<edges>
...
<edge source='w_request' target='w_calculate' role='request' />
</edges>
```

By injecting the following events into the `w_request` Source window, you send a **reconfig** event to the `w_calculate` window. The request changes the value of `windowLength` from 4 (as defined in the properties of `w_calculate`) to 100.

```
i,n,1,"action","reconfig"
i,n,2,"windowLength","100"
i,n,3,,
```

## 4.9 Using Online Models

### 4.9.1 Overview

Online models use algorithms that are packaged with Edge Streaming Analytics Analytics and that are trained in Edge Streaming Analytics projects.

---

**Note**

When you persist an online model, note that the Train window immediately starts training the new model upon model restore. No user intervention is required. For more information, see "Implementing Persist and Restore Operations".

---

For each model, input maps are tuples of (*Name, DataType, DefaultValue*) that map incoming event fields to an algorithm's variable values. Output maps are tuples of (*Name, DataType, DefaultValue*) that map the output fields that are produced by an algorithm to write variable values.

The following algorithms are packaged with Edge Streaming Analytics Analytics.

Table 4- 7 Algorithms for Train and Score Windows

Window	General Uses
Streaming K-Means Clustering	K-Means clustering is an iterative algorithm that partitions data into non-overlapping groups based on their similarity. You can use it to discover hidden structures within the data and to detect outliers. Common uses include market segmentation, image segmentation, and image compression.
Streaming DBSCAN Clustering	DBSCAN clustering is an unsupervised learning method to distinguish clusters of high density from clusters of low density. You can use DBSCAN clustering to cluster location data in order to identify where particular events occur.
Streaming Linear Regression	Streaming linear regression is an approximation of the standard linear regression model that is appropriate for streaming data.
Streaming Logistic Regression	Streaming logistic regression is an approximation of the standard logistic regression model that is appropriate for streaming data.
Streaming Support Vector Machines	Support vector machines are supervised learning models with associated algorithms. Support vector machines apply classification and regression analysis on incoming data. You supply training examples and mark them as belonging to a category. A support vector machine builds a model that assigns new examples to that category.
Streaming t-Distributed Stochastic Neighbor Embedding	t-Distributed Stochastic Neighbor Embedding (t-SNE) is a machine learning algorithm for dimensionality reduction that is used to visualize high-dimensional data sets. It is nonparametric and non-linear. t-SNE renders high-dimensional objects into two- or three-dimensional points, making them more suitable for human observation.

Table 4- 8 Algorithms for Calculate Windows

Window	General Uses
Streaming Summary (Univariate Statistics)	Streaming summary enables you to calculate univariate statistics (N, number of missing observations, sum, mean, and so on) on a sliding window. You can use it for data exploration and for data monitoring where you send the calculated statistics downstream to trigger action.
Streaming Pearson Correlation	Streaming Pearson correlation enables you to calculate the Pearson correlation coefficient between two variables in a sliding window. You can use it to determine the similarity or dissimilarity of the variables. This can be useful to detect when two matched signals become unmatched.
Segmented Correlation	Segmented correlation is similar to autocorrelation. It specifies the correlation between the elements of a series and others from the same series that are separated from them by a specified interval. You can use segmented correlation to find repeating patterns, such as the occurrence of a signal obscured by noise.
Streaming Distribution Fitting	The streaming distribution fitting algorithm fits a Weibull, Gamma, or Normal distribution to a series of data points in a sliding window. You can use the parameters from the fitting in downstream data monitoring.
Short-Time Fourier Transform (STFT)	STFT is commonly used to monitor the time-varying frequency content of a signal. It can be used to detect anomalies in a continuous stream of data, such as machine vibrations. Abnormal conditions can lead to changes in a vibration signal. STFT can be used to monitor the signal frequency band of interest. This monitoring can enable early detection of machine faults and thus lead to more efficient machine maintenance.
Compute Fit Statistics	The goodness of fit of a statistical model describes how well a model fits a set of data. Goodness-of-fit measures summarize the difference between observed values and predicted values of the model under consideration. These measures have a very broad range of practical applications.
Compute ROC Information	Receiver operating characteristic (ROC) information shows the diagnostic ability of a classifier system as you vary its discrimination threshold. You create ROC information by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings.
Calculate a Streaming Histogram	A histogram graphically represents a distribution of numerical data. This algorithm processes numerical data and puts it into bins in order to generate boundaries for a histogram to fit it.
Streaming Text Tokenization	Tokenization splits text into tokens, such as paragraphs, sentences, or individual words. You can submit the results of tokenization to further analysis to detect or predict patterns.

Window	General Uses
Streaming Text Vectorization	Vectorizing text creates maps from words or n-grams to a vector space. A vector space is an algebraic model to represent text documents as vectors of identifiers (for example, index terms).
Image Processing Algorithm	Use the image processing algorithm to process streaming image data and manipulate it in real time.
Moving Relative Range	The moving relative range (MRR) provides a measure of volatility for a nonstationary time series, where the mean and the variance of the series change over time. For example, you could use MRR to detect electrical disturbances in the power grid.
Term Frequency — Inverse Document Frequency (TFIDF)	TFIDF is a weight that shows how important a particular word is to a document in a document collection. Deriving this weight is often a necessary step before more advanced analytics such as clustering or classification. You can use TFIDF to analyze Amazon Reviews, Google News, and Twitter feeds.
Lag Monitoring	The lag monitoring algorithm computes the cross-correlation between a target time series and one or more additional time series. Results contain the selected lags and computed cross-correlation values that correspond to minimum, maximum, and maximum absolute value cross-correlations for each of the variables.
Cepstrum Transform	Cepstral analysis can be used to find out whether a signal contains periodic elements in seismic, speech, and radar signal processing. This method is very effective in digital speech processing to detect the pitch in the human speech signal and extract the transfer function of the vocal tract in voiced speech.
Video Encoding	Use the video encoding algorithm to parse image data in BayerRG8 format to a more common image format. JPEG is the default output format, and PNG is also supported.
Subspace Tracking (SST)	<p>SST is a method to detect anomalies and system degradation in systems that generate high-frequency, high-dimensional data. Examples include monitoring the following:</p> <ul style="list-style-type: none"> <li>• The lighting system in a smart campus to find defective flood lights</li> <li>• solar farm system to detect a defective panel</li> <li>• A credit card system to find a fraudulent transaction</li> </ul>
Streaming Audio Feature Computation	Audio feature computation is the process of applying an algorithm to an audio signal in order to convert it into a sequence of acoustic feature vectors. These vectors contain a serviceable numerical representation of the audio signal. You use the output of this analysis in order to perform streaming speech transcription.

Window	General Uses
Streaming Speech Transcription	Use the streaming speech transcription algorithm to transcribe the output of the acoustic model generated by streaming audio feature computation.
Change Detection	With change detection, a stream of measures is monitored and an alert is raised when values deviate from what is expected. This algorithm can be used for real-world acoustic event detection for surveillance or multimedia information retrieval.

Calculate windows analyze values in input events according to the specified algorithm and publish analysis results to write variable values in output events. In most cases, for each event that streams into a Calculate window, there is a single corresponding output event that contains those results. There are three exceptions:

Table 4-9 Exceptions to One-to-One Correspondence Between Input and Output Events for Calculate Windows

Algorithm	Number of Output Events per Input Event
Text Tokenizer	One for each word token that is produced ( <code>wordOut</code> ).
Short Time Fourier Transformation	The specified number of frequency bins ( <code>binsInSchema</code> ).
Compute ROC Information	The result of dividing 1 by the bin width ( <code>cutStep</code> ). The default value of <code>cutStep</code> is 0.01. Thus, by default you have 100 output events per input event.

Calculate windows that use algorithms with sliding windows do not produce output events until the total number of input events is equal to the length of the sliding window.

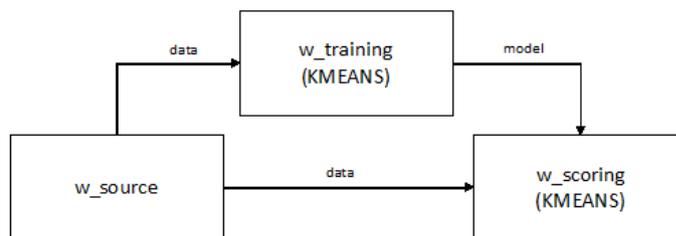
## 4.9.2 Training and Scoring with K-means Clustering

The classic k-means clustering algorithm performs two basic steps:

1. An assignment step in which data points are assigned to their nearest cluster centroid
2. An update step in which each cluster centroid is recomputed as the average of data points belonging to the cluster

The algorithm runs these two steps iteratively until a convergence criterion is met.

Consider the following example:



This continuous query includes the following:

- a Source window that receives the data to be scored
- a Train window that generates and periodically updates the k-means model
- a Score window that performs the scoring

The Source window `w_source` receives a data event. The input stream is placed into three fields for each observation: an ID that acts as the data stream's key, named `id`; an x coordinate of data named `x_c`; and a y coordinate of data named `y_c`.

```
<window-source name='w_source' insert-only='true'>
<schema>
<fields>
<field name='id' type='int64' key='true' />
<field name='x_c' type='double' />
<field name='y_c' type='double' />
</fields>
</schema>
<connectors>
...
</connectors>
</window-source>
```

The Train window `w_training` looks at all observations and periodically generates a new clustering model using the k-means algorithm. Generated clustering model events are published to the Score window `w_score`, where incoming events are clustered.

```
<window-train name='w_training' algorithm='KMEANS'>
<parameters>
<properties>
<property name="nClusters">2</property>
<property name="initSeed">1</property>
<property name="dampingFactor">0.8</property>
<property name="fadeOutFactor">0.05</property>
<property name="disturbFactor">0.01</property>
<property name="nInit">50</property>
<property name="velocity">5</property>
<property name="commitInterval">25</property>
</properties>
</parameters>
<input-map>
<properties>
<property name="inputs"><![CDATA[x_c,y_c]]></property>
</properties>
</input-map>
</window-train>
```

The following properties govern the k-means algorithm in the Train window:

Table 4- 10 Parameters

Name	Variable Type	Required or Optional?	Default Value	Description
nClusters	int32	Optional	2	Specifies the number of clusters $K$ $K > 0$ to report.
initSeed	int32	Optional	12345	Specifies the random seed used during initialization when each point is assigned to a random cluster.
dampingFactor	double	Optional	0.8	Specifies the damping factor $\alpha$ $0 < \alpha < 1$ for old data points. If the current time is $T$ , data points arriving at time $T$ would have weight 1, and data points arriving at time $T - t$ would have weight $\alpha^t$ .
fadeOutFactor	double	Optional	0.05	Specifies the factor $\theta$ $0 < \theta < 1$ for determining whether an existing cluster is fading out. If a cluster weight is smaller than the maximal cluster weight among other clusters multiplied by $\theta$ , then this cluster is considered to be fading out.
disturbFactor	double	Optional	0.01	Specifies the factor $\delta$ $\delta > 0$ for the disturbance when splitting a cluster. When an old cluster fades out, the cluster with the maximal weight is split into two, and both new clusters share half of its weight. If the old-centroid is $\vec{c}$ , the two new centroids are $(1 + \delta) \cdot \vec{c}$ and $(1 - \delta) \cdot \vec{c}$ , respectively.
nInit	int64	Optional	50	Specifies the number of data events used during initialization.

Name	Variable Type	Required or Optional?	Default Value	Description
velocity	int64	Optional	1	Specifies the number of events arriving at a single timestamp.
commitInterval	int64	Optional	25	Specifies the number of timestamps to elapse before committing a model to downstream scoring.

Table 4- 11 Input Mapping

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
inputs	varlist	double	Required	No default value	Specifies the list of variable names used in clustering. Variable names are defined in the input schema of the Source window, and they are separated by a comma in the list.

The Score window `w_scoring` assigns a cluster number to each input event. The cluster number indicates which cluster the observation falls into according to the k-means clustering algorithm.

```

<window-score name='w_scoring'>
  <schema>
    <fields>
      <field name='id' type='int64' key='true' />
      <field name='x_c' type='double' />
      <field name='y_c' type='double' />
      <field name='seg' type='int32' />
      <field name='min_dist' type='double' />
      <field name='model_id' type='int64' />
    </fields>
  </schema>
  <models>
    <online algorithm='KMEANS'>
      <input-map>
        <properties>
          <property name="inputs">
            <![CDATA[id, x_c, y_c, seg, min_dist, model_id]]>
          </property>
        </properties>
      </input-map>
    </online>
  </models>
</window-score>

```

```
</models>
</window-score>
```

The following properties are unique to Score windows for streaming k-means clustering:

Table 4- 12 Input Mapping

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
inputs	varlist	double	Optional	""	Specifies the list of variable names used in clustering. Variable names are defined in the input schema of the Source window, and they are separated by a comma in the list.

Table 4- 13 Output Mapping

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
labelOut	variable	int32	Optional	""	Specifies the output variable in the output schema that stores the cluster label.
minDistanceOut	variable	double	Optional	""	Specifies the output variable in the output schema that stores the distance to the nearest cluster. If not specified, the minimal distance column is not shown.
modelIdOut	variable	int64	Optional	""	Specifies the output variable in the output schema that stores the ID of the model from which the score is computed. If not specified, the model ID column is not shown.

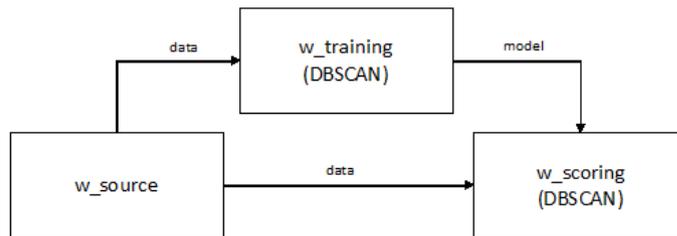
The edges are defined at the end of the project. Streaming analytics windows require a role for each incoming edge.

```
<edges>
<edge source='w_source' target='w_training' role='data' />
<edge source='w_source' target='w_scoring' role='data' />
<edge source='w_training' target='w_scoring' role='model' />
</edges>
```

You can view the default values of the k-means algorithm parameter properties for the Score and Train windows with the command-line utility.

### 4.9.3 Training and Scoring with DBSCAN Clustering

*DBSCAN* is a density-based clustering approach. Given a set of data points, the algorithm tries to find connected high-density regions as clusters. To do that, it searches for a core point where the number of neighbors in its  $\epsilon$  range is greater than or equal to  $\mu$ . If such a core point exists, the algorithm visits its neighbors. If a neighbor point is also a core point, then the point is further extended. Otherwise, no more core points can be reached, and the algorithm starts with an unvisited core point and repeats the previous process until all points are visited. In the end, all points (core and non-core) that are reachable from a given core point form a cluster. Consider the following example:



This continuous query includes the following:

- a Source window that receives data events that stream the data to be scored
- a Train window that generates and periodically updates the DBSCAN model
- a Score window that performs the scoring

The Source window `w_source` receives a data event. The input stream is placed into three fields for each observation: an ID that acts as the data stream's key, named `id`; an x coordinate of data named `x_c`, and a y coordinate of data named `y_c`.

```

<window-source name='w_source'>
<schema>
<fields>
<field name='id' type='int64' key='true'/>
<field name='x_c' type='double'/>
<field name='y_c' type='double'/>
</fields>
</schema>
<connectors>
...
</connectors>
</window-source>
  
```

The Train window `w_training` processes all observations and periodically generates a new clustering model using the DBSCAN algorithm.

```

<window-train name='w_training' algorithm='DBSCAN'>
<parameters>
<properties>
<property name="epsilon">2.0</property>
<property name="mu">3</property>
<property name="beta">0.5</property>
  
```

```

<property name="lambda">0.05</property>
<property name="recluster">1</property>
<property name="reclusterFactor">2.75</property>
<property name="nInit">50</property>
<property name="velocity">5</property>
  <property name="commitInterval">25</property>
</properties>
</parameters>
  <input-map>
<properties>
<property name="inputs"><![CDATA[x_c,y_c]]></property>
</properties>
</input-map>
</window-train>
    
```

The following properties govern the DBSCAN algorithm in the Train window:

Table 4- 14 Parameters

Name	Variable Type	Required or Optional?	Default Value	Description
epsilon	double	Optional	3.0	Specifies the range of the neighborhood being considered. ( $\epsilon > 0$ )
mu	int64	Optional	4	Specifies the weight of core micro clusters. ( $\mu > 1$ )
beta	double	Optional	0.3	Specifies the factor for $\mu$ to determine a micro cluster is p-mc or o-mc. ( $0 < \beta \leq 1$ ) and ( $\beta \cdot \mu > 1$ )
lambda	double	Optional	0.02	Specifies the decaying factor for the data weight. Assuming that the current time is T, data points arriving at time T have weight 1, and data points arriving at time T - have weight $2^{-\lambda \cdot t}$ ( $\lambda > 0$ ).
recluster	Boolean	Optional	1	Specifies whether re-clustering (with offline weighted DBSCAN) is performed: valid values are 1 for true and 0 for false.
reclusterFactor	double	Optional	2.0	Specifies the factor (c) for $\epsilon$ used in re-clustering. ( $c \geq 2$ ).
nInit	int64	Optional	50	Specifies the number of data events used during initialization.

Name	Variable Type	Required or Optional?	Default Value	Description
velocity	int64	Optional	1	Specifies the number of events arriving at a single timestamp.
commitInterval	int64	Optional	25	Specifies how many timestamps should elapse before sending a model to downstream scoring.

Table 4- 15 Input Mapping

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
input	varlist	double	Required	No default value	Specifies the list of variables used in clustering. Variables are defined in the input schema, and they are separated by a comma in the list.

Generated clustering models are published to the Score window `w_scoring`. This window assigns a cluster number to each input event. The cluster number indicates which cluster the observation falls into according to the DBSCAN algorithm.

```
<window-score name='w_scoring'>
  <schema>
    <fields> <field name='id' type='int64' key='true' />
    <field name='x_c' type='double' />
    <field name='y_c' type='double' />
    <field name='seg' type='int32' />
    <field name='min_dist' type='double' />
    <field name='model_id' type='int64' />
  </fields>
</schema>
<models>
  <online algorithm='DBSCAN'>
    <input-map>
      <properties>
        <property name="inputs">
          <![CDATA[id, x_c, y_c, seg, min_dist, model_id]]>
        </property>
      </properties>
    </input-map>
  </online>
</models>
</window-score>
```

The following properties are unique to Score windows for streaming DBSCAN clustering:

Table 4- 16 Input Mapping

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
inputs	varlist	double	Optional	""	Specifies the list of variable names used in clustering. Variable names are defined in the input schema of the Source window, and they are separated by a comma in the list.

Table 4- 17 Output Mapping

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
labelOut	variable	int32	Optional	""	Specifies the output variable name in the output schema that stores the cluster label.
minDistanceOut	variable	double	Optional	""	Specifies the output variable name in the output schema that stores the distance to the nearest cluster. If not specified, the minimal distance column is not shown.
modelIdOut	variable	int64	Optional	""	Specifies the output variable name in the output schema that stores the ID of the model from which the score is computed. If not specified, the model ID column is not shown.

The edges are defined at the end of the project. Streaming analytics windows require a role for each edge.

```
<edges>
<edge source='w_source' target='w_training' role='data' />
<edge source='w_source' target='w_scoring' role='data' />
<edge source='w_training' target='w_scoring' role='model' />
</edges>
```

You can view the default values of the DBSCAN algorithm parameter properties for the Score and Train windows with the command-line utility.

#### 4.9.4 Training and Scoring with Streaming Linear Regression

Linear regression models the relationship between a scalar dependent variable and one or more explanatory variables (independent variables). Streaming linear regression (LinearRegression) is an approximation of the standard linear regression model that is appropriate for streaming data.

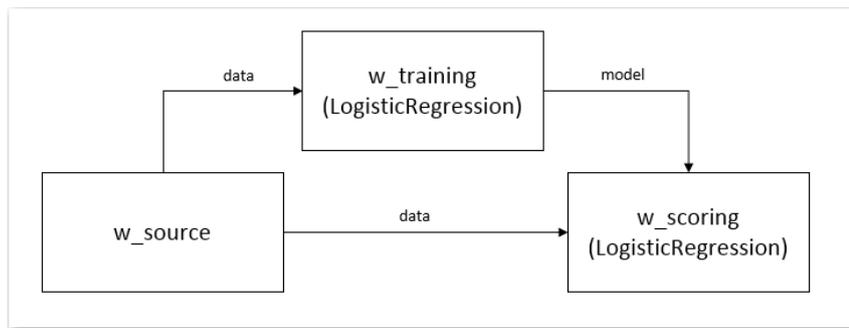
The basic linear regression model is  $Y = X\beta + e$ . Here,  $Y$  is the target vector,  $X$  is the data matrix,  $\beta$  is the vector of parameters, and  $e$  is the vector of errors. For a static data set, one way to solve for  $\beta$  is to use coordinate descent. After solving for  $\beta$ , you obtain an estimated parameter vector  $\beta$ . You can then use that vector with any new  $X$  data matrix in order to make predictions for the values of  $Y$ , that is  $Y$ .

In the streaming linear regression model, you must make predictions based on an ever-changing stream of data. After you obtain a specified number of events, `ninit`, you solve for the  $\beta$ s. When you reach a number of events that is equal to the `commitInterval`, you publish the most recent  $\beta$ s to the Score window. The Score window uses them to score all of the data published to it.

Now you have a model, but more events are streaming in. Use `batchSize` to define how many new events are required before you update the model. After the number new events is equal to `batchSize`, solve for the  $\beta$ s again. You do not use all of the data from the beginning, as this could get prohibitively large with time. Instead, you essentially combine the existing  $\beta$ s with the  $\beta$ s obtained from training the new batch of data.

Use the `dampingFactor` to define how much the old  $\beta$ s affect the new  $\beta$ s. After the number of new events since the last model was published to score window equals the `commitInterval`, you publish the most recent set of  $\beta$ s the Score window. Use the  $\beta$ s to score all data passed to the window. Repeat this process as needed. Instead of storing all the data and repeatedly training the data statically, you store only the previous parameter's  $\beta$ . This is the general goal of streaming algorithms, find a way to not store all previous data but remember what you learned from it to score data.

Consider the following example:



This continuous query includes the following:

- a Source window that receives data events that stream the data to score
- a Train window that generates and periodically updates the linear regression model
- a Score window that performs the scoring

The Source window `w_source` receives a data event. The input stream is placed into three fields for each observation: an ID that acts as the data stream's key, named `id`; a `y` coordinate of data named `y`; and 784 `x` coordinates.

```

<window-source name='w_source'>
<schema>
<fields>
<field name='id' type='int64' key='true' />
<field name='y' type='double' />
<field name='x1' type='double' />
... 1
<field name='x784' type='double' />
</fields>
</schema>
<connectors>
...
</connectors>
</window-source>
  
```

An ellipsis indicates that field name values range from `x1` to `x784`, inclusive.

The Train window `w_training` looks at all the observations and periodically generates a new model using the linear regression algorithm. Model events are published to the Score window `w_scoring`.

```

<window-train name='w_training' algorithm='LinearRegression'>
<parameters>
<properties>
<property name="nInit">60000</property>
<property name="commitInterval">10000</property>
<property name="dampingFactor">1</property>
  <property name="centerFlag">0</property>
<property name="scaleFlag">0</property>
</properties>
</parameters>
<input-map>
<properties>
  
```

```

    <property name="inputs">y,x1,...,x784</property> 1
  <property name="target">y</property>
</properties>
</input-map>
</window-train>

```

An ellipsis indicates that input values range from x1 to x784, inclusive.

The following properties govern the linear regression algorithm in the Train window:

Table 4- 18 Parameters

Name	Value Type	Required or Optional?	Default Value	Description
nInit	int64	Optional	50	Specifies the number of data events used during initialization. The specified value must be a positive integer.
commitInterval	int64	Optional	10	Specifies the number of data events to process before triggering a commitment of the model to downstream scoring. The specified value must be a positive integer.
batchSize	int64	Optional	0	Specifies the batch size in processing the training samples. The specified value must be a positive integer. This property affects how much memory is used to buffer data events. If you have sufficient memory, set this to the maximum of nInit and commitInterval.
dampingFactor	double	Optional	1	Specifies the damping factor $\alpha$ ( $0 \leq \alpha \leq 1$ ) for old data points. That is, if the current number of data events to process before triggering a commitment of the model is T, data points arriving at T would have weight 1. Data points at T — t would have weight $\alpha^t$ .

Name	Value Type	Required or Optional?	Default Value	Description
centerFlag	int64	Optional	0 (false)	Specifies whether to center the data (dense part) based on the first <code>batchSize</code> data events of the initialization. Specifically, the mean is computed with the first <code>bufferSize</code> data events of the initialization, and each data event is subtracted with the computed mean.
scaleFlag	int64	Optional	0 (false)	Specifies whether to scale the data (dense part) based on the first <code>batchSize</code> data events of the initialization. Data is scaled so that the variance of the first <code>batchSize</code> number of data events is 1.
maxSparseIndex	int64	Optional	0	Specifies the number of predictor variables contained in the sparse variable, if it exists. The value should be a nonnegative integer. <b>Note:</b> Sparse linear regression problems generate dense matrix calculations that often cannot be completed fast enough for streaming data applications. Thus, you should set a small value of <code>maxSparseIndex</code> when you expect problems to be solved in a reasonable time for a streaming data application.
inputs	varlist	Required	No default value	Specifies the list of variable names used in classification. Variable names are defined in the input schema, and they are separated by a comma in the list (for example, <code>x,y</code> ).

Name	Value Type	Required or Optional?	Default Value	Description
target	variable	Optional	""	Specifies the target response variable. For linear regression, this must be a continuous variable.
sparse	variable	Optional	""	Specifies the sparse variable.

The Score window `w_scoring` scores the data.

```

<window-score name='w_scoring'>
  <schema>
    <fields>
      <field name='id' type='int64' key='true' />
      <field name='y' type='double' />
      <field name='yPredictOut' type='double' />
      <field name='modelIdOut' type='int64' />
    </fields>
  </schema>
  <models>
    <online algorithm='LinearRegression'>
      <input-map>
        <properties>
          <property name="inputs">y,x1,...,x784</property>
        </properties>
      </input-map>
      <output-map>
        <properties>
          <property name='yPredictOut'>yPredictOut</property>
          <property name='modelIdOut'>modelIdOut</property>
        </properties>
      </output-map>
    </online>
  </models>
</window-score>

```

The following properties govern the linear regression algorithm in the Score window:

Table 4- 19 Input Mapping

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
inputs	varlist	double   string	Optional	""	Specifies the list of variable names used in classification. Variable names are defined in the input schema, and they are separated by comma in the list (for example, x,y).

Table 4- 20 Output Mapping

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
yOut	variable	double	Optional	""	Specifies the original response. If not specified, it is not displayed.
yPredictOut	variable	double	Optional	""	Specifies the predicted response. If not specified, it is not displayed.
modelIdOut	variable	int64	Optional	""	Specifies the column or field name in the output schema that stores the ID of the model from which the score is computed. If not specified, it is not displayed.

The edges are defined at the end of the project. Streaming analytics windows require a role for each incoming edge.

```
<edges>
<edge source='w_data' target='w_train' role='data' />
<edge source='w_data' target='w_score' role='data' />
```

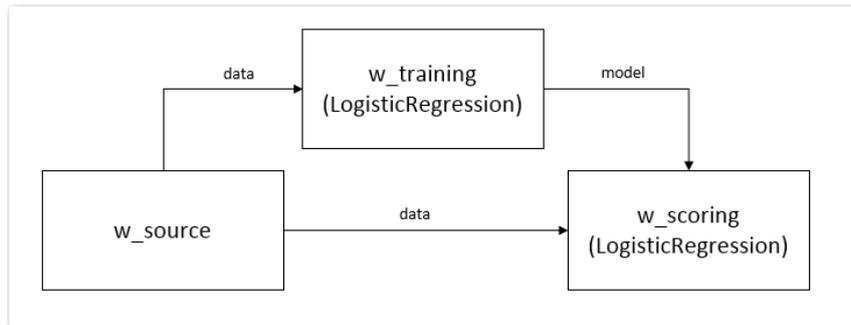
```
<edge source='w_train' target='w_score' role='model' />
</edges>
```

You can view the default values of the logistic regression algorithm parameter properties for the Score and Train windows with the command-line utility.

### 4.9.5 Training and Scoring Streaming Logistic Regression

With logistic regression, the dependent variable is categorical. Some logistic regression models use a binary dependent variable (alive or dead, yes or no, win or lose) and others use a dependent variable with more than two outcome categories. When the dependent variable has more than one outcome category, it is converted to a binary classification problem by choosing a positive class and treating all other classes as one. Streaming logistic regression (LogisticRegression) is an approximation of the standard logistic regression model that is appropriate for streaming data.

Consider the following example:



This continuous query includes the following:

- a Source window that receives data events that stream the data to score
- a Train window that generates and periodically updates the logistic regression model
- a Score window that performs the scoring

The Source window `w_source` receives a data event. The input stream is placed into three fields for each observation: an ID that acts as the data stream's key, named `id`; a `y` coordinate of data named `y`; and 784 `x` coordinates.

```
<window-source name='w_source'>
<schema>
<fields>
<field name='id' type='int64' key='true' />
<field name='y' type='double' />
<field name='x1' type='double' />
... 1
<field name='x784' type='double' />
</fields>
</schema>
<connectors>
...
</connectors>
</window-source>
```

1 An ellipsis indicates that field name values range from x1 to x784, inclusive.

The Train window `w_training` looks at all the observations and periodically generates a new model using the logistic regression algorithm. Model events are published to the score window `w_scoring`.

```
<window-train name='w_training' algorithm='LogisticRegression'>
  <parameters>
  <properties>
    <property name="nInit">60000</property>
    <property name="commitInterval">10000</property>
    <property name="dampingFactor">1</property>
    <property name="c">1</property>
    <property name="centerFlag">0</property>
    <property name="scaleFlag">0</property>
    <property name="maxSparseIndex">0</property>
    <property name="numC">5</property>
    <property name="ratioC">4</property>
    <property name="choose">-1</property>
    <property name="randSeed">123</property>
    <property name="positiveClass">8</property>
    <property name="augmentedValue">1</property>
    <property name="outerIterMax">10</property>
  </properties> </parameters> <input-map>
  <properties>
    <property name="inputs">y,x1,...,x784</property> 1
    <property name="target">y</property>
  </properties>
</input-map>
</window-train>
```

1 An ellipsis indicates that input values range from x1 to x784, inclusive.

The following properties govern the linear regression algorithm in the Train window:

Table 4- 21 Parameters

Name	Value Type	Required or Optional?	Default Value	Description
nInit	int64	Optional	50	Specifies the number of data events used during initialization. The specified value must be a positive integer.
commitInterval	int64	Optional	10	Specifies the number of data events to process before triggering a commitment of the model to downstream scoring. The specified value must be a positive integer.

Name	Value Type	Required or Optional?	Default Value	Description
<code>batchSize</code>	int64	Optional	0	Specifies the batch size in processing the training samples. The specified value must be a positive integer. This property affects how much memory is used to buffer data events. If you have sufficient memory, set this to the maximum of <code>nInit</code> and <code>commitInterval</code> .
<code>dampingFactor</code>	double	Optional	1	Specifies the damping factor $\alpha$ ( $0 \leq \alpha \leq 1$ ) for old data points. That is, if the current number of data events to process before triggering a commitment of the model is $T$ , data points arriving at $T$ would have weight 1. Data points at $T - t$ would have weight $\alpha^t$ .
<code>centerFlag</code>	Boolean	Optional	0	Specifies whether to center the data (dense part) based on the first <code>batchSize</code> data events of the initialization. Specifically, the mean is computed with the first <code>bufferSize</code> data events of the initialization, and each data event is subtracted with the computed mean.
<code>scaleFlag</code>	Boolean	Optional	0	Specifies whether to scale the data (dense part) based on the first <code>batchSize</code> data events of the initialization. Data is scaled so that the variance of the first <code>batchSize</code> number of data events is 1.
<code>maxSparseIndex</code>	int64	Optional	0	Specifies the number of variables contained in the sparse variable, if it exists. Specify a nonnegative integer.

Name	Value Type	Required or Optional?	Default Value	Description
c	double	Optional	0	Specifies the regularization parameter. The specified value must be positive.
numC	int64	Optional	1	Specifies the number of regularization parameters to try. The specified value must be positive.
ratioC	double	Optional	10	Specifies the ratio in setting the set of regularization parameters.
choose	double	Optional	-2	Specifies the criterion in selecting the best regularization parameter. If choose=-2, the c that achieves the smallest misclassification error is used. If choose=-1, the c that achieves the smallest hinge loss is used. If choose is nonnegative, the c that achieves the largest choose score is used.
randSeed	int64	Optional	123	Specifies the random seed in reshuffling data events. Specify a positive value. If randSeed=0, the data in the buffer is not reshuffled. If randSeed>0, the data in the buffer is implicitly reshuffled with the corresponding random seed.
positiveClass	double	Optional	1	Specifies the value of the response that is treated as the positive class.
augmentedValue	double	Optional	1	Specifies the augmented value for handling the intercept. The specified value must be positive.

Name	Value Type	Required or Optional?	Default Value	Description
outerIterMax	int64	Optional	1	Specifies the number of outer iterations used in coordinate descent for non- initialization data events. The specified value should be positive. It determines the precision of the solution with data events outside the initialization step.
outerIterMaxInit	int64	Optional	outerIter Max	Specifies the number of outer iterations used in coordinate descent for initialization data events.

Table 4- 22 Input Mapping

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
inputs	varlist	double   string	Required	No default value	Specifies the list of variable names used in classification. Variable names are defined in the input schema, and they are separated by a comma in the list (for example, x,y) .
target	variable	double   string	Optional	""	Specifies the target response variable. For logistic regression, it must be a class variable.
sparse	variable	string	Optional	""	Specifies the sparse variable that is stored in LibSVM format.

The Score window `w_scoring` scores the data.

```
<window-score name='w_scoring'>
<schema>
<fields>
  <field name='id' type='int64' key='true' />
```

```

<field name='y' type='double' />
<field name='yPredictOut' type='double' />
<field name='modelIdOut' type='int64' />
</fields>
</schema>
<models>
<online algorithm='LogisticRegression'>
  <input-map>
<properties>
<property name="inputs">y,x1,...,x784</property> 1
</properties>
</input-map>
<output-map>
<properties>
<property name='yPredictOut'>yPredictOut</property>
  <property name='modelIdOut'>modelIdOut</property>
</properties>
</output-map>
</online>
</models>
</window-score>

```

1 An ellipsis indicates that input values range from x1 to x784, inclusive.

The following properties govern the logistic regression algorithm in the Score window:

Table 4- 23 Input Mapping

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
inputs	varlist	double   string	Optional	""	Specifies the list of variable names used in classification. Variable names are defined in the input schema, and they are separated by a comma in the list (for example, x,y).

Table 4- 24 Output Mapping

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
yOut	variable	double	Optional	""	Specifies the original response. If not specified, it is not displayed.
yPredictOut	variable	double	Optional	""	Specifies the predicted response. If not specified, it is not displayed.
modelIdOut	variable	int64	Optional	""	Specifies the column or field name in the output schema that stores the ID of the model from which the score is computed. If not specified, it is not displayed.
totalErrorOut	variable	double	Optional	""	Specifies the error between yPredictOut and target. If not specified, it is not displayed.
cChosenOut	variable	double	Optional	""	Specifies the regularization parameter whose model had the best results. If not specified, it is not displayed.

The edges are defined at the end of the project. Streaming analytics windows require a role for each incoming edge.

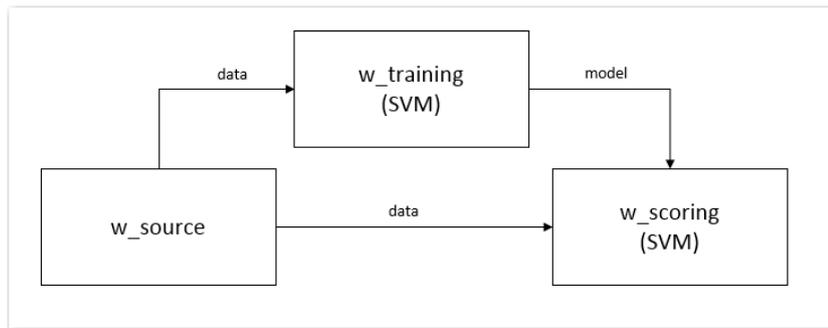
```
<edges>
<edge source='w_data' target='w_train' role='data' />
<edge source='w_data' target='w_score' role='data' />
<edge source='w_train' target='w_score' role='model' />
</edges>
```

## 4.9.6 Training and Scoring with Support Vector Machines

*Support vector machines* are supervised learning models with associated algorithms. Support vector machines apply classification and regression analysis on incoming data. You supply training examples and mark them as belonging to a category. A support vector machine builds a model that assigns new examples to that category.

A support vector machine model represents examples as points in space. Points are mapped onto this space so that examples of each category are separated by a gap. New examples are then mapped into that same space and predicted to belong to a category based on which side of the gap they fall.

Consider the following example:



This continuous query includes the following:

- a Source window that receives data events that stream the data to score
- a Train window that generates and periodically updates the vector machine model
- a Score window that performs the scoring

The Source window `w_source` receives a data event. The input stream is placed into three fields for each observation: an ID that acts as the data stream's key, named `id`; a `y` coordinate of data named `y`; and 784 `x` coordinates.

```
<window-source name='w_source'>
<schema>
<fields>
<field name='id' type='int64' key='true' />
  <field name='y' type='double' />
<field name='x1' type='double' />
  ... 1
<field name='x784' type='double' />
</fields>
</schema>
<connectors>
...
</connectors>
</window-source>
```

**1** An ellipsis indicates that field name values range from `x1` to `x784`, inclusive.

The Train window `w_training` looks at all observations and periodically generates a new model using the support vector machines algorithm. Model events are published to the Score window `w_scoring`.

```

<window-train name='w_training' algorithm='SVM'>
<parameters>
<properties>
  <property name="nInit">60000</property>
  <property name="commitInterval">10000</property>
<property name="dampingFactor">1</property>
<property name="c">1</property>
<property name="centerFlag">0</property>
<property name="scaleFlag">0</property>
  <property name="maxSparseIndex">100000</property>
<property name="numC">5</property>
  <property name="ratioC">4</property>
  <property name="choose">-1</property>
<property name="randSeed">123</property>
  <property name="positiveClass">8</property>
<property name="augmentedValue">1</property>
  <property name="outerIterMax">10</property>
</properties>
</parameters>
<input-map>
<properties>
<property name="inputs">y,x1,...,x784</property>1
  <property name="target">y</property>
<property name="sparse">x2</property>
  </properties>
</input-map>
</window-train>

```

1 An ellipsis indicates that input values range from x1 to x784, inclusive.

The following properties govern the vector machine algorithm in the Train window:

Table 4- 25 Parameters

Name	Variable Type	Required or Optional?	Default Value	Description
nInit	int64	Optional	50	Specifies the number of data events used during initialization.
commitInterval	int64	Optional	10	Specifies the number of data events to process before triggering a commitment of the model to downstream scoring. The specified value must be a positive integer.

Name	Variable Type	Required or Optional?	Default Value	Description
batchSize	int64	Optional	0	Specifies the batch size in processing the training samples. The specified value must be a positive integer. This property affects how much memory is used to buffer data events. If you have sufficient memory, set this to the maximum of <code>nInit</code> and <code>commitInterval</code> .
dampingFactor	double	Optional	1	Specifies the damping factor $\alpha$ ( $0 \leq \alpha \leq 1$ ) for old data points. That is, if the current number of data events to process before triggering a commitment of the model is $T$ , data points arriving at $T$ would have weight 1. Data points at $T - t$ would have weight $\alpha^t$ .
centerFlag	int64	Optional	1	Specifies whether to center the data (dense part) based on the first <code>batchSize</code> data events of the initialization. Specifically, the mean is computed with the first <code>bufferSize</code> data events of the initialization, and each data event is subtracted with the computed mean.
scaleFlag	double	Optional	1	Specifies whether to scale the data (dense part) based on the first <code>batchSize</code> data events of the initialization.
maxSparseIndex	int64	Optional	1	Specifies the number of variables contained in the sparse variable, provided that it exists. Specify a nonnegative integer.
c	double	Optional	10	Specifies the regularization parameter for vector machines. The specified value must be positive.

Name	Variable Type	Required or Optional?	Default Value	Description
numC	int64	Optional	1	Specifies the number of regularization parameters to try.
ratioC	double	Optional	10	Specifies the ratio in setting the set of regularization parameters. The specified value must be greater than 1.
choose	double	Optional	-2	Specifies the criterion in selecting the best regularization parameter. If <code>choose=-2</code> , then the <code>c</code> that achieves the smallest misclassification error is used. If <code>choose=-1</code> , then the <code>c</code> that achieves the smallest hinge loss is used. If <code>choose</code> is nonnegative, then the <code>c</code> that achieves the largest <code>choose score</code> is used.
randSeed	int64	Optional	123	Specifies the random seed in reshuffling data events. Specify a positive value. If <code>randSeed=0</code> , the data in the buffer is not reshuffled. If <code>randSeed&gt;0</code> , the data in the buffer is implicitly reshuffled with the corresponding random seed.
positiveClass	double	Optional	1	Specifies the value of the response that is treated as the positive class.
augmentedValue	double	Optional	1	Specifies the augmented value for handling the intercept. The specified value must be positive.

Name	Variable Type	Required or Optional?	Default Value	Description
outerIterMax	int64	Optional	1	Specifies the number of outer iterations used in coordinate descent for non-initialization data events. The specified value must be positive.
outerIterMaxInit	int64	Optional	outerIterMax	Specifies the number of outer iterations used in coordinate descent for initialization data events. The specified value must be positive.

Table 4- 26 Input Mapping

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
inputs	varlist	double   string	Required	No default value	Specifies the list of variable names used in classification. Variable names are defined in the input schema, and they are separated by a comma in the list (for example, x,y).
target	variable	double   string	Optional	""	Specifies the list of variable names used in classification. Variable names are defined in the input schema, and they are separated by a comma in the list (for example, x,y).
sparse	variable	string	Optional	""	Specifies the name of the sparse variable.

The Score window `w_scoring` scores the data.

```

<window-score name='w_scoring'>
<schema>
<fields>
  <field name='id' type='int64' key='true' />
  <field name='y' type='double' />
  <field name='yPredictOut' type='double' />
  <field name='modelIdOut' type='int64' />
</fields>

```

```

</schema>
<models>
<online algorithm='SVM'>
<input-map>
<properties>
<property name="inputs">y,x1,...,x784</property> 1
</properties>
</input-map>
<output-map>
<properties>
<property name='yPredictOut'>yPredictOut</property>
<property name='modelIdOut'>modelIdOut</property>
<property name='totalErrorOut'>totalErrorOut</property>
<property name='cChosenOut'>cChosenOut</property>
</properties>
</output-map>
</online>
</models>
</window-score>

```

1 An ellipsis indicates that input values range from x1 to x784, inclusive

The following properties govern the vector machine algorithm in the Score window:

Table 4- 27 Input Mapping

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
inputs	varlist	double   string	Optional	""	Specifies the list of variable names used in clustering. Variable names are defined in the input schema, and they are separated by a comma in the list (for example, x,y).

Table 4- 28 Output Mapping

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
yOut	variable	double	Optional	""	Specifies the original response. If not specified, it is not displayed.
yPredictOut	variable	double	Optional	""	Specifies the predicted response. If not specified, it is not displayed.

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
modelIDOut	variable		Optional	""	Specifies the column or field name in the output schema that stores the ID of the model from which the score is computed. If not specified, it is not displayed.
totalErrorOut	variable	double	Optional	""	Specifies the error between <code>yPredictOut</code> and target. If not specified, it is not displayed.
cChosenOut	variable	double	Optional	""	Specifies the value of the regularization parameter whose model had the best results. If not specified, it is not displayed.

The edges are defined at the end of the project. Streaming analytics windows require a role for each incoming edge.

```
<edges>
<edge source='w_data' target='w_train' role='data' />
<edge source='w_data' target='w_score' role='data' />
<edge source='w_train' target='w_score' role='model' />
</edges>
```

You can view the default values of the support vector machines algorithm parameter properties for the Score and Train windows with the command-line utility.

### 4.9.7 Training and Scoring with t-Distributed Stochastic Neighbor Embedding

t-Distributed Stochastic Neighbor Embedding (t-SNE) is a machine learning algorithm for dimensionality reduction that is used to visualize high-dimensional data sets. It is nonparametric and non-linear. t-SNE renders high-dimensional objects into two- or three-dimensional points, making them more suitable for human observation. Similar objects are modeled by nearby points and dissimilar objects are modeled by distant points. For more information, refer to Laurens Van Der Maaten’s github site.

Consider the following model:

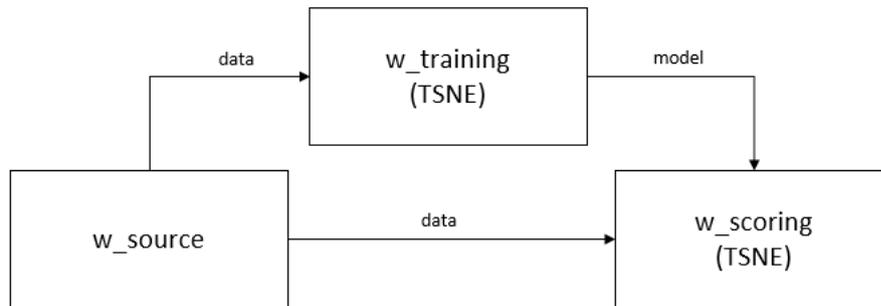


Figure 4-1 TSNE model

This continuous query includes the following:

- a Source window that receives data to be scored and trained
- a Train window that generates and periodically updates the t-SNE model
- a Score window that performs the scoring

The Source window `w_source` receives a data event. Each observation that is streamed into the window consists of several fields that pertain to the characteristics of a flower.

```

<window-source name="w_source" pubsub="true" insert-only="true">
  <schema>
    <fields>
      <field type="int64" name="id" key="true" />
      <field type="string" name="sepal_length" />
      <field type="double" name="sepal_width" />
      <field type="double" name="petal_length" />
      <field type="double" name="petal_width" />
      <field type="double" name="species" />
    </fields>
  </schema>
  <connectors>
    ...
  </connectors>
</window-source>
  
```

The Train window `w_training` parses all observations and periodically generates a new model that uses the t-SNE algorithm.

```

<window-train name="w_training" algorithm="TSNE">
  <parameters>
    <properties>
      <property name="nDims">2</property>
      <property name="initSeed">54321</property>
      <property name="nLandmarks">150</property>
      <property name="perplexity">5</property>
      <property name="learnRate">200</property>
      <property name="maxIters">600</property>
      <property name="nInit">150</property>
      <property name="commitInterval">1000</property>
    </properties>
  </parameters>
  
```

```

<input-map>
<properties>
<property name="inputs">sepal_length,sepal_width,petal_length,petal_width</property>
</properties>
</input-map>
</window-train>
    
```

The following properties govern the TSNE algorithm in the Train window:

Table 4- 29 Parameters

Name	Type	Required or Optional?	Default Value	Description
nDims	int32	Optional	2	Specifies the number of embedding dimensions to report.
initSeed	int32	Optional	12345	Specifies the random seed to use during initialization.
nLandmarks	int32	Optional	100	Specifies the number of landmarks to keep in the model. Landmarks usually consist of past observations. The model retains both the original landmark observations and their embeddings.
perplexity	double	Optional	30	Specifies the perplexity parameter. This parameter determines how tightly to couple the embeddings for faraway observations. Lower perplexity values tend to produce a higher number of small, scattered group.
learnRate	double	Optional	100	Specifies the learning rate for the gradient descent optimization procedure.
maxIters	int32	Optional	500	Specifies the maximum number of iterations for the gradient descent optimization procedure.

Name	Type	Required or Optional?	Default Value	Description
nInit	int64	Optional	50	Specifies the number of data events used during initialization.
commitInterval	int64	Optional	25	Specifies the number of timestamps to be elapsed before triggering a commit of model to downstream scoring.

Table 4- 30 Input Mapping

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
inputs	varlist	double	Optional	"" (empty string)	Specifies the list of variable names used for the high-dimensional analysis. Variable names are defined in the input schema, and they are separated by a comma (for example, x,y ).

The Train window publishes models to the Score window `w_scoring`, which uses them to score incoming data.

```
<window-score name="w_scoring" pubsub="true">
<schema>
<fields>
<field name='species' type='string' />
<field name='id' type='int64' key='true' />
<field name='_DIM_1_' type='double' />
<field name='_DIM_2_' type='double' />
</fields>
</schema>
<models>
<online algorithm="TSNE">
<input-map>
<properties>
<property name="inputs">sepal_length,sepal_width,petal_length,petal_width</property>
</properties>
</input-map>
<output-map>
<properties>
<property name="embeddingOut1">_DIM_1_</property>
<property name="embeddingOut2">_DIM_2_</property>
```

```
</properties>  
</output-map>  
</online>  
</models>  
<connectors>  
  ...  
</connectors>  
</window-score>
```

The following properties are unique to Score windows for TSNE models:

Table 4- 31 Input Mapping

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
inputs	varlist	double	Optional	"" (empty string)	Specifies the list of variable names that are used for the high-dimensional analysis. These variables should be identical to those you specify for the Train window.

Table 4- 32 Output Mapping

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
embeddingOut1	variable	double	Optional	"" (empty string)	Specifies the output variable name in the output schema that stores the first embedding dimension. If not specified, the embedding dimensions column is not shown.
embeddingOut2	variable	double	Optional	"" (empty string)	Specifies the output variable name in the output schema that stores the second embedding dimension. If not specified, the embedding dimensions column is not shown.
embeddingOut3	variable	double	Optional	"" (empty string)	specifies the output variable name in the output schema that stores the third embedding dimension. If not specified, the embedding dimensions column is not shown. Note: There can be no more than three embedding dimensions.

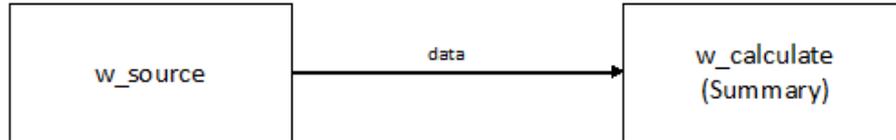
Edges are defined at the end of the project.

```
<edges>
<edge role="data" source="w_source" target="w_training" />
<edge role="data" source="w_source" target="w_scoring" />
<edge role="model" source="w_training" target="w_scoring" />
</edges>
```

### 4.9.8 Calculating Streaming Summary Statistics for One Variable

Edge Streaming Analytics Analytics includes univariate summary statistics as an algorithm for the Calculate window.

Consider the following example:



This continuous query includes the following:

- a Source window that receives the data to be analyzed
- a Calculate window that calculates summary statistics on incoming data events and publishes the results

The Source window `w_source` receives input data. The input stream is placed into three fields for each observation: an ID that acts as the data stream's key, named `id`; an `x` coordinate of data named `x_c`; and a `y` coordinate of data named `y_c`

```

<window-source name='w_source' insert-only='true' autogen-key='false'>
<schema>
  <fields>
    <field name='id' type='int64' key='true' />
    <field name='x_c' type='double' />
    <field name='y_c' type='double' />
  </fields>
</schema>
<connectors>
  ...
</connectors>
</window-source>
  
```

The Calculate window `w_calculate` receives data events and publishes calculated summary statistics according to the summary algorithm properties that are specified.

```

<window-calculate name='w_calculate' algorithm='Summary'>
<schema>
  <fields>
    <field name='id' type='int64' key='true' />
    <field name='y_c' type='double' />
    <field name='x_c' type='double' />
    <field name='nOut' type='double' />
    <field name='nmissOut' type='double' />
    <field name='minOut' type='double' />
    <field name='maxOut' type='double' />
    <field name='sumOut' type='double' />
    <field name='meanOut' type='double' />
    <field name='stdOut' type='double' />
    <field name='varOut' type='double' />
    <field name='cssOut' type='double' />
  </fields>
</schema>
</window-calculate>
  
```

```
<field name='ussOut' type='double' />
<field name='stderrOut' type='double' />
<field name='cvOut' type='double' />
</fields>
</schema>
<parameters>
  <properties>
    <property name='windowLength'>5</property>
  </properties>
</parameters>
<input-map>
  <properties>
    <property name='input'>x_c</property>
  </properties>
</input-map>
<output-map>
  <properties>
    <property name='nOut'>nOut</property>
    <property name='nmissOut'>nmissOut</property>
    <property name='minOut'>minOut</property>
    <property name='maxOut'>maxOut</property>
    <property name='sumOut'>sumOut</property>
    <property name='meanOut'>meanOut</property>
    <property name='stdOut'>stdOut</property>
    <property name='varOut'>varOut</property>
    <property name='cssOut'>cssOut</property>
    <property name='ussOut'>ussOut</property>
    <property name='stderrOut'>stderrOut</property>
    <property name='cvOut'>cvOut</property>
  </properties>
</output-map>
<connectors>
  ...
</connectors>
</window-calculate>
```

The following properties govern the summary algorithm in the Calculate window:

Table 4- 33 Parameters

Name	Variable Type	Required or Optional?	Default Value	Description
windowLength	int64	Optional	3	Specifies the length of the sliding window. The default value is 3. The overlap between consecutive windows is $windowLength - 1$ .
alpha	double	Optional	0.05	Specifies the coverage probability $(1 - \alpha)$ for two-sided confidence intervals.

Table 4- 34 Input Mapping

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
input	variable	double	Required	No default value	Specifies the input variable by its name in the source schema. The univariate summary statistics are calculated for this variable.

Table 4- 35 Output Mapping

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
nOut	variable	int64	Optional	"" (empty string)	Specifies the output variable name for the number of observations analyzed for the incoming data events (N).
nmissOut	variable	int64	Optional	"" (empty string)	Specifies the output variable name for the number of missing values in the incoming data events (NMISS).

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
minOut	variable	double	Optional	"" (empty string)	Specifies the output variable name for the minimum observed value (MIN).
maxOut	variable	double	Optional	"" (empty string)	Specifies the output variable name for the maximum value (MAX).
sumOut	variable	double	Optional	"" (empty string)	Specifies the output variable name for the linear sum (SUM).
meanOut	variable	double	Optional	"" (empty string)	Specifies the output variable name for the mean (MEAN).
stdOut	variable	double	Optional	"" (empty string)	Specifies the output variable name for the standard deviation (STD).
varOut	variable	double	Optional	"" (empty string)	Specifies the output variable name for the sample variance (VAR).
cssOut	variable	double	Optional	"" (empty string)	Specifies the output variable name for the corrected sum of squares (CSS).
ussOut	variable	double	Optional	"" (empty string)	Specifies the output variable name for the uncorrected sum of squares (USS).
stderrOut	variable	double	Optional	"" (empty string)	Specifies the output variable name for the standard error (STDERR).
cvOut	variable	double	Optional	"" (empty string)	Specifies the output variable name for the coefficient of variation (CV).

The calculated summary statistics are organized into event fields that are specified in the schema of the Calculate window. The edge is defined at the end of the project. Streaming analytics windows require a role for each edge.

```
<edges>
<edge source='w_source' target='w_calculate' role='data' />
</edges>
```

You can view the default values of the summary statistics algorithm parameter properties for the Calculate window with the command-line utility.

### 4.9.9 Calculating Streaming Pearson's Correlation

The most common measure of how sets of data correlate with one another is the Pearson correlation coefficient. It shows the linear relationship between two sets of data. Consider the following example:

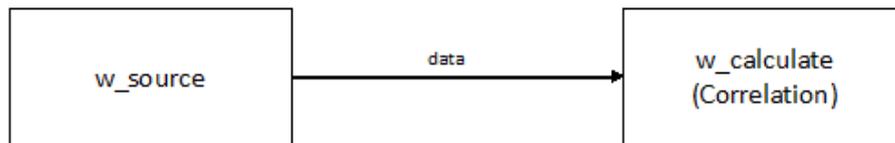


Figure 4-2

This continuous query includes the following:

- a Source window that receives the data to be analyzed
- a Calculate window that calculates the correlation between two variables from an incoming data stream and publishes the results in real time

The Source window `w_source` receives a data event. The input stream is placed into three fields for each observation: an ID that acts as the data stream's key, named `id`; an x coordinate of data named `x_c`; and a y coordinate of data named `y_c`.

```

<window-source name='w_source' insert-only='true' autogen-key='false'>
<schema>
<fields>
  <field name='id' type='int64' key='true' />
  <field name='x_c' type='double' />
  <field name='y_c' type='double' />
</fields>
</schema>
<connectors>
...
</connectors>
</window-source>
  
```

The Calculate window `w_calculate` receives data events including the values of two variables. It publishes their calculated correlation according to the correlation algorithm properties that are specified.

```

<window-calculate name="w_calculate" algorithm="Correlation">
<schema>
<fields>
  <field name='id' type='int64' key='true' />
  <field name='y_c' type='double' />
  <field name='x_c' type='double' />
  <field name='corOut' type='double' />
</fields>
</window-calculate>
  
```

```

</fields>
</schema>
<parameters>
<properties>
<property name="windowLength">5</property>
</properties>
</parameters>
<input-map>
<properties>
<property name="x">x_c</property>
<property name="y">x_c</property>
</properties>
</input-map>
<output-map>
<properties>
<property name="corOut">corOut</property>
</properties>
</output-map>
<connectors>
...
</connectors>
</window-calculate>

```

The following properties govern the correlation algorithm in the Calculate window:

Table 4- 36 Parameters

Name	Variable Type	Required or Optional?	Default Value	Description
windowLength	int64	Optional	3	Specifies the length of the sliding window. The default value is 3. The overlap between consecutive windows is windowLength — 1.

Table 4- 37 Input Mapping

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
x	variable	double	Required	No default value	Specifies the input variable X by its name in the source schema.
y	variable	double	Required	No default value	Specifies the input variable Y by its name in the source schema.

Table 4- 38 Input Mapping

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
corOut	variable	double	Required	"" (empty string)	Specifies the name of the output variable for the correlation between X and Y input variables.

The edge is defined at the end of the project.

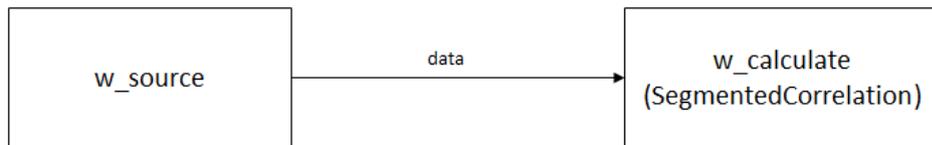
```
<edges>
<edge source='w_source' target='w_calculate' role='data' />
</edges>
```

You can view the default values of the streaming correlation algorithm parameter properties for the Calculate window with the command-line utility.

### 4.9.10 Calculating Segmented Correlation

Segmented correlation is similar to autocorrelation. It specifies the correlation between the elements of a series and others from the same series that are separated from them by a specified interval. You can use segmented correlation to find repeating patterns, such as the occurrence of a signal obscured by noise. The Calculate window can calculate the segmented correlation of variable values streaming over time.

Consider the following example:



This continuous query includes the following:

- a Source window that receives the data to be analyzed
- a Calculate window that calculates the segmented correlation of a variable from an incoming data stream and publishes the results

The Source window `w_source` receives a data event. The input stream is placed into four fields for each observation: an ID that acts as the data stream’s key, named `id`; an x coordinate of data named `x_c`; a y coordinate of data named `y_c`; and an indicator variable named `indicator` that signals when a segment of the series begins and ends.

```
<window-source name='w_source' insert-only='true' autogen-key='false'>
  <schema>
  <fields>
  <field name='id' type='int64' key='true' />
  <field name='x_c' type='double' />
```

```

    <field name='y_c' type='double' />
  <field name='indicator' type='int64' />
</fields>
</schema>
<connectors>
  ...
</connectors>
</window-source>

```

The Calculate window `w_calculate` receives data events from `w_source`. It publishes the calculated autocorrelation of the specified `x` variable according to the segmented correlation algorithm properties that are specified at the window-calculate level.

```

<window-calculate name="w_calculate" algorithm="SegmentedCorrelation">
<schema>
<fields>
  <field name='id' type='int64' key='true' />
  <field name='y_c' type='double' />
  <field name='x_c' type='double' />
  <field name='corOut' type='double' />
</fields>
</schema>
<parameters>
  <properties>
    <property name="sampleSize">4</property>
    <property name="minSize">0</property>
    <property name="maxSize">100</property>
  </properties>
</parameters>
  <input-map>
    <properties>
      <property name="x">x_c</property>
      <property name="indicator">indicator</property>
    </properties>
  </input-map>
  <output-map>
    <properties>
      <property name="corOut">corOut</property>
    </properties>
  </output-map>
<connectors>
  ...
</connectors>
</window-calculate>

```

The segmented correlation algorithm is governed by the following properties:

Table 4- 39 Parameters

Name	Variable Type	Required or Optional?	Default Value	Description
sampleSize	int64	Optional	1,000	Specifies the number of samples to run the correlation. If indicator is not specified, then the input data stream is divided into segments of size sampleSize, and the correlation is performed on adjacent segments. If indicator is specified, then the input data stream is first divided into segments on events when indicator=1. If sampleSize is smaller than 1, then correlation is performed on all adjacent segments. Otherwise, segments are further divided into subsegments of size sampleSize (the size of the last subsegment can be smaller than sampleSize). Then correlation is computed on the first set of subsegments in adjacent segments, the second set of subsegments, and so on.
minSize	int64	Optional	0	Specifies the lower bound of the number of samples to run the correlation.
maxSize	int64	Optional	INT64_MAX	Specifies the upper bound of the number of samples to run the correlation. <b>Note:</b> The default value is INT64_MAX=9223372036854775807.

Table 4- 40 Input Mapping

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
x	variable	double	Required	No default value	Specifies the input variable for the segmented correlation.
indicator	variable	int32	Optional	"" (empty string)	Specifies the input variable for the segment indicator. The variable value should be 1 when an old segment ends and a new segment begins and 0 otherwise.

Table 4- 41 Output Mapping

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
corOut	variable	double	Optional	"" (empty string)	Specifies the output variable name for the segmented correlation.

The edges are defined at the end of the project.

```
<edges>
<edge source='w_source' target='w_calculate' role='data' />
</edges>
```

You can view the default values of the segmented correlation algorithm parameter properties for the Calculate window with the command-line utility.

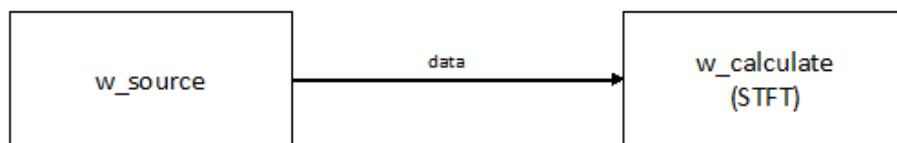
### 4.9.11 Calculating Short-Time Fourier Transforms

A Fourier transform decomposes a function of time into its underlying frequencies. The amplitude, offset, and rotation speed of every underlying cycle is returned by the function. Short-time Fourier transform (STFT) computations consist of multiple “local” discrete Fourier transform computations. The input time series is divided into multiple contiguous bins, and their discrete Fourier transforms are computed in succession. The use of window functions makes the spectra smooth.

STFT is commonly used to monitor the time-varying frequency content of a signal. It can be used in digital filters to detect anomalies in a continuous stream of data. For example, vibrations indicate machine operating conditions. Abnormal conditions can lead to changes in the vibration signal. STFT can be used to monitor the signal frequency band of interest. This monitoring can enable early detection of machine faults and thus more efficient machine maintenance.

For more information about how STFT is implemented in SAS software, see the SAS Forecast Server: Time Series Packages.

Consider the following example:



This continuous query includes the following:

- a Source window that receives the data to be analyzed
- a Calculate window that calculates short-time Fourier transforms (STFTs) on incoming data events and publishes the results

In the code that follows, a Source window `w_source` receives input data. The input stream is placed into two fields for each observation: an ID that acts as the data stream’s key, named `ID`, and a `y` coordinate of data named `y`.

```
<window-source name='w_source' insert-only='true' autogen-
key='true'>
<schema>
<fields>
<field name='ID' type='int64' key='true' />
<field name='y' type='double' />
</fields>
</schema>
<connectors>
...
</connectors> </window-source>
```

The Calculate window `w_calculate` receives data events and publishes calculated transforms according to the STFT algorithm properties that are specified.

The following properties govern the STFT algorithm in the Calculate window:

Table 4- 42 Parameters

Name	Variable Type	Required or Optional?	Default Value	Description
<code>windowLength</code>	<code>int64</code>	Optional	128	Specifies the length of the sliding window. The value that you specify must be greater than the value you specify for <code>overlap</code> .
<code>windowType</code>	<code>int64</code>	Optional	15	Specify one of the following window types: 1=Bartlett, 2=Bohman, 3=Chebyshev, 4=Gaussian, 5=Kaiser, 6=Parzen, 7=Rectangular, 10=Tukey, 11=Bartlett- Hann, 12=Blackman-Harris, 13=Blackman, 14=Hamming, 15=Hanning, and 16=Flat Top.  For more information about these window types, see SAS Visual Forecasting: Time Series Packages.

Name	Variable Type	Required or Optional?	Default Value	Description
windowParam	double	Optional	-1.0	Specifies the parameters for <code>windowType</code> . If not required for the window type selected, this value is ignored.
fftLength	int64	Optional	128	Specifies the length to which windowed data should be expanded. Zeros are appended to the data before the Fast Fourier Transform (FFT) is performed. The specified value must be positive and at least as large as <code>windowLength</code> . A power of two is suggested to maximize computational efficiency.
overlap	int64	Optional	127	Specifies the overlap between consecutive windows. Must be strictly less than <code>windowLength</code> .
binsInSchema	int64	Optional	64	Specifies the number of frequency bins to output. Must be less than or equal to <code>fftLength</code> . For real signals, bins greater than $(fftLength/2)$ are not physically meaningful.

Table 4- 43 Input Mapping

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
input	variable	double	Required	No default value	Specifies the input variable by its name in the source schema. The Calculate window analyzes this variable.
timeId	variable	int64	Required	No default value	Specifies the time ID variable name in the input stream. This variable should be of type int64.

Table 4- 44 Output Mapping

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
keyOut	variable	int64	Optional	"" (empty string)	Specifies a key variable name (unique for each output event) in the output stream.
timeIdOut	variable	int64	Required	No default value	Specifies the time ID variable name in the output stream. There is more than one output event for a given time ID.
binOut	variable	int64	Optional	"" (empty string)	Specifies the frequency bin variable name in the output stream.
powerOut	variable	double	Optional	"" (empty string)	Specifies the name of the power variable in the output stream.

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
phaseOut	variable	double	Optional	"" (empty string)	Specifies the name of the phase variable in the output stream.
powerListOut	varlist	double	Optional	"" (empty string)	Specifies a list of power variables in the output stream.
phaseListOut	varlist	double	Optional	"" (empty string)	Specifies a list of phase variables in the output stream.

You can use one of two formats for output mapping in the Calculate window:

- a list format where one event is generated for each time ID and the event contains an array of the output power variables for the corresponding bins
- a non-list format where a separate output event is generated for each time ID and bin pair

For output mapping that uses a list format, you specify `powerListOut` or `phaseListOut` or both. You use a key of `timeIdOut`.

For output mapping that uses a non-list format, you specify `powerOut` or `phaseOut` or both. You can use a key of `keyOut` or a composite key of `timeIdOut` and `binOut`.

The following code uses non-list format:

```
<window-calculate name="w_calculate" algorithm="STFT">
<schema>
<fields>
<field name='time' type='int64' key='true' />
<field name='bin' type='int64' key='true' />
<field name='power' type='double' />
<field name='phase' type='double' />
</fields>
</schema>
<parameters>
<properties>
<property name="windowLength">64</property>
<property name="windowType">12</property>
<property name="fftLength">256</property>
<property name="binsInSchema">256</property>
<property name="overlap">32</property>
</properties>
</parameters>
<input-map>
<properties>
<property name="input">y</property>
<property name="timeId">ID</property>
</properties>
</input-map>
<output-map>
```

```

<properties>
<property name="timeIdOut">time</property>
<property name="binOut">bin</property>
<property name="powerOut">power</property>
<property name="phaseOut">phase</property>
</properties>
</output-map>
<connectors>
...
</connectors>
</window-calculate>

```

The following code uses the list format:

```

<window-calculate name="w_calculate" algorithm="STFT">
<schema>
<fields>
<field name='time' type='int64' key='true' />
<field name='powerlist' type='array/dbl' />
<field name='phaselist' type='array/dbl' />
</fields>
</schema>
<parameters>
<properties>
<property name="windowLength">64</property>
<property name="windowType">12</property>
<property name="fftLength">256</property>
<property name="binsInSchema">256</property>
<property name="overlap">32</property>
</properties>
</parameters>
<input-map>
<properties>
<property name="input">y</property>
<property name="timeId">ID</property>
</properties>
</input-map>
<output-map>
<properties>
<property name="timeIdOut">time</property>
<property name="powerListOut">powerlist[1-256]</property>
<property name="phaseListOut">phaselist[1-256]</property>
</properties>
</output-map>
<connectors>
...
</connectors>

```

An error is generated when you specify the list and non-list formats in the same Calculate window.

---

### Note

When keyOut is the key in the output schema, there is a limitation of 9999 bins for each timeId.

---

In all cases, the calculated STFT output data is organized by event fields that are specified in the schema of the Calculate window.

The edge is defined at the end of the project. Streaming analytics windows require a role for each edge.

```
<edges>
<edge source='w_source' target='w_calculate' role='data' />
</edges>
```

You can change the values of the properties that govern the STFT algorithm in the Calculate window while data is streaming through the model. First, create an edge between the Source window and the Calculate window with the role "request." Then, stream a reconfig request and events that change the property values. For example, to change the values of all of the properties for STFT:

```
i,n,1,"action","reconfig"
i,n,2,"windowLength","64"
i,n,3,"overlap","32"
i,n,4,"windowType","15"
i,n,5,"fftLength","64"
i,n,6,"binsInSchema","32"
i,n,7,,
```

These events immediately change property values. You can change one or more property values at a time, as required.

You can view the parameters and the input and output mapping properties required to set up a streaming STFT project with the command-line utility.

## 4.9.12 Streaming Distribution Fitting

The distribution fitting algorithm fits a Weibull, Gamma, or Normal distribution to a variable in the incoming data stream.

The Weibull distribution is described by the following probability density function:

$$f(x; c, \sigma, \theta) = \begin{cases} \frac{c}{\sigma} \left( \frac{x-\theta}{\sigma} \right)^{c-1} \exp\left[ -\left( \frac{x-\theta}{\sigma} \right)^c \right] & \text{for } x \geq \theta \\ 0 & \text{for } x < \theta \end{cases}$$

Here the shape parameter  $c > 0$ , the scale parameter  $\sigma > 0$ , and the threshold parameter is  $\theta$ .

The Gamma distribution is described by the following probability density function:

$$f(x; \alpha, \sigma, \theta) = \begin{cases} \frac{1}{\Gamma(\alpha)\sigma} \left(\frac{x-\theta}{\sigma}\right)^{\alpha-1} \exp\left[-\left(\frac{x-\theta}{\sigma}\right)\right] & \text{for } x > \theta \\ 0 & \text{for } x \leq \theta \end{cases}$$

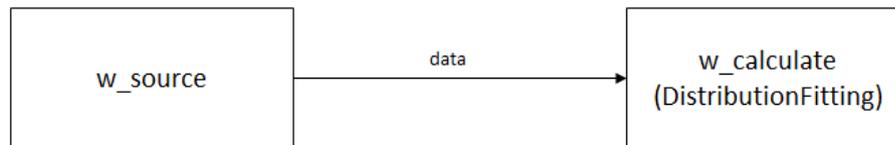
Here the shape parameter  $\alpha > 0$ , the scale parameter  $\sigma > 0$ , and the threshold parameter is  $\theta$ .

The Normal distribution is described by the following probability density function:

$$f(x; \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left[-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right]$$

Here the mean is  $\mu$  and the standard deviation  $\sigma > 0$ .

Consider the following example:



This continuous query includes the following:

- a Source window that receives the data to be analyzed
- a Calculate window that fits the Weibull distribution to a variable from an incoming data stream and publishes the variable's functional parameters as results

The Source window `w_source` receives a data event. The input stream is placed into three fields for each observation: an ID that acts as the data stream's key, named `id`; an x coordinate of data named `x_c`; and a y coordinate of data named `y_c`.

```

<window-source name='w_source' insert-only='true' autogen-key='false'>
<schema>
<fields>
<field name='id' type='int64' key='true' />
<field name='x_c' type='double' />
<field name='y_c' type='double' />
</fields>
</schema>
<connectors>
...
</connectors>
</window-source>
  
```

The Calculate window `w_calculate` receives data events. It publishes the  $\beta$ ,  $\alpha$ , and  $\mu$  parameters of the Weibull probability density function for the specified x variable.

```

<window-calculate name="w_calculate" algorithm="DistributionFitting">
<schema>
<fields>
  
```

```

<field name='id' type='int64' key='true' />
<field name='y_c' type='double' />
<field name='x_c' type='double' />
<field name='betaOut' type='double' />
<field name='alphaOut' type='double' />
<field name='muOut' type='double' />
<field name='convergeOut' type='int64' />
</fields>
</schema>
<parameters>
<properties>
<property name="windowLength">5</property>
<property name="overlap">2</property>
<property name="maxIter">50</property>
<property name="distribution">Weibull</property>
</properties>
</parameters>
<input-map>
<properties>
<property name="x">x_c</property>
</properties>
</input-map>
<output-map>
<properties>
<property name="betaOut">betaOut</property>
<property name="alphaOut">alphaOut</property>
<property name="muOut">muOut</property>
<property name="convergeOut">convergeOut</property>
</properties>
</output-map>
<connectors>
...
</connectors>
</window-calculate>

```

The distribution fitting algorithm is governed by the following properties:

Table 4- 45 Parameters

Name	Value Type	Required or Optional?	Default Value	Description
windowLength	int64	Optional	3	Specifies the length of the sliding window. The value that you specify must be greater than the value you specify for <code>overlap</code> .
overlap	int64	Optional	1	Specifies the overlap between consecutive windows. Must be strictly less than <code>windowLength</code> .
maxIter	int32	Optional	100	Specifies the maximum number of iterations.

Name	Value Type	Required or Optional?	Default Value	Description
distribution	varchar	Optional	Weibull	Specifies the type of probability distribution to fit.
beta	double	Optional	NaN	Specifies the fixed value of $\beta$ .
alpha	double	Optional	NaN	Specifies the fixed value of $\alpha$ .
mu	double	Optional	NaN	Specifies the fixed value of $\mu$ .
c	double	Optional	NaN	Specifies the fixed value of $c$ .
sigma	double	Optional	NaN	Specifies the fixed value of $\sigma$ .
theta	double	Optional	NaN	Specifies the fixed value of $\theta$ .

Table 4- 46 Input Mapping

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
x	variable	double	Required	No default value	Specifies the input variable for distribution fitting.

Table 4- 47 Output Mapping

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
betaOut	variable	double	Optional	"" (empty string)	Specifies the output variable name for parameter $\beta$ .
alphaOut	variable	double	Optional	"" (empty string)	Specifies the output variable name for parameter $\alpha$ .
muOut	variable	double	Optional	"" (empty string)	Specifies the output variable name for parameter $\mu$ .
cOut	variable	double	Optional	"" (empty string)	Specifies the output variable name for parameter $c$ .
sigmaOut	variable	double	Optional	"" (empty string)	Specifies the output variable name for parameter $\sigma$ .

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
thetaOut	variable	double	Optional	"" (empty string)	Specifies the output variable name for parameter $\theta$ .
convergeOut	variable	double	Optional	"" (empty string)	Specifies the output variable name that indicates whether convergence is attained. Set the value to 1 when computation is converged and 0 otherwise.

The edge is defined at the end of the project.

```
<edges>
<edge source='w_source' target='w_calculate' role='data' />
</edges>
```

You can view the default values of the streaming distribution fitting algorithm parameter properties for the Calculate window with the command-line utility.

### 4.9.13 Computing Fit Statistics for Scored Results

The goodness of fit of a statistical model describes how well a model fits a set of data. Goodness-of-fit measures summarize the difference between observed values and predicted values of the model under consideration. The goodness-of-fit algorithm calculates fit statistics such as the following:

- average square error
- mean square logarithmic error
- mean absolute error
- mean consequential error
- multiclass log loss

You can apply these metrics to the output of a model (from a Score window) to compare models. For more information, see the ASSESS Procedure in the SAS Visual Statistics: Procedures.

Consider the following example:



The continuous query includes the following:

- a Source window that receives scored data from a Score window to be analyzed
- a Calculate window that runs the algorithm calculating fit statistics

The Source window `w_source` receives a data event. The input stream is placed into three fields for each observation: an ID that acts as the data stream's key, named `id`; an `x` coordinate of data named `x_c`; and a `y` coordinate of data named `y_c`. `x_c` contains an observed value, and `y_c` contains a value predicted by a regression model.

```
<window-source name='w_source' insert-only='true' index='pi_EMPTY'>
<schema>
<fields>
<field name='id' type='int64' key='true' />
<field name='x_c' type='double' />
<field name='y_c' type='double' />
</fields>
</schema>
<connectors>
...
</connectors>
</window-source>
```

The Calculate window `w_calculate` receives data events. It publishes goodness-of-fit statistics according to the algorithm properties that are specified.

```
<window-calculate name='w_calculate' algorithm='FitStat'>
<schema>
<fields>
<field name='id' type='int64' key='true' />
<field name='nOut' type='double' />
<field name='nmissOut' type='double' />
<field name='aseOut' type='double' />
<field name='divOut' type='double' />
<field name='raseOut' type='double' />
<field name='mceOut' type='double' />
<field name='mcllOut' type='double' />
<field name='maeOut' type='double' />
<field name='rmaeOut' type='double' />
<field name='msleOut' type='double' />
<field name='rmsleOut' type='double' />
</fields>
</schema>
<properties>
<property name="inputs">y_c</property>
<property name="response">x_c</property>
</properties>
</input-map>
<output-map>
<properties>
<property name="nOut">nOut</property>
<property name="nmissOut">nmissOut</property>
<property name="aseOut">aseOut</property>
<property name="divOut">divOut</property>
<property name="raseOut">raseOut</property>
<property name="mceOut">mceOut</property>
<property name="mcllOut">mcllOut</property>
<property name="maeOut">maeOut</property>
```

```

<property name="rmaeOut">rmaeOut</property>
<property name="msleOut">msleOut</property>
<property name="rmsleOut">rmsleOut</property>
</window-calculate>

```

The following properties govern the algorithm in the Calculate window:

Table 4- 48 Parameters

Name	Value Type	Required or Optional?	Default Value	Description
windowLength	int64	Optional	3	Specifies the length of the sliding window. The overlap between consecutive windows is $\text{windowLength} - 1$ .
ClassLabels	string-list	Optional	No default value	Specifies a comma-separated list that contains the corresponding label for each predicted probability. This parameter is required for classification models.
labelLen	int64	Optional	123	Specifies the length of response labels.

Table 4- 49 Input Mapping

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
inputs	variable(s)	double	Required	No default value	Specifies input variables. For regression models, only one input variable is required. That variable specifies the predicted response. For classification models, this variable list specifies the predicted probabilities for each response class.
response	response variable	double   string	Required	No default value	Specifies the response variable (that is, the target variable).

Table 4- 50 Output Mapping

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
nOut	variable	double	Optional	""	Specifies the output variable for the number of observations (N).
nmissOut	variable	double	Optional	""	Specifies the output variable for the number of missing values (NMISS).
aseOut	variable	double	Optional	""	Specifies the average square error (ASE). $ASE = \frac{1}{N} \sum_{i=1}^n y_i (-y_i)^2$ <ul style="list-style-type: none"> <li>• <math>y_i</math> is the actual target value of observation <math>i</math></li> <li>• <math>y_i</math> is the predicted target value of observation <math>i</math></li> </ul>
divOut	variable	double	Optional	""	Specifies the divisor of the average square error.
raseOut	variable	double	Optional	""	Specifies the root average square error. $RASE = \sqrt{ASE}$
mceOut	variable	double	Optional	""	Specifies the mean consequential error. $MCE = \frac{1}{N} \sum_{t_i \neq \hat{t}_i} 1$
mcllOut	variable	double	Optional	""	Specifies the multiclass log loss. $\text{logloss} = - \frac{1}{N} \sum_{i=1}^n \sum_{j=1}^m y_{i,j} \log(p_{i,j})$
maeOut	variable	double	Optional	""	Specifies the mean absolute error.
rmaeOut	variable	double	Optional	""	Specifies the root mean absolute error.
msleOut	variable	double	Optional	""	Specifies the mean square logarithmic error
rmsleOut	variable	double	Optional	""	Specifies the root mean square logarithmic error.

The edge is defined at the end of the project.

```
<edges>
<edge source='w_source' target='w_calculate' role='data' />
</edges>
```

You can view the default values of the goodness of fit algorithm parameter properties for the Calculate window with the command-line utility.

### 4.9.14 Computing Receiver Operating Characteristic (ROC) Information

Receiver operating characteristic (ROC) information shows the diagnostic ability of a classifier system as you vary its discrimination threshold. You create ROC information by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings.

In a ROC information table, the confusion matrix is calculated based on the event in each cutoff point.

- $m$  is the total cutoff points
- $n$  is the number of observations
- $N$  is the sum of observation frequencies in the data
- $w_i$  are the observation frequencies
- $a_k$  is true positive at cutoff point  $k$ ,  $k \in [0, m - 1]$
- $b_k$  is false positive at cutoff point  $k$ ,  $k \in [0, m - 1]$
- $c_k$  is false negative at cutoff point  $k$ ,  $k \in [0, m - 1]$

For more information, see the following:

- The ASSESS Procedure in SAS Visual Statistics: Procedures.
- Fawcett, T. (2006). "An Introduction to ROC Analysis." Pattern Recognition Letters 27:861–874. Special issue on ROC analysis in pattern recognition, edited by F. Tortorella.

Consider the following example.



The continuous query includes the following:

- a Source window that receives the data to be analyzed
- a Calculate window that performs the ROC calculation The Source window `w_source` sends a data event.

```

<window-source name='w_source'>
<schema>
<fields>
<field name='id' type='int64' key='true' />
<field name='y_c' type='string' />
<field name='p_0' type='double' />
<field name='p_1' type='double' />
<field name='p_2' type='double' />
</fields> </schema> <connectors>
...
</connectors>
</window-source>
  
```

The Calculate window `w_calculate` receives data events including the values of several variables. It publishes a confusion table.

```

<window-calculate name='w_calculate' algorithm='ROC'>
<schema>
<fields> <field name='id' type='int64' key='true' />
<field name='binIdOut' type='int64' key='true' />
  
```

```
<field name='cutOffOut' type='double' />
<field name='tpOut' type='double' />
<field name='fpOut' type='double' />
<field name='fnOut' type='double' />
<field name='tnOut' type='double' />
<field name='sensitivityOut' type='double' />
<field name='specificityOut' type='double' />
<field name='ksOut' type='double' />
<field name='ks2Out' type='double' />
<field name='fHalfOut' type='double' />
<field name='fprOut' type='double' />
<field name='accOut' type='double' />
<field name='fdrOut' type='double' />
<field name='flOut' type='double' />
<field name='cOut' type='double' />
<field name='giniOut' type='double' />
<field name='gammaOut' type='double' />
<field name='tauOut' type='double' />
<field name='miscEventOut' type='double' />
</fields> </schema>
<parameters>
<properties>
<property name='cutStep'>0.1</property>
<property name='event'>good</property>
</properties>
</parameters>
<input-map>
<properties>
<property name="input">p_0</property>
<property name="response">y_c</property>
</properties>
</input-map>
<output-map>
<properties>
<property name="binIdOut">binIdOut</property>
<property name="cutOffOut">cutOffOut</property>
<property name="tpOut">tpOut</property>
<property name="fpOut">fpOut</property>
<property name="fnOut">fnOut</property>
<property name="tnOut">tnOut</property>
<property name="sensitivityOut">sensitivityOut</property>
<property name="specificityOut">specificityOut</property>
<property name="ksOut">ksOut</property>
<property name="ks2Out">ks2Out</property>
<property name="fHalfOut">fHalfOut</property>
<property name="fprOut">fprOut</property>
<property name="accOut">accOut</property>
<property name="fdrOut">fdrOut</property>
<property name="flOut">flOut</property>
<property name="cOut">cOut</property>
<property name="giniOut">giniOut</property>
<property name="gammaOut">gammaOut</property>
<property name="tauOut">tauOut</property>
<property name="miscEventOut">miscEventOut</property>
</properties>
```

```

</output-map>
<connectors>
...
</connectors>
</window-calculate>

```

The following properties govern the ROC algorithm in the Calculate window:

Table 4- 51 Parameters

Name	Value Type	Required or Optional?	Default Value	Description
labelLen	int64	Optional	128	Specifies the length of the response event.
cutStep	double	Optional	0.01	Specifies the bin width. Specify a value between 0 and 1. The default, 0.01, generates 100 bins to fit the ROC.
windowLength	int64	Optional	0	Specifies the length of the sliding window. The overlap between consecutive windows is $\text{windowLength} - 1$ .
event	string	Required	No default value	Specifies the desired response event to use for ROC calculations.
input	variable	Required	No default value	Specifies the input variable, which is the predicted probability for the given event.
response	response variable	Required	No default value	Specifies the response variable.

Table 4- 52 Input Mapping

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
input	variable	double	Required	No default value	Specifies the input variable, which is the predicted probability for the given event.
response	response variable	string	Required	No default value	Specifies the response variable.

Table 4- 53 Output Mapping

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
binIDout	variable	int64	Optional	"" (empty string)	Specifies the bin ID.
cutoffOut	variable	double	Optional	"" (empty string)	Specifies the cutoff probability.
tpOut	variable	double	Optional	"" (empty string)	Specifies the number of true positives.
fpOut	variable	double	Optional	"" (empty string)	Specifies the number of false positives.
fnOut	variable	double	Optional	"" (empty string)	Specifies the number of false negatives.
tnOut	variable	double	Optional	"" (empty string)	Specifies the number of true negatives.
sensitivityOut	variable	double	Optional	"" (empty string)	Specifies the ROC sensitivity.
specificityOut	variable	double	Optional	"" (empty string)	Specifies the ROC specificity.
ksOut	variable	double	Optional	"" (empty string)	Specifies the Kolmogorov-Smirnov statistic.
ks2Out	variable	double	Optional	"" (empty string)	Specifies the KS2.
fHalfOut	variable	double	Optional	"" (empty string)	Specifies the F_Half.
fprOut	variable	double	Optional	"" (empty string)	Specifies the false positive rate.
accOut	variable	double	Optional	"" (empty string)	Specifies the accuracy (ACC).
fdrOut	variable	double	Optional	"" (empty string)	Specifies the false discovery rate.
f1Out	variable	double	Optional	"" (empty string)	Specifies the F1 score.
cOut	variable	double	Optional	"" (empty string)	Specifies C (area under the curve). $C = \frac{\mu + \theta}{\rho}$
giniOut	variable	double	Optional	"" (empty string)	Specifies the Gini coefficient. $Gini = \frac{\mu - \omega}{\rho}$
gammaOut	variable	double	Optional	"" (empty string)	Specifies Goodman and Kruskal's Gamma $gamma = \frac{\mu - \omega}{\mu + \omega}$
tauOut	variable	double	Optional	"" (empty string)	Specifies Kendall's Tau-a $tau = \frac{N(\mu - \omega)}{2(N - 1)}$
miscEventOut	variable	double	Optional	"" (empty string)	Specifies the Misclassification rate (1- area under the curve).

For these output variables, the following is true:

$$\theta = \sum_{k=1}^m ((a_{k-1} - a_k)(b_{k-1} - b_k))$$

$$\mu = \sum_{k=2}^m ((a_{k-1} - a_k) \sum_{j=1}^k (b_{j-1} - b_j))$$

$$\omega = \sum_{k=1}^m \left( \left( a_{k-1} - a_k \right) \sum_{j=k+1}^m (b_{j-1} - b_j) \right)$$

$$\rho = a_0 b_0, a_m = 0 \text{ and } b_m = 0$$

The edge is defined at the end of the project.

```
<edges>
<edge source='w_source' target='w_calculate' role='data' />
</edges>
```

You can view the default values of the ROC algorithm parameter properties for the Calculate window with the command-line utility.

### 4.9.15 Streaming Numerical Data to Create a Histogram

A *histogram* graphically represents a distribution of numerical data. This algorithm processes a stream of numerical data and puts it in bins to generate boundaries for creating a histogram that fits it.

Specific features of this algorithm are as follows:

- It keeps track of the center and the height of each bin, so all the points that are encapsulated in a bin are represented by that bin center.
- It can insert every new point in a logarithmic time with respect to the number of bins. If  $nBins$  is the number of bins, the insertion needs  $O(\lg nBins)$  time. Thus, you can use a lot of bins with very little time penalty.
- There is a fading factor value called alpha that fades the height of each bin. This, in effect, forgets old data and gives greater weight to the most recent data. You can set the fading factor directly by using the alpha value or by setting the half-life-steps value.
- The algorithm can calculate a good approximation of any quantile on the histogram. This approximation improves as the number of bins increases.

Consider the following example:



The continuous query includes the following:

- a Source window that receives the data to fit into a histogram
- a Calculate window that performs bucketing

The Source window `w_source` receives input data that consists of an ID and x,y coordinates.

```
<window-source name='w_source' insert-only='true' index='pi_EMPTY'>
<schema>
<fields>
<field name='id' type='int64' key='true' />
<field name='x_c' type='double' />
<field name='y_c' type='double' />
</fields>
</schema>
<connectors>
...

```

```
</connectors>
</window-source>
```

The Calculate window `w_calculate` receives data events and publishes output variable values for bin centers and heights and for quantiles.

```
<window-calculate name='w_calculate' algorithm='Histogram'>
<schema>
<fields>
<field name='id' type='int64' key='true' />
<field name='binCenters' type='array(dbl)' />
<field name='binHeights' type='array(dbl)' />
<field name='quantiles' type='array(dbl)' />
</fields> </schema> <parameters>
<properties>
<property name="nBins">5</property>
<property name="alpha">0.999</property>
<property name="quantileList">0.25,0.50,0.75</property>
<property name="halfLifeSteps">2</property>
<property name="reportInterval">10</property>
</properties>
</parameters>
<input-map>
<properties>
<property name="input">x_c</property>
</properties>
</input-map>
<output-map>
<properties>
<property name="binCentersOut">binCenters[1-20]</property>
<property name="binHeightsOut">binHeights[1-20]</property>
<property name="quantilesOut">quantiles[1-3]</property>
</properties>
</output-map>
<connectors>
...
</connectors>
</window-calculate>
```

The following properties govern the Histogram algorithm in the Calculate window:

Table 4- 54 Parameters

Name	Variable Type	Required or Optional?	Default Value	Description
nBins	int64	Optional	20	Specifies the maximum number of bins in the histogram.
alpha	double	Optional	1.0	Specifies the fading out factor ( $0 < \alpha \leq 1$ ). The recommended value for alpha is greater than 0.997.

Name	Variable Type	Required or Optional?	Default Value	Description
halfLifeSteps	int64	Optional	0	Specifies the number of steps at which the weight of the input reaches half of its original weight.
binRemovalThreshold	double	Optional	0	Specifies the threshold for bin removal during fading mode ( $\alpha < 1$ ). Bins with heights smaller than the threshold are removed.
quantileList	double-list	Optional	"" (empty string)	Specifies a comma-separated list that contains quantiles to compute. Probabilities must be in the range [0,1] and sorted in ascending order.
reportInterval	int64	Optional	5	Specifies the interval of reporting histogram and quantile results (if any).

Table 4- 55 Input Mapping

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
input	variable	double	Required	No default value	Specifies the input variable with which to build the histogram.
binCentersOut	variable list	double	Optional	"" (empty string)	Specifies a list of output variable names for bin centers.
binHeightsOut	variable list	double	Optional	"" (empty string)	Specifies a list of output variable names for bin heights.
quantilesOut	variable list	double	Optional	"" (empty string)	Specifies a list of output variable names for quantiles.

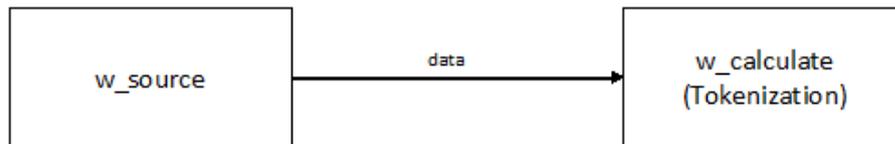
The edge is defined at the end of the project.

```
<edges>
<edge source='w_source' target='w_calculate' role='data' />
</edges>
```

You can view the default values of the streaming histogram algorithm parameter properties for the Calculate window with the command-line utility.

### 4.9.16 Streaming Text Tokenization

The Calculate window supports text tokenization through a tokenization algorithm. Consider the following example:



This continuous query includes the following:

- a Source window that receives the text data to be analyzed
- a Calculate window that tokenizes text in incoming data events and publishes the results

The Source window `w_source` receives input data. The input stream is placed into two fields for each observation: a document ID that acts as the data stream's key, named `docId`, and a string of incoming text, named `doc`.

```

<window-source name='w_source' insert-only='true'>
<schema>
<fields>
<field name='docId' type='int64' key='true' />
<field name='doc' type='string' />
</fields>
</schema>
<connectors>
...
</connectors>
</window-source>
  
```

The Calculate window `w_calculate` receives data events and publishes word tokens created with the tokenization algorithm.

```

<window-calculate name='w_calculate' algorithm='Tokenization'>
<schema>
<fields>
<field name='docId' type='int64' key='true' />
<field name='tokenId' type='int64' key='true' />
<field name='word' type='string' />
<field name='startPos' type='int32' />
<field name='endPos' type='int32' />
</fields>
</schema>
<input-map>
<properties>
<property name='docId'>docId</property>
<property name='doc'>doc</property>
  
```

```

</properties>
</input-map>
<output-map>
<properties>
<property name='docIdOut '>docId</property>
<property name='tokenIdOut '>tokenId</property>
<property name='wordOut '>word</property>
<property name='startPosOut '>startPos</property>
<property name='endPosOut '>endPos</property>
</properties>
</output-map>
<connectors>
...
</connectors>
</window-calculate>

```

The following properties govern the tokenization algorithm in the Calculate window:

Table 4- 56 Input Mapping

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
doc	variable	int64	Required	No default value	Specifies the input variable for the input document from the Source window.
docId	variable	string	Required	No default value	Specifies the input variable for the unique document ID.
docIdOut	variable	int64	Required	No default value	Specifies the output variable for the unique doc ID.
tokenIdOut	variable	int64	Required	No default value	Specifies the output variable for the unique ID of the token.
wordOut	variable	string	Required	No default value	Specifies the output variable for the word content in the token.
startPosOut	variable	int32	Optional	"" (empty string)	Specifies the output variable for the starting position of the token word.
endPosOut	variable	int32	Optional	"" (empty string)	Specifies the output variable for the ending position of the token word.

The calculated tokens are organized by the event fields that are specified in the schema of the Calculate window.

The edges are defined at the end of the project. Streaming analytics windows require a role for each edge.

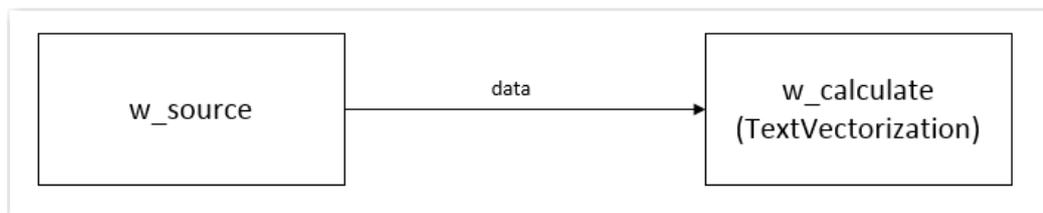
```
<edges>
<edge source='w_source' target='w_calculate' role='data' />
</edges>
```

You can view the default values of the streaming text tokenization algorithm parameter properties for the Calculate window with the command-line utility.

### 4.9.17 Streaming Text Vectorization

The Calculate window supports text vectorization through a proprietary vectorization algorithm. Vectorizing text creates maps from words or n-grams to a vector space. A vector space is an algebraic model to represent text documents as vectors of identifiers (for example, index terms).

Consider the following example:



This continuous query includes the following:

- a Source window that receives the text data to analyze
- a Calculate window that vectorizes text in incoming data events and publishes the results

The Source window `w_source` receives input data that consists of a document ID and the name of a token.

```
<window-source name='w_source'>
<schema>
<fields>
<field name='docId' type='int64' key='true' />
<field name='tokenId' type='int64' key='true' />
</fields>
</schema>
<connectors>
...
</connectors>
</window-source>
```

The Calculate window `w_calculate` receives data events and publishes word vectors created with the vectorization algorithm.

```
<window-calculate name='w_calculate' algorithm='TextVectorization'>
<schema>
<fields>
<field name='docId' type='int64' key='true' />
<field name='tokenId' type='int64' key='true' />
```

```

<field name='word' type='string' />
<field name='v1' type='double' />
<field name='v2' type='double' />
<field name='v3' type='double' />
</fields>
</schema>
<parameters>
<properties>
<property name="wordVec">wordVec1.csv</property>
<property name='outputDocVec'>0</property>
<property name="wordVecDelimiter">COMMA</property>
</properties>
</parameters>
<input-map>
<properties>
<property name='docId'>docId</property>
<property name='token'>word</property>
</properties>
</input-map>
<output-map>
<properties>
<property name='docIdOut'>docId</property>
<property name='vectorOut'>v1,v2,v3</property>
</properties>
</output-map>
</window-calculate>
</windows>

```

The following properties govern the vectorization algorithm in the Calculate window:

Table 4- 57 Parameters

Name	Value Type	Required or Optional?	Default Value	Description
wordVec	string	Required	No default value	Specifies the word vector filename.
wordVecDelimiter	string	Optional	"COMMA"	Specifies the delimiter of the word vector file. Legal values are "COMMA", "TAB", or "SPACE".
wordVecLineBreak	string	Optional	"LF"	Specifies the line break of the word vector file. It can be "LF", "CR", or "CRLF".
startList	string	Optional	""	Specifies the filename of the start list, which contains the words that are considered during vectorization.

Name	Value Type	Required or Optional?	Default Value	Description
stopList	string	Optional	""	Specifies the filename of the stop list, which contains the words that are ignored.
outputDocVec	int64	Optional	0	Specifies whether to return a document vector or not. If it is set to 0, then word vectors are returned. Otherwise, document vectors are returned.

Table 4- 58 Input Mapping

Name	Value Type	Variable Type	Required or Optional ?	Default Value	Description
docID	variable	int64	Optional	""	Specifies the input variable name of a document ID. It is required when <code>outputDocVec</code> is set to nonzero.
token	variable	string	Required	No default value	Specifies the input variable name of a token.

Table 4- 59 Output Mapping

Name	Value Type	Variable Type	Required or Optional ?	Default Value	Description
docIDOut	variable	int64	Optional	""	Specifies the output variable name of a document ID. It is required when <code>outputDocVec</code> is set to nonzero.
vectorOut	variable	string	Required	No default value	Specifies a list of output variable names for word or document vectors.

The edge is defined at the end of the project. Streaming analytics windows require a role for each incoming edge.

```
<edges>
<edge source='w_source' target='w_calculate' role='data' />
</edges>
```

You can view the default values of the text vectorization algorithm parameter properties for the Calculate window with the command-line utility.

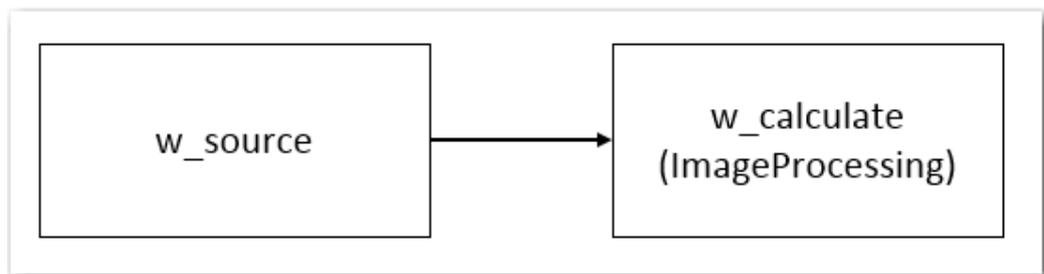
## 4.9.18 Processing Image Data

Edge Streaming Analytics provides an image processing algorithm that you can use on streaming image data. In the Calculate window, you specify one of the following processing functions to apply to the incoming image: resize, crop rotate, or flip.

### Note

For a list of the supported image formats, see the table of image formats supported by the loadImages action in the SAS Visual Data Mining and Machine Learning: Programming Guide.

Consider the following example:



The continuous query includes the following:

- a Source window that receives images
- a Calculate window that runs the image processing algorithm on those images

Here is code for the Source window:

```

<window-source index="pi_EMPTY" insert-only="true" name="w_source">
<schema>
<fields>
<field key="true" name="id" type="int64" />
<field key="false" name="image" type="blob" /> 1
</fields>
</schema>
</window-source>
  
```

The input data image is a binary large object.

The following Calculate window applies the crop function to that image:

```

<window-calculate algorithm="ImageProcessing" name="w_calculate">
<schema>
<fields>
<field key="true" name="id" type="int64" />
<field key="false" name="resized" type="blob" />
</fields>
</schema>
<parameters>
<properties>
<property name="function">crop</property>
<property name="width">200</property>
<property name="outputHeight">250</property>
<property name="outputWidth">250</property>
  
```

```
<property name="y">50</property>
<property name="x">50</property>
<property name="height">200</property>
</properties>
</parameters>
<input-map>
<properties>
<property name="imageInput">image</property>
</properties>
</input-map>
<output-map>
<properties>
<property name="imageOutput">cropped</property>
</properties>
</output-map>
</window-calculate>
```

The following Calculate window applies the resize function to the image:

```
<window-calculate algorithm="ImageProcessing" name="w_calculate">
<schema>
<fields>
<field key="true" name="id" type="int64" />
<field key="false" name="resized" type="blob" />
</fields>
</schema>
<parameters>
<properties>
<property name="function">resize</property>
<property name="width">200</property>
<property name="height">200</property>
</properties>
</parameters>
<input-map>
<properties>
<property name="imageInput">image</property>
</properties>
</input-map>
<output-map>
<properties>
<property name="imageOutput">resized</property>
</properties>
</output-map>
</window-calculate>
```

The following properties govern the image processing algorithm in the Calculate window:

Table 4- 60 Parameters

Name	Value Type	Required or Optional?	Default Value	Description
function	string	Required	"resize"	Specifies the image processing function to be applied: <code>resize</code> , <code>crop</code> , <code>rotate</code> , or <code>flip</code> .
preFlip	int32	Optional	-1000	Specifies whether the input image is flipped before processing. This is used for video streaming.
x	int64	Optional	0	Specifies the x location. Useful for the <code>crop</code> function.
y	int64	Optional	0	Specifies the y location. Useful for the <code>crop</code> function.
width	int64	Optional	100	Specifies the width of an image. Useful for the <code>crop</code> and <code>resize</code> functions.
height	int64	Optional	100	Specifies the height of an image. Useful for the <code>crop</code> function.
outputWidth	int64	Optional	0	Specifies the output width of an image. Useful for the <code>crop</code> function.
outputHeight	int64	Optional	0	Specifies the output height of an image. Useful for the <code>crop</code> and <code>resize</code> functions.
theta	double	Optional	0	Specifies the theta parameter (the rotation angle when you use the <code>rotate</code> function).
type	double	Optional	0	Specifies the flip type. A value of 0 specifies vertical flipping. A value of 1 specifies horizontal flipping. A value of -1 specifies horizontal and vertical flipping.

Table 4- 61 Input Mapping

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
imageInput	variable	blob	Required	No default value	Specifies the input image.

Table 4- 62 Output Mapping

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
imageOutput	variable	blob	Required	No default value	Specifies the output image.

The edge is defined at the end of the project.

```
<edges>
<edge role="data" source="w_source" target="w_calculate" />
</edges>
```

You can view the default values of the image processing algorithm parameter properties for the Calculate window with the command-line utility.

### 4.9.19 Computing the Moving Relative Range

The moving relative range (MRR) provides a measure of volatility for a nonstationary time series, where the mean and the variance of the series change over time. For example, you could use MRR to detect electrical disturbances in the power grid.

Let  $X_t$  denote the  $t^{\text{th}}$  element of the time series. The Range and the MRR for  $X_t$  is computed as follows:

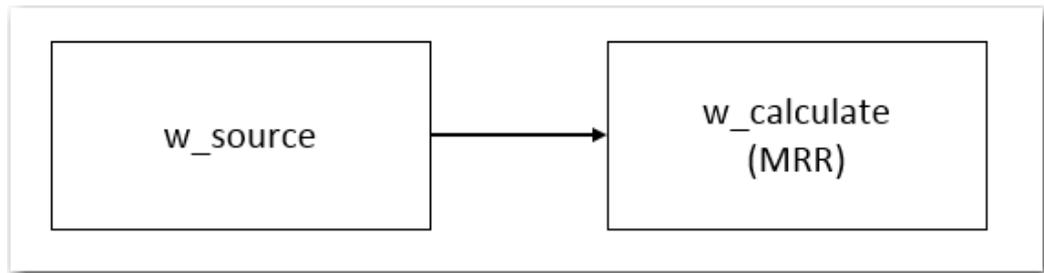
$$Range_t = Range(X_t, X_{t-1}, \dots, X_{t-M+1})$$

$$MRR_t = \frac{Range_t}{Median(Range_t, Range_{t-1}, \dots, Range_{t-K+1})}$$

M is the window length to calculate the range. K is the window length to compute the moving relative range. The process computes the range over the last M data points, and then it uses that computed range over the last K points to compute the MRR.

For more information, see “Time Filters Package for the TSMODEL Procedure” in the SAS Visual Forecasting: Time Series Packages.

Consider the following example:



The continuous query includes the following:

- a Source window that receives the data to analyze
- a Calculate window that performs the moving relative range calculation

The Source window `w_source` receives a data event. The input stream is placed into two fields for each observation: an ID that acts as the data stream's key, named `id`, and a variable `x1` that represents an element of the time series.

```

<window-source name='w_source'>
<schema> <fields>
<field name='id' type='int64' key='true' />
<field name='x1' type='double' />
</fields>
</schema>
<connectors>
...
</connectors>
</window-source>
  
```

The Calculate window `w_calculate` receives data events from `w_source` and performs the moving relative range calculation using the specified algorithm parameters.

```

<window-calculate name='w_calculate' algorithm='MRR'>
<schema>
<fields>
<field name='id' type='int64' key='true' />
<field name='rangeOut' type='double' />
<field name='erangeOut' type='double' />
</fields>
</schema>
<parameters>
<properties>
<property name='rangeWindowLength'>3</property>
<property name='expRangeWindowLength'>3</property>
</properties>
</parameters>
<input-map>
<properties>
<property name="input">x1</property>
<property name="timeId">id</property>
</properties>
</input-map>
<output-map>
<properties>
<property name="timeIdOut">id</property>
<property name="erangeOut">erangeOut</property>
<property name="rangeOut">rangeOut</property>
  
```

4.9 Using Online Models

```

</properties>
</output-map>
<connectors>
...
</connectors>
</window-calculate>
    
```

The moving relative range algorithm is governed by the following properties:

Table 4- 63 Parameters

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
rangeWindowLength	variable	int64	Optional	128	Specifies the window length to calculate the range (M). For a time series whose mean is changing quickly, specify a lower value.
expRangeWindowLength	variable	int64	Optional	128	Specifies the window length to calculate the moving relative range (K). For a time series whose variance is changing quickly, specify a lower value.

Table 4- 64 Input Mapping

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
input	variable	double	Required	No default value	Specifies the analysis variable name in the input stream.
timeId	variable	int64	Required	No default value	Specifies the time ID variable name in the input stream.
timeIdOut	variable	int64	Required	No default value	Specifies the time ID (key) variable name in the output stream.

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
erangeOut	variable	double	Required	No default value	Specifies the expected range variable name in the output stream.
rangeOut	variable	double	Optional	"" (empty string)	Specifies the range variable name in the output stream.

The edge is defined at the end of the project.

```
<edges>
<edge source='w_source' target='w_calculate' role='data' />
</edges>
```

You can change the values of the properties that govern the MRR algorithm in the Calculate window while data is streaming through the model. First, create an edge between the Source window and the Calculate window with the role “request.” Then, stream a **reconfig** request and events that change the property values.

For example, to change the values of all of the properties for MRR:

```
i,n,1,"action","reconfig"
i,n,2,"rangeWindowLength","2"
i,n,3,"expRangeWindowLength","2"
i,n,4,,
```

These events immediately change property values. You can change one or more property values at a time, as required.

You can view the default values of the MRR algorithm parameter properties for the Calculate window with the command-line utility.

## 4.9.20 Using Term Frequency — Inverse Document Frequency (TFIDF)

Term frequency — inverse document frequency, or TFIDF, is a weight that shows how important a word is to a document in a document collection. TFIDF increases proportionally to the frequency with which a word appears in a document, but it is offset by the frequency with which the word appears in the document collection. You can use TFIDF as a weighing factor in text mining or general information searches.

TFIDF is composed by two terms:

- Term Frequency (TF), which measures how frequently a term occurs in a document. A term could appear more frequently in long documents than in short ones. Thus, TF is often normalized by dividing the number of times that a particular term occurs by the total number of terms in the document.
- Inverse Document Frequency (IDF), which measures the importance of a term. Some terms might occur a lot of times (for example, “is,” and “or”) but have little importance. IDF scales up rare terms and weighs down frequent terms.

$IDF(t) = \log(\text{total number of documents} / \text{number of documents that contain term } t + 1)$

TFIDF is as follows:

$$\text{TFIDF}(t) = \text{TF}(t) * \text{IDF}(t)$$

Consider the following example:



At the project level, a single continuous query includes the following:

- a Source window that receives output from a Calculate window that produced text tokenization results
- a Calculate window that runs the TFIDF algorithm

The Source window `w_source` receives input data. The input stream is placed into three fields for each observation: two key fields named `docId` and `tokenId` and a string named `token`.

```

<window-source name='w_source'>
<schema>
<fields>
<field name='docId' type='int64' key='true'/>
<field name='tokenId' type='int64' key='true'/>
<field name='token' type='string' key='false'/>
</fields>
</schema>
<connectors>
...
</connectors>
</window-source>
  
```

The Calculate window `w_calculate` receives data events and publishes calculated transforms according to the TFIDF algorithm properties that are specified.

```

<window-calculate algorithm="TFIDF" name="w_calculate">
<schema>
<fields>
<field key="true" name="docId" type="int64"/>
<field key="true" name="tokenId" type="int64"/>
<field key="false" name="token" type="string"/>
<field key="false" name="tf" type="double"/>
<field key="false" name="idf" type="double"/>
<field key="false" name="tfidf" type="double"/>
</fields>
</schema>
<input-map>
<properties>
<property name="docId">docId</property>
<property name="tokenId">tokenId</property>
<property name="token">token</property>
</properties>
</input-map>
<output-map>
<properties>
<property name="docIdOut">docIdOut</property>
<property name="tokenIdOut">tokenIdOut</property>
  
```

```

<property name="tokenOut">token</property>
<property name="tfOut">tf</property>
<property name="idfOut">idf</property>
<property name="tfidfOut">tfidf</property>
</properties>
</output-map>
<connectors>
...
</connectors>
</window-calculate>

```

The following properties govern the TFIDF algorithm in the Calculate window:

Table 4- 65 Parameters

Parameter	Type	Required or Optional?	Default Value	Description
startList	string	Optional	"" (empty string)	Specifies the file name of the start list, which contains the words that are evaluated during vectorization. The start list is a text file that lists one word per line. The start list is required whenever the dense vector is produced. The order of the words in the start list corresponds to the order that the TFIDF is represented in the vector output.
stopList	string	Optional	"" (empty string)	Specifies the file name of the stop list, which contains words that are ignored. The stop list is a text file that lists one word per line. <b>Note:</b> When a word exists in both the start list and the stop list, the word is ignored.
outputDenseVec	int64	Optional	"" (empty string)	Specifies whether word vectors or document vectors are returned. When set to 0, word vectors are returned. When set to 1, document vectors are returned.
outputVecType	string	Optional	"" (empty string)	Specifies whether term frequency (TF) or term frequency — inverse document frequency (TFIDF) is written to the document vectors.

Table 4- 66 Input Mapping

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
docId	variable	int64	Required	No default value	Specifies the input variable for the unique doc ID (key).
tokenId	variable	int64	Required	No default value	Specifies the input variable for the token ID (key).
token	variable	string	Required	No default value	Specifies the input token string.

Table 4- 67 Output Mapping

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
docIDOut	variable	int64	Required	No default value	Specifies the output variable for the unique doc ID (key).
tokenIDOut	variable	int64	Optional	"" (empty string)	Specifies the output variable for the unique ID of the token. It is a key when the word vectors are output. It is not a key when the document vectors are output.
tfOut	variable	double	Optional	"" (empty string)	Specifies the output variable for the term frequency (TF).
idfOut	variable	double	Optional	"" (empty string)	Specifies the output variable for the inverse document frequency (IDF).
tfidfOut	variable	double	Optional	"" (empty string)	Specifies the output variable for the TFIDF.
vectorOut	variable	varlist	Optional	"" (empty string)	Specifies a list of output variable names for document vectors.
tokenOut	variable	string	Optional	"" (empty string)	Specifies the output variable for the token.

The edge is defined at the end of the project. Streaming analytics windows require a role for each edge.

```
<edges>
<edge source='w_source' target='w_calculate' role='data' />
</edges>
```

You can view the parameters and the input and output mapping properties required to set up a streaming TFIDF project with the command-line utility.

### 4.9.21 Lag Monitoring

The lag monitoring algorithm computes the cross-correlation between a target time series and one or more additional time series. Results contain the selected lags and computed cross-correlation values that correspond to minimum, maximum, and maximum absolute value cross-correlations for each of the variables. The positive values in the `minLag` and `maxLag` parameters control the range of lags to be computed, and both positive and negative lags are computed. The `nDigits` parameter limits the cross-correlation value comparisons to the specified number of significant figures.

Consider the following example:



The continuous query includes the following:

- a Source window that receives the data to be analyzed
- a Calculate window that performs the LagMonitor calculation

The Source window `w_source` receives data that consists of an ID and a set of input variables (y1 through y7):

```
<window-source autogen-key="false" index="pi_EMPTY" insert-
only="true" name="w_source">
<schema>
<fields>
<field key="true" name="id" type="string" />
<field key="false" name="y1" type="double" />
<field key="false" name="y2" type="double" />
<field key="false" name="y3" type="double" />
<field key="false" name="y4" type="double" />
<field key="false" name="y5" type="double" />
<field key="false" name="y6" type="double" />
<field key="false" name="y7" type="double" />
</fields>
</schema>
<connectors>
...
</connectors>
</window-source>
```

The Calculate window `w_calculate` receives data events including the values of several variables. It publishes selected lags and computed cross-correlation values that correspond to minimum, maximum, and maximum absolute value cross-correlations for each of the variables.

```
<window-calculate algorithm="LagMonitor" name="w_calculate">
<schema>
<fields>
<field key="true" name="id" type="int64" />
<field key="false" name="y2" type="double" />
<field key="false" name="y1" type="double" />
<field key="false" name="absCCFOut" type="double" />
<field key="false" name="absLagOut" type="int64" />
<field key="false" name="maxCCFOut" type="double" />
<field key="false" name="maxLagOut" type="int64" />
<field key="false" name="minCCFOut" type="double" />
<field key="false" name="minLagOut" type="int64" />
<field key="false" name="numComputedLagsOut" type="int64" />
</fields>
</schema>
<parameters>
<properties>
<property name="maxLag">500</property>
<property name="minLag">1</property>
<property name="windowLength">10000</property>
</properties>
</parameters>
<input-map>
<properties>
<property name="input">y2</property>
<property name="target">y1</property>
<property name="timeId">id</property>
</properties>
</input-map>
<output-map>
```

```

<properties>
<property name="absCCFOut">absCCFOut</property>
<property name="absLagOut">absLagOut</property>
<property name="maxCCFOut">maxCCFOut</property>
<property name="maxLagOut">maxLagOut</property>
<property name="minCCFOut">minCCFOut</property>
<property name="minLagOut">minLagOut</property>
<property name="numComputedLagsOut">numComputedLagsOut</property>
<property name="timeIdOut">id</property>
</properties>
</output-map>
<connectors>
...
</connectors>
</window-calculate>

```

The lag monitoring algorithm is governed by the following properties:

Table 4- 68 Algorithm Parameters

Name	Type	Required or Optional?	Default Value	Description
windowLength	int64	Optional	128	Specifies the length of the sliding window. The value that you specify must be greater than the value you specify for <code>overlap</code> .
overlap	int64	Optional	127	Specifies the overlap between consecutive windows. Must be less than <code>windowLength</code> .
minLag	int64	Optional	1	Specifies the minimum lag to consider.
maxLag	int64	Optional	10	Specifies the maximum lag to consider.
nDigits	int64	Optional	0	Specifies the number of significant digits to use when comparing cross-correlation values (0–16).

Table 4- 69 Input Mapping

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
input	variable	double	Required	NA	Specifies the analysis variable name in the input stream.
target	variable	double	Required	NA	Specifies the target variable name in the input stream.
timeId	variable	int64	Required	NA	Specifies the time ID variable name in the input stream.

Table 4- 70 Output Mapping

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
timeIDout	variable	int64	Required	NA	Specifies the time ID variable name in the output stream.
minLagOut	variable	int64	Optional	"" (empty string)	Specifies the variable name for the lag that corresponds to minimum cross-correlation.
minCCFOut	variable	double	Optional	"" (empty string)	Specifies the variable name for minimum cross-correlation.
maxLagOut	variable	int64	Optional	"" (empty string)	Specifies the variable name for lag that corresponds to maximum cross-correlation.
maxCCFOut	variable	double	Optional	"" (empty string)	Specifies the variable name for maximum cross-correlation.
absLagOut	variable	int64	Optional	"" (empty string)	Specifies the variable name for lag that corresponds to maximum absolute cross-correlation.

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
absCCFOut	variable	double	Optional	"" (empty string)	Specifies the variable name for maximum absolute cross-correlation.
numComputedLagsOut	variable	int64	Optional	"" (empty string)	Specifies the variable name for the number of computed lags in the output stream.

The edge is defined at the end of the project.

```
<edges>
<edge role="data" source="w_source" target="w_calculate" />
</edges>
```

You can change the values of the properties that govern the lag monitoring algorithm in the Calculate window while data is streaming through the model. First, create an edge between the Source window and the Calculate window with the role "request." Then, stream a **reconfig** request and events that change the property values.

For example, to change the values of all of the properties for lag monitoring:

```
i,n,1,"action","reconfig"
i,n,2,"windowLength","64"
i,n,3,"overlap","32"
i,n,4,"minLag","1"
i,n,5,"maxLag","32"
i,n,6,"nDigits","0" i,n,7,,
```

These events immediately change property values. You can change one or more property values at a time, as required.

You can view the default values of the lag monitoring algorithm parameter properties for the Calculate window with the command-line utility.

### 4.9.22 Applying a Cepstrum Transform

A *cepstrum* results from taking the inverse Fourier transform of the logarithm of the estimated spectrum of a signal. This application is often used in speech analysis, specifically voice pitch detection. Two variations are supported: a complex cepstrum and a real cepstrum. For more information, see SAS Visual Forecasting: Time Series Packages.

Consider the following example:



The continuous query includes the following:

- a Source window that receives the data to be analyzed
- a Calculate window that performs the cepstrum calculation

The Source window `w_source` receives a data event.

```
<window-source name='w_source' insert-only='true' autogen-
key='true'>
<schema>
<fields>
<field name='datetime' type='int64' key='true' />
<field name='x' type='double' />
</fields>
</schema>
<connectors>
...
</connectors>
</window-source>
```

The Calculate window `w_calculate` receives data events. It calculates the specified cepstrum.

```
<window-calculate name="w_calculate" algorithm="Cepstrum">
<schema>
<fields>
<field name='datetime' type='int64' key='true' />
<field name='y' type='array(dbl)' />
</fields>
</schema>
<parameters>
<properties>
<property name="windowLength">128</property>
<property name="windowType">1</property>
<property name="windowParam">-1.0</property>
<property name="overlap">0</property>
<property name="complexCepstrum">1</property>
</properties>
</parameters>
<input-map>
<properties>
<property name="input">x</property>
<property name="timeId">datetime</property>
</properties>
</input-map>
<output-map>
<properties>
<property name="timeIdOut">datetime</property>
<property name="cepstrumListOut">y[1-128]</property>
</properties>
</output-map>
<connectors>
...
</connectors>
</window-calculate>
```

The following properties govern the Cepstrum algorithm:

Table 4- 71 Parameters

Name	Value Type	Required or Optional?	Default Value	Description
windowLength	int64	Optional	128	Specifies the length of the sliding window. The value that you specify must be greater than the value you specify for <code>overlap</code> .
windowType	int64	Optional	15	Specifies the type of the window. Options: 1=Bartlett, 2=Bohman, 3=Chebyshev, 4=Gaussian, 5=Kaiser, 6=Parzen, 7=Rectangular, 10=Tukey, 11=Bartlett-Hann, 12=Blackman-Harris, 13=Blackman, 14=Hamming, 15=Hanning, 16=Flat Top For more information about these window types, see SAS Visual Forecasting: Time Series Packages.
windowParam		Optional	-1.0	Some window types require an additional parameter. If not required for the window type selected, this value is ignored.
overlap	int64	Optional	127	Overlap between consecutive windows. Must be strictly less than the value of <code>windowLength</code> .
complexCepstrum	int64	Optional	0	When set to 1, calculate the complex cepstrum. When set to 0 or unspecified, calculate the real cepstrum.

Table 4- 72 Input Mapping

Name	Type	Variable Type	Required or Optional?	Default Value	Description
input	variable	double	Required	No default value	Name of analysis variable in input stream.
timeId	variable	int64	Required	No default value	Name of time ID variable in input stream.

Table 4- 73 Output Mapping

Name	Type	Variable Type	Required or Optional?	Default Value	Description
timeIdOut	variable	int64	Required	No default value	Name of time ID variable in output stream.
cepstrumListOut	varlist	double	Optional	"" (empty string)	List of cepstrum variables in the output stream. Cepstrum variable values are always real numbers. The algorithm writes <code>windowLength</code> values for each output event.

The edge is defined at the end of the project.

```
<edges>
<edge source='w_source' target='w_calculate' role='data' />
</edges>
```

You can change the values of the properties that govern the Cepstrum algorithm in the Calculate window while data is streaming through the model. First, create an edge between the Source window and the Calculate window with the role "request." Then, stream a **reconfig** request and events that change the property values.

For example, to change the values of all of the properties for Cepstrum:

```
i,n,1,"action","reconfig"
i,n,2,"windowLength","32"
i,n,3,"overlap","16"
i,n,4,"windowType","15"
i,n,5,"windowParam","-1.0"
i,n,6,"complexCepstrum","0" i,n,7,,
```

These events immediately change property values. You can change one or more property values at a time, as required.

You can view the default values of the Cepstrum algorithm parameter properties for the Calculate window with the command-line utility.

### 4.9.23 Video Encoding

The video-encoding algorithm parses image data in BayerRG8 format to a more common image format. JPEG is the default output format, and PNG is also supported.

---

#### Note

This encoder supports only data streamed from the Pylon Publisher adapter.

---

Consider the following example:



The continuous query includes the following:

- a Source window that receives a live video feed from a camera adapter (Pylon Publisher)
- a Calculate window that runs the video-encoding algorithm

The Source window `w_source` receives a data event that consists of video data in blob format. It receives the event through a Pylon connector.

```

<window-source index="pi_EMPTY" insert-only="true" name="w_source">
<schema copy-keys="false">
<fields>
<field key="true" name="id" type="int64" />
<field key="false" name="image" type="blob" />
</fields>
</schema>
<connectors>
<connector class="pylon" name="pub" type="publish">
<properties>
<property name="cameraheight">224</property>
<property name="cameraipaddress">10.40.19.48</property>
<property name="camerawidth">200</property>
<property name="maxframerate">1</property>
</properties>
</connector>
</connectors>
</window-source>
  
```

The Calculate window `w_calculate` receives data events that consist of image data. It publishes encoded images.

```

<window-calculate algorithm="VideoEncoding" name="w_calculate">
<schema copy-keys="false">
<fields>
<field key="true" name="id" type="int64" />
<field key="false" name="image" type="blob" />
<field key="false" name="encodedImage" type="blob" />
</fields>
</schema>
<parameters>
  
```

```

<properties>
<property name="height">224</property>
<property name="inputFormat">bayerrg8</property>
<property name="outputColorFormat">rgb</property>
<property name="outputFormat">wide</property>
<property name="width">225</property>
</properties>
</parameters>
<input-map>
<properties> <property name="videoBuffer">image</property>
</properties>
</input-map>
<output-map>
<properties>
<property name="imageOut">encodedImage</property>
</properties>
</output-map>
</window-calculate>

```

BayerRG8 is a common video-encoding format not suitable for image processing. A Calculate window can convert BayerRG8 to PNG, JPG, or wide. JPEG provides fast but lossy image compression. PNG provides lossless image compression, but it is slower than JPEG. Wide specifies no image compression. After converting the video encoding to one of these formats, you can perform object detection or image classification on the resulting image.

The width and height of the encoding window should exactly match the specifications of the adapter. In other words, if the camera is feeding a 600x800 video image, then the width and height parameters should be set to 600 and 800.

The video-encoding algorithm is governed by the following properties:

Table 4- 74 Parameters

Name	Type	Required or Optional?	Default Value	Description
inputFormat	string	Required	bayerrg8	Specifies the input format of the source feed.
outputColorFormat	string	Optional	rgb	Specifies the output color space. Valid values are <b>rgb</b> and <b>bgr</b> .
outputFormat	string	Optional	jpg	Specifies the type of output compression. Other values are: <ul style="list-style-type: none"> <li>• wide: no compression. This is the fastest format.</li> <li>• png; lossless compression. Slower than jpg.</li> </ul>

Name	Type	Required or Optional?	Default Value	Description
width	int64	Required	0	Specifies the width of the input video buffer. Specify a size to match the size specified in the connector in the Source window.
height	int64	Required	0	Specifies the height of the input video buffer. Specify a size to match the size specified in the connector in the Source window.

Table 4- 75 Input Mapping

Name	Type	Variable Type	Required or Optional?	Default Value	Description
videoBuffer	variable	blob	Required	NA	Specifies the name of the input video buffer

Table 4- 76 Output Mapping

Name	Type	Variable Type	Required or Optional?	Default Value	Description
imageOut	variable	blob	Required	NA	Specifies the name of the output image

The edge is defined at the end of the project.

```
<edges>
<edge role="data" source="w_source" target="w_calculate" />
</edges>
```

You can view the default values of the video–encoding algorithm parameter properties for the Calculate window with the command-line utility.

### 4.9.24 Subspace Tracking (SST)

Suppose that data contains a sequence of  $n \times 1$  vectors:  $x(t)$ . Subspace tracking (SST) estimates the covariance matrix for each vector  $x(t)$  and then computes the first  $p$  principal eigenvectors of the covariance matrix. For each iteration at time  $t$ , the covariance matrix  $C(t)$ s obtained by the following:

$$\begin{aligned} \mu(t) &= (1 - \alpha)\mu(t - 1) + \alpha x(t) \\ C(t) &= (1 - \beta)C(t - 1) + \beta (x(t) - \mu(t))(x(t) - \mu(t))^T \end{aligned}$$

Here,  $\alpha$  and  $\beta$  are the mean and covariance forgetting factors whose values are predetermined to be between 0 and 1, respectively. The first  $p$  principal eigenvector  $W$  can be obtained by the eigendecomposition of the covariance matrix.

SST can be applied to industrial data to detect outliers and use results to identify potential errors before they occur.

Consider the following example:



The continuous query includes the following:

- a Source window that receives the data to analyze
- a Calculate window that performs SST

The Source window `w_source` receives input data. The input stream is placed into three fields for each observation: an ID that acts as the data stream's key, named `id`; and a series of six different  $x$  coordinate fields (`x1`, `x2`, `x3`, `x4`, `x5`, `x6`).

```

<window-source name='w_source'>
<schema>
<fields>
<field name='id' type='int64' key='true' />
<field name='x1' type='double' />
<field name='x2' type='double' />
<field name='x3' type='double' />
<field name='x4' type='double' />
<field name='x5' type='double' />
<field name='x6' type='double' />
</fields>
</schema>
<connectors>
...
</connectors>
</window-source>
  
```

The Calculate window `w_calculate` receives data events from `w_source`. It publishes the calculated principal components and output values of the subspace according to the SST algorithm properties that are specified.

```

<window-calculate name='w_calculate' algorithm='SST'>
<schema>
<fields>
<field name='id' type='int64' key='true' />
<field name='projAngle' type='double' />
<field name='residualRate' type='double' />
<field name='numRank' type='int32' />
<field name='prin1_x1' type='double' />
<field name='prin1_x2' type='double' />
<field name='prin1_x3' type='double' />
<field name='prin1_x4' type='double' />
<field name='prin1_x5' type='double' />
</fields>
</window-calculate>
  
```

```
<field name='prin1_x6' type='double' />
</fields>
</schema>
<parameters>
<properties>
<property name="maxPrincipal">2</property>
<property name="meanForgetFactor">0.1</property>
<property name="covForgetFactor">0.5</property>
<property name="eigvalTolCumulative">1</property>
</properties>
</parameters>
<input-map>
<properties>
<property name="inputs">x_c, y_c</property>
</properties>
</input-map>
<output-map>
<properties>
<property name="projAngleOut">projAngle</property>
<property name="residualOut">residualRate</property>
<property name="numRankOut">numRank</property>
<property name="principalVecOut">prin1_x1, prin1_x2, prin1_x3,
prin1_x4, prin1_x5, prin1_x6</ </properties>
</output-map>
<connectors>
...
</connectors>
</window-calculate>
```

The following properties govern the SST algorithm:

Table 4- 77 Parameters

Name	Value Type	Required or Optional?	Default Value	Description
windowLength	int64	Optional	0	Specifies the length of the sliding window. A sliding window enables you to use multiple events to update principal components. A value of 0 denotes unlimited length. If the value is greater than 0, <code>covForgetFactor</code> and <code>meanForgetFactor</code> are ignored for updating the covariance matrix. The value that you specify must be greater than the value that you specify for <code>overlap</code> .
overlap	int64	Optional	-1	Specifies the overlap between consecutive windows. Must be strictly less than <code>windowLength</code> . The default value of -1 means that overlap is internally calculated as <code>windowLength-1</code> .
maxPrincipal	int32	Optional	1	Specifies the maximum number of the principal eigenvectors. Specify a value greater than 0 and less than or equal to the number of input variables.
covForgetFactor	double	Optional	0.5	Specifies the forgetting factor that is used to update the covariance matrix. Specify a value between 0 and 1.
meanForgetFactor	double	Optional	0.1	Specifies the value of the forgetting factor used to update the mean. Specify a value between 0 and 1.
eigvalTolCumulative	double	Optional	1	Specifies the threshold on the cumulative rate of eigenvalues. Specify a positive value less than or equal to 1.

Table 4- 78 Input Mapping

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
inputs	varlist	double	Required	No default value	Specifies the list of variable names used to compute the principal subspace.

Table 4- 79 Output Mapping

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
PCAngleChangeOut	variable	double	Optional	"" (empty string)	Specifies the output variable name for the angle change between the first principal component vector of two consecutive subspaces.
PCAbsoluteAngleOut	variable	double	Optional	"" (empty string)	Specifies the output variable name for the absolute angle of the first principal component vector.
projAngleOut	variable	double	Optional	"" (empty string)	Specifies the name of the projection angle.
numRankOut	variable	int32	Optional	"" (empty string)	Specifies the name of the rank of the principal subspace.

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
principalVecOut	varlist	double	Optional	"" (empty string)	<p>Specifies a list of variable names that correspond to the elements of each principal eigenvector.</p> <p>Suppose that the data consists of five variables: x1, x2, x3, x4, and x5. If you want to produce the third principal eigenvector, prin3, then you must specify the variable names as follows:</p> <pre>prin1_x1, prin1_x2, prin1_x3, prin1_x4, prin1_x5, prin2_x1, prin2_x2, prin2_x3, prin2_x4, prin2_x5, prin3_x1, prin3_x2, prin3_x3, prin3_x4, and prin3_x5.</pre> <p>That is, you must specify all of the elements of the first and second principal eigenvector to produce the elements of the third. Often, the first principal eigenvector provides the most useful information.</p>
residualOut	variable	double	Optional	"" (empty string)	Specifies the output name of the residual.

The edge is defined at the end of the project. Streaming analytics windows require a role for each edge.

```
<edges>
<edge source='w_source' target='w_calculate' role='data' />
</edges>
```

You can view the default values of the SST algorithm parameter properties for the Calculate window with the command-line utility. You can change the values of the properties that govern the SST algorithm in the Calculate window while data is streaming through the

model. First, create an edge between the Source window and the Calculate window with the role "request." Then, stream a reconfig request and events that change the property values.

For example, to change the values of all of the properties for SST:

```
i,n,1,"action","reconfig"
i,n,2,"windowLength","64"
i,n,3,"maxPrincipal","2"
i,n,4,"covForgetFactor","1"
i,n,5,"meanForgetFactor","1"
i,n,6,"eigvalTolCumulative","0.75"
i,n,7,,
```

These events immediately change property values. You can change one or more property values at a time, as required.

### 4.9.25 Converting Speech to Text

#### Conversion Overview

Edge Streaming Analytics provides two algorithms to enable conversion of speech to text: Audio Feature Computation and Transcription. You use these two algorithms together in order to convert audio input into a transcript.

Consider the following example:

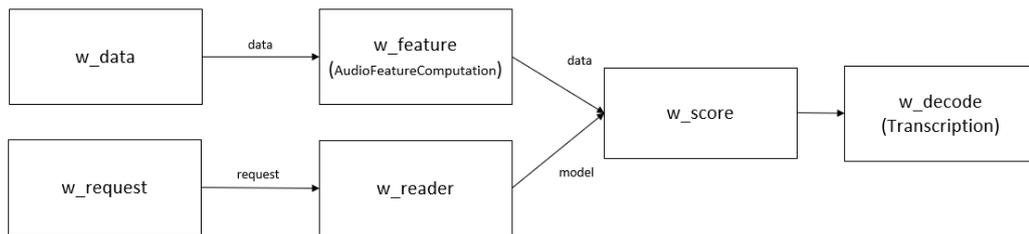


Figure 4-3 Audio to Speech

- A Source window (`w_data`) streams binary audio data into a Calculate window that processes it with the Audio Feature Computation algorithm. For more information, see “Streaming Audio Feature Computation”.
- Another Source window (`w_request`) sends a request event into a Model Reader window to inject an analytic store file that contains a pre-trained acoustic model.
- The Audio Feature Computation results and the model in the analytic store file are streamed into a Score window (`w_score`).
- Results from the Score window are streamed to a Calculate window (`w_decode`) that applies the Transcription algorithm in order to produce the transcript. For more information, see “Streaming Speech Transcription”.

Here is the Source window that sends the request event and the Model Reader window that receives it:

```
<window-source name='w_request' insert-only='true' index='pi_EMPTY'>
<schema>
<fields>
```

```

<field name='req_id' type='int64' key='true' />
<field name='req_key' type='string' />
<field name='req_val' type='string' />
</fields>
</schema>
<connectors>
</connectors>
</window-source>
<window-model-reader name='w_reader' model-type='astore' />

```

Here is the edge between those two windows:

```
<edge source='w_request' target='w_reader' role='request' />
```

The Model Reader window uses a model event to stream the analytic store file into a Score window (`w_score`) that then streams its results into a Calculate window running the Transcription algorithm.

```
<edge source='w_score' target='w_decode' role='data' />
```

For more information about how to train an RNN acoustic model with the Language Model action set, see *SAS Visual Data Mining and Machine Learning: Programming Guide*.

### Streaming Audio Feature Computation

Audio feature computation is the process of applying an algorithm to an audio signal in order to convert it into a sequence of acoustic feature vectors. These vectors contain a serviceable numerical representation of the audio signal. You can perform further analytics using these acoustic feature vectors as input.

---

#### Note

The audio feature computation algorithm is supported on Linux (x86 architecture) and Microsoft Windows systems.

---

Consider the following example:

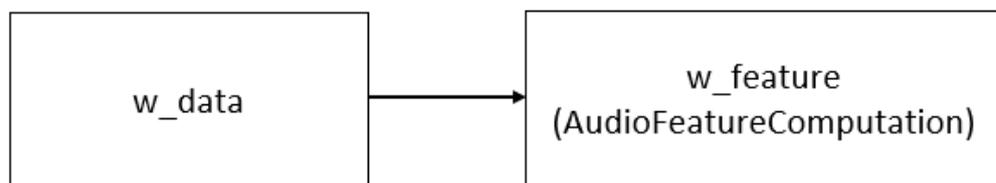


Figure 4-4 Applying the Audio Feature Computation Algorithm

This continuous query includes the following:

- a Source window that provides audio data to be analyzed
- a Calculate window that processes the audio data and converts it into a sequence of feature vectors

The Source window `w_data` creates streaming data events based on input data from a CSV file.

- The first field ( `path` ) denotes where the file originated. It might be useful to propagate this value along the entire processing chain. For example, in a speech-to-text application,

it might be useful to maintain the connection between the final transcripts and the source audio files.

- The second field (`audio`) references a binary large object (`blob`) that comprises the audio in the current streaming data event.

```
<window-source name='w_data' insert-only='true' index='pi_EMPTY'>
<schema>
<fields>
<field name='_path_' type='string' key='true'/>
<field name='audio' type='blob'/>
</fields>
</schema>
<connectors>
...
</connectors>
</window-source>
```

The Calculate window `w_feature` receives those data events and processes the audio data with the audio feature computation algorithm. The parameters chosen enable you to work with Mel Frequency Cepstral Coefficient (MFCC) features that consist of 40 coefficients for each 25 millisecond long frame.

```
<window-calculate name="w_feature" output-insert-only='true'
produces-only-inserts='true' index='pi_EMPTY'
algorithm="AudioFeatureComputation">
<schema>
<fields>
<field name='_path_' type='string' key='true'/>
<field name='_num_frames_' type='int64'/>
<field name='f' type='array(dbl)'/>
</fields>
</schema>
<parameters>
<properties>
<property name="frameExtractionFrameShift">10</property>
<property name="frameExtractionFrameLength">25</property>
<property name="frameExtractionDither">0</property>
<property name="melBanksNBins">40</property>
<property name="mfccNCeps">40</property>
<property name="featureScalingMethod">STANDARDIZATION</property>
<property name="nOutputFrames">3500</property>
</properties>
</parameters>
<input-map>
<properties>
<property name="audioIn">audio</property>
</properties>
</input-map>
<output-map>
<properties>
<property name="numFramesOut">_num_frames_</property>
<property name="computedFeatureValuesOut">f</property>
</properties>
</output-map>
<connectors>
...

```

```
</connectors>
</window-calculate>
```

The following properties govern the streaming audio feature computation algorithm:

Table 4- 80 Parameters

Parameter	Type	Required or Optional?	Default Value	Description
frameExtraction-frameShift	double	Optional	10	Specifies the time difference (in milliseconds) between the beginning of consecutive frames. This value must be greater than 0.
frameExtraction-frameLength	double	Optional	25	Specifies the length of a frame (in milliseconds). This value must be greater than 0.
frameExtractionDither	double	Optional	1.0	Specifies the dithering constant. A value of 0.0 means no dithering. This value must be greater than or equal to 0.
frameExtraction-PreemphCoeff	double	Optional	0.97	Specifies the coefficient used in performing signal preemphasis. This value must be greater than or equal to 0 and less than or equal to 1.
frameExtractionRemoveDcOffset	int32	Optional	1	Specifies whether the mean signal value should be subtracted from all frames. Specify 0 to represent false or 1 to represent true.

Parameter	Type	Required or Optional?	Default Value	Description
frameExtraction-WindowType	string	Optional	<b>RECTANGULAR</b>	Specifies the type of window to apply to each frame upon extraction. Valid values are <b>HAMMING</b> , <b>HANNING</b> , <b>RECTANGULAR</b> , and <b>BLACKMAN</b> .
frameExtractionRoundToPowerOfTwo	int32	Optional	<b>1</b>	Specifies whether the window size should be rounded to the next power of two. Specify <b>0</b> to represent false or <b>1</b> to represent true.
frameExtractionBlackmanCoeff	double	Optional	<b>0.42</b>	Specifies the constant coefficient used for the generalized Blackman window. This value is ignored for other window types.
frameExtractionSnipEdges	int32	Optional	<b>1</b>	Specifies whether end effects should be handled by writing only frames that completely fit the data. Specify <b>0</b> to represent false or <b>1</b> to represent true.
melBanksNBins	int32	Optional	<b>23</b>	Specifies the number of triangular Mel Frequency bins. This value must be greater than or equal to <b>1</b>
melBanksLowFreq	double	Optional	<b>20</b>	Specifies the low cutoff frequency for the Mel Frequency bins.

Parameter	Type	Required or Optional?	Default Value	Description
melBanksHighFreq	double	Optional	0	Specifies the high cut-off frequency for the Mel Frequency bins. When negative, the value is added to the Nyquist frequency, which is half of the sampling rate of the audio.
computeFbankFeatures	double	Optional	0	Specifies whether to perform FBank feature computations. Specify 0 to represent false or 1 to represent true.
fbankUseEnergy	int32	Optional	0	Specifies whether an extra dimension that contains the computed energy should be appended to each FBank feature frame. Specify 0 to represent false or 1 to represent true.
fbankEnergyFloor	double	Optional	0	Specifies the linear floor on energy (absolute, not relative) for the FBank feature computations. This value must be greater than or equal to 0.
fbankRawEnergy	int32	Optional	1	Specifies whether energy should be computed before preemphasis and windowing. Specify 0 to represent false or 1 to represent true.

Parameter	Type	Required or Optional?	Default Value	Description
fbankUseLogFbank	int32	Optional	1	Specifies whether the output should contain log- filterbank values. Otherwise, the output values are linear. Specify 0 to represent false or 1 to represent true.
fbankUsePower	int32	Optional	1	Specifies whether power should be used in the FBank feature computations. Otherwise, the magnitude is used. Specify 0 to represent false or 1 to represent true.
computeMfccFeatures	double	Optional	0	Specifies whether to perform mel frequency cepstral coefficients (MFCC) feature computations. Specify 0 to represent false or 1 to represent true.
mfccNCeps	int32	Optional	13	Specifies the number of cepstral coefficients in each frame, including C0. This value must be greater than or equal to 1 and less than or equal to $\text{mel-BanksNBins}$ .
mfccUseEnergy	int32	Optional	1	Specifies whether energy (not C0) should be used in the MFCC feature computations. Specify 0 to represent false or 1 to represent true.

Parameter	Type	Required or Optional?	Default Value	Description
<code>mfccEnergyFloor</code>	double	Optional	<b>0</b>	Specifies the linear floor on energy (absolute, not relative) for the MFCC feature computations. This value must be greater than or equal to <b>0</b> .
<code>mfccRawEnergy</code>				
<code>plpCompressFactor</code>	double	Optional	<b>0.33333</b>	Specifies the compression factor used in the PLP feature values. This value must be greater than 0.
<code>mfccCepstralLifter</code>	double	Optional	<b>22</b>	Specifies the constant that controls the scaling of the MFCC feature values.
<code>computePlpFeatures</code>	double	Optional	<b>0</b>	Specifies whether to perform perceptual linear prediction (PLP) feature computations. Specify <b>0</b> to represent false or <b>1</b> to represent true.
<code>plpLpcOrder</code>	int32	Optional	<b>12</b>	Specifies the order of the linear predictive coding (LPC) analysis used in the PLP feature computations. This value must be greater than or equal to <b>1</b> and less than or equal to <b>25</b> .
<code>plpNCeps</code>	int32	Optional	<b>13</b>	Specifies the number of cepstral coefficients in each PLP feature frame, including C0. This value must be greater than or equal to <b>1</b> and less than or equal to <code>plpLpcOrder + 1</code> .

Parameter	Type	Required or Optional?	Default Value	Description
plpUseEnergy	int32	Optional	1	Specifies whether energy (not C0) should be used for the zeroth feature value in the PLP feature computations. Specify <b>0</b> to represent false or <b>1</b> to represent true.
plpEnergyFloor	double	Optional	0	Specifies the linear floor on energy (absolute, not relative) for the PLP feature computations. This value must be greater than or equal to <b>0</b> .
plpRawEnergy	int32	Optional	1	Specifies whether energy should be computed before preemphasis and windowing. Specify <b>0</b> to represent false or <b>1</b> to represent true.
plpCompressFactor	double	Optional	0.33333	Specifies the compression factor used in the PLP feature computations. This value must be greater than or equal to <b>0</b> and less than <b>1</b> .
plpCepstralLifter	int32	Optional	22	Specifies the constant that controls the scaling of the PLP feature values. This value must be greater than or equal to <b>0</b> .
plpCepstralScale	double	Optional	1	Specifies the cepstral scaling constant used in the PLP computation. This value must be greater than <b>0</b> .

Parameter	Type	Required or Optional?	Default Value	Description
<code>featureScaling-Method</code>	string	Optional	<b>NONE</b>	Specifies the feature scaling method to apply to the computed feature vectors. Valid values are <b>NONE</b> and <b>STANDARDIZATION</b> .
<code>nOutputFrames</code>	int32	Optional	computed	Specifies the exact number of frames to include in the output. Extra frames are dropped and missing frames are padded with zeros. When not explicitly specified, this value is set to the minimum number of frames required to contain all of the feature values for the incoming audio file. This value must be greater than or equal to <b>1</b> .
<code>nContextFrames</code>	int32	Optional	<b>0</b>	Specifies the number of context frames to append before and after the current audio frame. This value must be greater than or equal to <b>0</b> .

Table 4- 81 Input Mapping

Name	Type	Variable Type	Required or Optional?	Default Value	Description
<code>audioIn</code>	variable	blob	Required	No default value	Specifies the name of the variable that contains the incoming audio data.

Table 4- 82 Output Mapping

Name	Type	Variable Type	Required or Optional?	Default Value	Description
numFramesOut	variable	int64	Required	No default value	Specifies the name of the variable that tracks the valid number of output frames.
computedFeatureValuesOut	variable	array(dbl)	Required	No default value	Specifies the name of the variable that contains the computed feature values.

Edges are defined at the end of the project. The Source window streams data into the Calculate window:

```
<edge source='w_data' target='w_feature' role='data' />
```

The Calculate window (`w_feature`) streams its results to a Score window (`w_score`).

```
<edge source='w_feature' target='w_score' role='data' />
```

This Score window streams its results into another Calculate window that applies the Transcription algorithm.

### Streaming Speech Transcription

Streaming speech transcription is the process taking the output of an acoustic model and applying the natural structure and patterns of language in order to generate a final transcript. Edge Streaming Analytics provides the Transcription algorithm to implement streaming speech transcription. Two models are involved in data analysis:

- An `acoustic model` that associates acoustic feature vectors with the linguistic units that comprise the spoken utterance. This model is typically pre-trained with a large speech corpus. For all acoustic feature vectors, the model computes the probability that a given feature vector corresponds to each individual linguistic unit.
- A `language model` that is responsible for modeling word sequences in language. Basically, this model is a probability distribution. When given a specific sequence of words, it assigns a probability to the entire sequence. It then estimates the likelihood of a given phrase. When you build language models, you often assume that the probability of the occurrence of a particular word depends on the previous words in the sequence. The number of previous words that you use to determine this probability is essentially arbitrary. This leads to the idea of using an `n-gram model` to build language models. For example, a trigram (3-gram) model uses the previous two words to predict the occurrence of a particular word.

Before running the Transcription algorithm, you must supply the pre-trained acoustic model as an analytic store (ASTORE) file that is the result of a deep learning model. After an audio file has been processed by the Audio Feature Computation algorithm, it is represented as a set of vectors. Together, the analytic store file and these vectors are supplied to a score windows as shown in “Conversion Overview”.

Consider these windows:

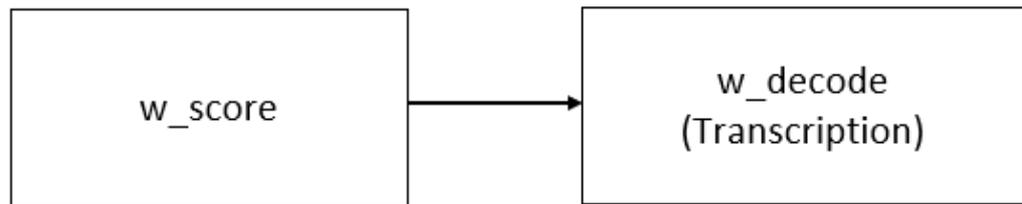


Figure 4-5 Applying the Transcription Algorithm

- a Score window uses the acoustic model contained in an analytic store file and results from the Audio Feature Computation algorithm to generate an array of score data
- a Calculate window performs the speech transcription

The Score window `w_score` receives a model event from `w_reader`. In the input schema, the variable `_path_` specifies where to find the audio data file that has been transcribed. The variable `v` represents a vector that contains a set of probability distributions across time frames received from `w_feature`. This variable is required.

```

<window-score name='w_score'>
<schema>
<fields>
<field name='_path_' type='string' key='true' />
<field name='v' type='array/dbl' />
</fields>
</schema>
<models>
<offline model-type='astore'>
<input-map>
<properties>
<property name="inputDblArray">f</property>
</properties>
</input-map>
<output-map>
<properties>
<property name="outputDblArray">v</property>
</properties>
</output-map>
</offline>
</models>
</window-score>
  
```

The Calculate window `w_decode` applies the Transcription algorithm to the data.

```

<window-calculate name='w_decode' algorithm='Transcription'>
<schema>
<fields>
<field name='_path_' type='string' key='true' />
<field name='_audio_content_' type='string' />
</fields>
</schema>
<parameters>
<properties>
<property name="langModelPath">path/filename</property>
<property name="columnMapPath">path/filename</property>
<property name="blankLabel">&#x20;</property>
<property name="spaceLabel">&amp;</property>
</properties>
  
```

```
<property name="nFrames">3500</property>
<property name="alpha">2.5</property>
<property name="beta">3.5</property>
</properties>
</parameters>
<input-map>
<properties>
<property name="inputs">v</property>
</properties>
</input-map>
<output-map>
<properties>
<property name="transOut">_audio_content_</property>
</properties>
</output-map>
<connectors>
...
</connectors>
</window-calculate>
```

The following properties govern the speech transcription algorithm in the Calculate window:

Table 4- 83 Parameters

Parameter	Type	Required or Optional?	Default Value	Description
langModelPath	string	Required	No default value	<p>Specifies the path of the language model file, which is in CSV format. Each row in the file except the header represents an N-gram term. This term refers to a continuous sequence of n words and the information that it contains.</p> <p>The number in the first column of each row is the log probability of that row's N-gram term. The last column contains the value of the back-off weight. The words that each N-gram term contains are listed in order, starting from the second column, with one word per column.</p>
columnMapPath	string	Required	No default value	<p>Specifies the path of the column map file, which is in CSV format. Each row in the file except the header specifies the mapping between an index and a label.</p> <p>For example, suppose that the first row shows the label "A". In the <code>inputs</code> that stream into the Calculate window, the first score in each time frame indicates the predicted score for label "A".</p>

Parameter	Type	Required or Optional?	Default Value	Description
nFrames	int64	Required	No default value	<p>Specifies the maximum number of time frames in an audio that are used to extract acoustic features. Specify an integer value.</p> <p>The value of nFrames is determined by the input mapping parameter inputs. The value that you specify must be the same as the number of consecutive time frames that inputs provides the probability distribution over labels.</p> <p>If score data arriving at the Calculate window are generated by a Score window, then the model used by that Score window must be trained on the same features. It must also have the same number of time frames.</p> <p>When the features coming into the Score window are generated from a Calculate window that uses the AudioFeatureComputation algorithm, the value of nFrames must be the same value as the nOutputFrames parameter.</p>
blankLabel	string	Optional	-	<p>Specifies the string used to indicate the 'blank' label. The value should match one of the rows from the column map file that is specified by columnMapPath.</p>

Parameter	Type	Required or Optional?	Default Value	Description
spaceLabel	string	Optional	–	Specifies the string used to indicate the 'space' label. The value should match one of the rows from the column map file that is specified by <code>columnMapPath</code> .
alpha	double	Optional	1.0	Specifies a tunable parameter that strengthens or weakens the influence of the language model on results.  When alpha is large, the language model heavily influences transcription. Conversely, when alpha is small, the transcription depends more heavily on the acoustic model.
beta	double	Optional	0.0	Specifies a tunable parameter that strengthens or weakens the influence of the length of a sentence on results.  When you choose a large value of <code>beta</code> , the transcription includes more words. A small value of <code>beta</code> leads to little dependency on sentence length, which leads to a transcription of fewer words.

Parameter	Type	Required or Optional?	Default Value	Description
maxPathSize	int64	Optional	100	<p>Specifies the maximum number of paths kept as candidates of the final result during the decoding process. Specify a positive integer.</p> <p>When you choose a large value of maxPathSize, more candidates of transcripts are considered. This can lead to better results. When you choose a small value of maxPathSize, fewer candidates of word sequences are considered. This can lead to faster processing speeds.</p>
ngramsOrder	int64	Optional	3	<p>Specifies the highest order of N-grams to use during decoding process. Specify a positive integer no less than 1.</p>

Table 4- 84 Input Mapping

Name	Type	Variable Type	Required or Optional?	Default Value	Description
inputs	varlist	array (double)	Required	No default value	<p>Specifies name of the array that contains the incoming score data. This array contains scores returned by an acoustic model. The scores indicate the probability distribution over labels in every time frame. The Calculate window uses the probability distributions as well as the language model to generate reasonable transcription result. The scores are frame by frame in order. Scores from the same time frame strictly follow the same order, which is the order of label list specified in the column map file.</p> <p>For example, if the second value of the array <code>inputs</code> is 0.1 and the system does predict more than one label, then the probability of the second label in the first frame is estimated to be 0.1 by the acoustic model. If an audio has fewer time frames than <code>nFrames</code>, then the scores of those non-existing time frames are padded as zeros.</p>

Table 4- 85 Output Mapping

Name	Type	Variable Type	Required or Optional?	Default Value	Description
transOut	variable	string	Required	No default value	Specifies the variable name of the transcription result.

The edge between the Score window and Calculate window is defined at the end of the project.

```
<edges>
<edge source='w_score' target='w_decode' role='data' />
</edges>
```

You can view the default values of the Transcription algorithm parameter properties for the Calculate window with the command-line utility.

### 4.9.26 Change Detection

Edge Streaming Analytics uses Kullback-Leibler divergence (KL divergence) to detect changes through histogram intersection. KL divergence measures how one probability distribution is different from a second, reference distribution over the same variable. It can be used in the fields of applied statistics and machine learning.

Consider the following example:



The continuous query includes the following:

- a Source window that receives input data
- a Calculate window that runs the change detection algorithm on that data

Here is the code for the Source window:

```
<window-source name="w_source" insert-only="true" index="pi_EMPTY">
<schema>
<fields>
<field type="int64" name="id" key="true" />
<field type="double" name="x" />
</fields>
</schema>
<connectors>
...
</connectors>
</window-source>
```

The following Calculate window applies the ChangeDetection algorithm to that data:

```
<window-calculate name="w_calculate" algorithm="ChangeDetection">
<schema>
```

```

<fields>
<field type="int64" name="id" key="true" />
<field type="double" name="x" />
<field type="double" name="changeVal" />
<field type="int32" name="eval" />
<field type="int32" name="changeDetected" />
</fields>
</schema>
<parameters>
<properties>
<property name="maxBins">100</property>
<property name="slidingAlpha">0.997</property>
<property name="refWindowSize">500</property>
<property name="maxEvalSteps">300</property>
<property name="adaptiveEval">1</property>
<property name="showEval">1</property>
<property name="showAll">0</property>
</properties>
</parameters>
<input-map>
<properties>
<property name="input">x</property>
</properties>
</input-map>
<output-map>
<properties>
<property name="evaluatedOut">eval</property>
<property name="changeValueOut">changeVal</property>
<property name="changeDetectedOut">changeDetected</property>
</properties>
</output-map>
</window-calculate>

```

The following parameters govern the change detection algorithm in the Calculate window:

Table 4- 86 Parameters

Name	Value Type	Required or Optional?	Default Value	Description
maxBins	int64	Optional	50	Specifies the maximum number of bins in the histogram for both the reference window and the sliding window.
slidingAlpha	double	Optional	1.0	Specifies the fading factor for the sliding window. Value range is $0 < \alpha \leq 1$ .
slidingHalfLifeSteps	int64	Optional	0	Specifies the number of steps at which the weight of the input reaches half of its original weight for the sliding window.

4.9 Using Online Models

Name	Value Type	Required or Optional?	Default Value	Description
refWindowSize	int64	Optional	1000	Specifies the size of the reference window.
changeThreshold	double	Optional	0.1	Specifies the threshold to determine whether a change occurred.
nComparisonBins	int64	Optional	100	Specifies the number of bins when computing KL-Divergence.
maxEvalSteps	int64	Optional	300	Specifies the maximum number of steps before performing a new evaluation.
adaptiveEval	int32	Optional	1	Specifies whether to use the adaptive evaluation step size or not.
showEval	int32	Optional	0	Specifies whether to show evaluation events regardless of whether a change is detected.
showAll	int32	Optional	0	Specifies whether to show all events, regardless of whether an evaluation occurs.

Table 4- 87 Input Mapping

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
input	variable	double	Required	No default value.	Specifies the input variable for change detection.

Table 4- 88 Output Mapping

Name	Value Type	Variable Type	Required or Optional?	Default Value	Description
evaluatedOut	varlist	int32	Optional	""	Specifies the name of the output variable that indicates whether an evaluation occurred.
changeValueOut	varlist	int32	Optional	""	Specifies the name of the output variable that contains the change value (the difference between two KL divergence values).
changeDetectedOut	varlist	int32	Optional	""	Specifies the name of the output variable that indicates whether a change has been detected.

The edge is defined at the end of the project.

```
<edges>
<edge role="data" target="w_calculate" source="w_data" />
</edges>
```

You can view the default values of the image processing algorithm parameter properties for the Calculate window with the command-line utility.

## 4.10 Using Recommender Systems

### 4.10.1 Overview

A recommender system attempts to predict the rating or preference that someone would give an item such as a book, article, or product based on previous ratings or other items. The recommender system shipped with Edge Streaming Analytics provides two approaches:

- Regularized Matrix Factorization (RMF), which projects objects into a lower dimensional latent space in order to discover latent features that underlie the interactions between two different types of entities
- K Nearest Neighbor (KNN) classification, which takes an input measure in a feature space and assigns a class based on the K-nearest neighbors in that space

**Note**

The current implementation of recommender scoring in Edge Streaming Analytics does not account for already rated items when it produces the top number of recommendations. That is, items already rated by a user might appear in the recommendations generated for that user. Recommendations generated by the SAS Visual Analytics Recommender System Action Set ignore those items. Thus, results differ across these products.

You can apply offline or online recommender models to streaming events.

**4.10.2 Offline Recommender Models**

**Generating Offline Recommender Models**

To generate offline recommender models, use the Recommender System action set provided by SAS Visual Analytics. Specifically, generate the following SAS Cloud Analytic System (CAS) tables:

Table 4- 89 SAS Cloud Analytic System (CAS) Tables for Recommender Systems

Recommender	Tables
RMF	<ul style="list-style-type: none"> <li>• item factor table (generated by <code>recommend.recomals</code> simultaneously with the user factor table)</li> <li>• user factor table (generated by <code>recommend.recomals</code> simultaneously with the item factor table)</li> <li>• item average rating table (generated by <code>recommend.recomrateinfo</code>)</li> <li>• user average rating table (generated by <code>recommend.recomrateinfo</code>)</li> </ul>
KNN	<ul style="list-style-type: none"> <li>• item average rating table (generated by <code>recommend.recomrateinfo</code>)</li> <li>• user average rating table (generated by <code>recommend.recomrateinfo</code>)</li> <li>• user similarity table (generated by <code>recommend.recomsim</code>)</li> <li>• ratings by users table (this is the training data that you create to generate the user similarity table)</li> </ul>

To use Edge Streaming Analytics to apply these models to streaming events, you must export each of those tables to CSV files.

Table 4- 90 SAS Cloud Analytic System (CAS) Tables for Recommender Systems

CAS Table	Required Format of CSV File	Model Parameter to Specify CSV File
item factor table	item_id,_F0_,..._FN_. Here, _FN_ is the Nth latent factor variable. The CSV file can also contain a leading ID column.	itemTable
user factor table	user_id,_F0_,..._FN_. Here, _FN_ is the Nth latent factor variable. The CSV file can also contain a leading ID column.	userTable
item average rating table	item_id,_Stat_,_NRatings_. The _Stat_ is the average rating.	itemRateInfo
user average rating table	user_id,_Stat_,_NRatings_. The _Stat_ is the average rating.	userRateInfo
user similarity table	USER_ID_1,USER_ID_2,_Sim_. The _Sim_ is the similarity rating.	similarUsers
ratings by user table	This CSV file must contain three fields that specify values for the following parameters: userid, itemid, rating. The fields must appear in that order.	ratingsByUser

For more information about the Recommender System action set, see SAS Visual Analytics: Programming Guide.

You can update the parameter values of offline models dynamically or statically.

### Dynamically Updating Parameter Values

The following continuous query dynamically updates the model parameter values of an offline model:

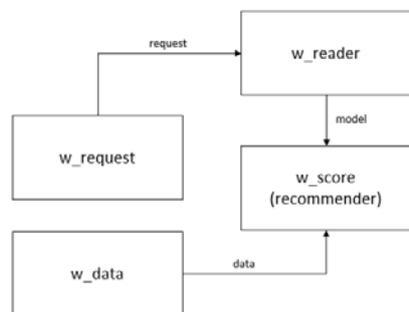


Figure 4-6 Offline Recommender Model with Dynamically Updated Parameter Values

- a Source window streams request events into a Model Reader window that specify parameter values for the offline model
- a Model Reader window streams model events that specify the model into a Score window
- a separate Source window streams data events that contain the data to be scored by the model into the Score window

For example:

```

<window-source name="w_data" pubsub="true" insert-only="true" index="pi_EMPTY">
<schema>
<fields>...
</fields>
</schema>
<connectors>...
</connectors>
</window-source>
<window-source name="w_request" pubsub="true" insert-only="true" index="pi_EMPTY">
<schema>
<fields>...
</fields>
</schema>
<connectors>...
</connectors>
</window-source>
<window-model-reader name="w_reader" pubsub="true" model-type="recommender" />
<window-score name="w_score" pubsub="true">
<schema>
<fields>...
</fields>
</schema>
<models>
<offline model-type="recommender">
<input-map>
<properties> ...
</properties>
</input-map>
<output-map>
<properties>...
</properties>
</output-map>
</offline>
</models>
<connectors>...
</connectors>
</window-score>
</windows>
<edges>
<edge role="data" source="w_data" target="w_score" />
<edge role="model" source="w_reader" target="w_score" />
<edge role="request" source="w_request" target="w_read" />
</edges>

```

### Statically Updating Parameter Values

The following continuous query statically updates model parameter values:

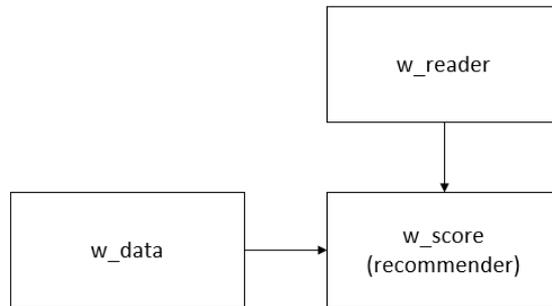


Figure 4-7 Offline Recommender Model with Statically Updated Parameter Values

Notice that here, the Model Reader window does not receive request events from a Source window. Model parameter values are coded within the Model Reader window itself. For example:

```

<window-source name="w_data" pubsub="true" insert-only="true" index="pi_EMPTY">

<schema>
<fields> ...
</fields>
</schema>
<connectors>...
</connectors>
</window-source>
<window-model-reader name="w_reader" pubsub="true" model-type="recommender">
<parameters>
<properties>
<property name="name">value</property>...
</properties>
</parameters>
</window-model-reader>
<window-score name="w_score" pubsub="true">
<schema>...
</schema>
<models>
<offline model-type="recommender">
<input-map>
<properties>...
</properties>
</input-map>
<output-map>
<properties>...
</properties>
</output-map>
</offline>
</models>
<connectors> ...
</connectors>
  
```

4.10 Using Recommender Systems

```

</window-score> ...
<edges>
<edge role="data" source="w_data" target="w_score" />
<edge role="model" source="w_read" target="w_score" />
</edges>
    
```

### 4.10.3 Specifying Model Parameters

The following model parameters are common to both the RMF and KNN recommender systems:

Table 4- 91 Common Model Parameters for RMF and KNN

Parameter	Type	Required or Optional?	Default Value	Description
method	string	Optional	<b>RMF</b>	Specifies the algorithm to be used for recommendation. Valid values are <b>RMF</b> or <b>KNN</b> .
userRateInfo	string	Required	No default value	Specifies the name of the user average rating CSV file. These can be used as bias terms. See Table 6.248 for details.
userRateInfoDelimiter	string	Optional	<b>"COMMA"</b>	Specifies the delimiter used in the user average rating CSV file. Valid values are <b>"COMMA"</b> , <b>"TAB"</b> , or <b>"SPACE"</b> .
userRateInfoLineBreak	string	Optional	<b>"LF"</b>	Specifies the line break character used in the user average rating CSV file. Valid values are <b>"LF"</b> , <b>"CF"</b> , or <b>"CRLF"</b> .
itemRateInfo	string	Required	No default value	Specifies the name of the item average rating CSV file. See Table 6.248 for details. This file is used with popularity-based cold start.

Parameter	Type	Required or Optional?	Default Value	Description
itemRateInfoDelimiter	string	Optional	"COMMA"	Specifies the delimiter used in the item average rating CSV file. Valid values are "COMMA", "TAB", or "SPACE".
itemRateInfoLineBreak	string	Optional	"LF"	Specifies the line break character used in the item average rating CSV file. Valid values are "LF", "CF", or "CRLF".

The following model parameters are specific to an RMF recommender model:

Table 4- 92 Model Parameters for the RMF Algorithm

Parameter	Type	Required or Optional?	Default Value	Description
itemTable	string	Required	""	Specifies the name of the item factor CSV file. It contains the low-rank features of each item.
itemTableDelimiter	string	Optional	"COMMA"	Specifies the delimiter used in the item factor CSV file. Valid values are "COMMA", "TAB", or "SPACE".
itemTableLineBreak	string	Optional	"LF"	Specifies the line break character used in the item factor CSV file. Valid values are "LF", "CF", or "CRLF".
userTable	string	Required	""	Specifies the name of the user factor CSV file.
userTableDelimiter	string	Optional	"COMMA"	Specifies the delimiter used in the user factor CSV file. Valid values are "COMMA", "TAB", or "SPACE".
userTableLineBreak	string	Optional	"LF"	Specifies the line break character used in the user factor CSV file. Valid values are "LF", "CF", or "CRLF".

The following model parameters are specific to a KNN recommender model:

Table 4- 93 Model Parameters for the KNN Algorithm

Parameter	Type	Required or Optional?	Default Value	Description
k	int64	Optional	20	Specifies the maximum number of nearest neighbors when calculating the rating of each user.
similarUsers	string	Required	""	Specifies the name of the user similarity CSV file. See Table 6.248 for details.
similarUsersDelimiter	string	Optional	"COMMA"	Delimiter used in the user similarity CSV file. Valid values are "COMMA", "TAB", or "SPACE".
similarUsersLineBreak	string	Optional	"LF"	Specifies the line break character used in the user similarity CSV file. Valid values are "LF", "CF", or "CRLF".
ratingsByUser	string	Required	""	Specifies the name of the user ratings CSV file.  This CSV file must contain three fields that specify values for the following parameters: <code>userid</code> , <code>itemid</code> , <code>rating</code> . The fields must appear in that order.
ratingsByUserDelimiter	string	Optional	"COMMA"	Delimiter used in the user ratings file. Valid values are "COMMA", "TAB", or "SPACE".
ratingsByUserLineBreak	string	Optional	"LF"	Specifies the line break character used in the user ratings file. Valid values are "LF", "CF", or "CRLF".

## 4.10.4 Online Recommender Models

The following continuous query applies an online recommender model that is being trained online.

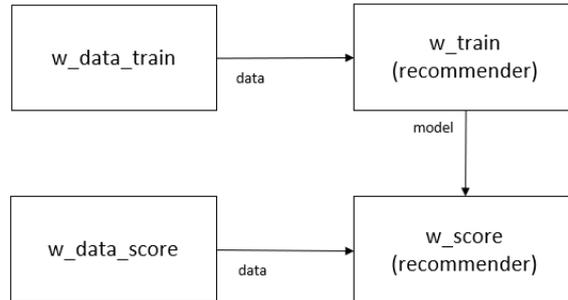


Figure 4-8 Scoring Data with a Recommender Model Being Trained

Here is an example of a Train window that uses an initial training data set:

```

<window-train name="w_train" pubsub="true" algorithm="recommender">
  <parameters>
  <properties>
  <property name="method">RMF</property>
  <property name="initMethod">BATCH_TRAINING</property>
  <property name="initModelFormat">CSV4</property>
  ...
  <property name="ratingsByUser">./input/ratings_by_user2.csv</property>
  <property name="ratingsByUserDelimiter">COMMA</property>
  <property name="ratingsByUserLineBreak">LF</property>
  <property name="ratingsByUserIndex">N</property>
  <property name="userTableFilename">./input/userFactorBook.csv</property>
  <property name="itemTableFilename">./input/itemFactorBook.csv</property>
  <property name="userBiasFilename">./input/userBiasBook.csv</property>
  <property name="itemBiasFilename">./input/itemBiasBook.csv</property>
  </properties>
  </parameters>
  <input-map>
  <properties>
  ...
  </properties>
  </input-map>
</window-train>
  
```

### 4.10.5 Training an Online Recommender Model

The following properties tune the recommender algorithm within the Train window:

Table 4- 94 Input Mapping

Name	Type	Required or Optional?	Default value	Description
user	string	Required	No default value	Specifies the incoming user ID.
item	string	Required	No default value	Specifies the incoming item ID.
rating	double	Required	No default value	Specifies the rating value.

You can reconfigure some of the parameter values in the Train window while data is streaming through the model. Create an edge between the Source window and the Train window with the role "request." Then, stream a reconfig request and events that change parameter values. For example:

```
i,n,1,"action","reconfig"
i,n,2,"maxIters","20"
i,n,3,"commitInterval","50"
i,n,4,"trainInterval","50"
i,n,5,"updateItemFactor","2"
i,n,6,"scale","1"
i,n,7,,
```

Table 4- 95 Input Mapping

Name	Type	Required or Optional?	Default value	Description	Parameter Value Can Be Reconfigured
maxUsers	int64	Optional	200000	Specifies the maximum number of users.	No
maxItems	int64	Optional	200000	Specifies the maximum number of items.	No
maxRatings	int64	Optional	100000	Specifies the maximum number of stored ratings per user and per item. Any incoming rating beyond this capacity causes the removal of the oldest rating.	No

Name	Type	Required or Optional?	Default value	Description	Parameter Value Can Be Reconfigured
nFactors	int32	Optional	3	Specifies the number of latent features per user and per item, including the user bias and item bias. There must be at least two, in which case the user and item bias terms are the only features that are used in the model.	No
regL2	double	Optional	1e-2	Specifies a value that improves matrix conditions in the training algorithm.	Yes
maxInitIters	int32	Optional	20	Specifies the number of iterations in initial training.	No
maxIters	int32	Optional	3	Specifies the number of iterations in each online training episode triggered by streaming training events.	Yes
initMethod	string	Optional	BATCH_TRAINING	Specifies the method used to initialize the model. Valid values are "BATCH_TRAINING", "MODEL_READING", or "ONLINE_TRAINING".	No
initModelFormat		Optional	CSV4	When initMethod="MODEL_READING", specifies the format of the pre-trained model used to initialize the Train window. Valid values are "CSV4" or "CSV2".	No

4.10 Using Recommender Systems

Name	Type	Required or Optional?	Default value	Description	Parameter Value Can Be Reconfigured
nInit	int64	Optional	500	Specifies the number of training events required to initiate online training (that is, start the first episode of model update on streaming data).	No
commitInterval	int64	Optional	100	Specifies the number of training events required between two consecutive model commitments to score window.	Yes
trainInterval	int64	Optional	100	Specifies the number of training events between two consecutive online training episodes.	Yes
updateItemFactor	int32	Optional	1	Specifies whether to update item factors: A value of 0 indicates not to update item factors. A nonzero value indicates to update items factors.	Yes
hasUserBias	int32	Optional	1	Specifies whether the model contains user bias terms: A value of 0 indicates no user bias terms. A nonzero value indicates user bias terms.	Yes
hasItemBias	int32	Optional	1	Specifies whether the model contains item bias terms: A value of 0 indicates no item bias terms. A nonzero value indicates item bias terms	Yes

Name	Type	Required or Optional?	Default value	Description	Parameter Value Can Be Reconfigured
hasCommonBias	int32	Optional	1	Specifies whether the model contains common bias terms: A value of zero indicates no common bias terms. A nonzero value indicates common bias terms.	Yes
scale	int32	Optional	0	Specifies how to rescale ratings to [0, 10]. A value of 0 indicates no scaling. A nonzero value indicates scaling.	Yes
userTableName	string	Optional	""	Specifies the CSV file to which the user factor matrix is saved.	No
itemTableName	string	Optional	""	Specifies the CSV file to which the item factor matrix is saved.	No
userBiasName	string	Optional	""	Specifies the CSV file to which the user bias terms are saved.	No
itemBiasName	string	Optional	""	Specifies the CSV file to which the item bias terms are saved.	No

When `initMethod="BATCH_TRAINING"`, the RMF recommender model is initialized with a single CSV file that contains an initial training set. Use the value of `ratingsByUser` to specify the

4.10 Using Recommender Systems

name of the training set. Specify values for the remaining parameters within the Train window.

Table 4- 96 Training Parameters When `initMethod="BATCH_TRAINING"`

Name	Type	Required or Optional?	Default value	Description
<code>ratingsByUser</code>	string	Optional	""	Name of the CSV file that contains the initial training data.
<code>ratingsByUserDelimiter</code>	string	Optional	"COMMA"	Delimiter used in the initial training data file. Valid values are "COMMA", "TAB", or "SPACE".
<code>ratingsByUserLineBreak</code>	double	Optional	"LF"	Line break character used in the initial training data file. Valid values are "LF", "CR", or "CRLF".

The RMF recommender model is imported from a specified source when `initMethod="MODEL_READING"`. The model is defined by four CSV files when `initModelFormat="CSV4"` or two CSV files when `initModelFormat="CSV2"`.

Model parameters when `initModelFormat="CSV4"` are as follows:

Table 4- 97 Training Parameters When `initMethod="MODEL_READING"` and `initModelFormat="CSV4"`

Parameter	Type	Required or Optional?	Default Value	Description
<code>itemTable</code>	string	Required	""	Name of the CSV file that contains the item model without biases.
<code>itemTableIndex</code>	string	Optional	"Y"	Specifies whether the item model file contains row indexes.
<code>userTable</code>	string	Required	""	Specifies the name of the CSV file that contains the user model without biases.
<code>userTableIndex</code>	string	Optional	"Y"	Specifies whether the user model file contains row indexes.
<code>userRateInfo</code>	string	Required	""	Specifies the name of the CSV file that contains the user biases.
<code>userRateInfoIndex</code>	string	Optional	"N"	Specifies whether the user biases file contains row indexes.
<code>itemRateInfo</code>	string	Required	""	Name of the CSV file that contains the item biases.

Parameter	Type	Required or Optional?	Default Value	Description
itemRateInfoIndex	string	Optional	"N"	Specifies whether the item biases file contains row indexes.
csvDelimiter	string	Optional	"COMMA"	Delimiter used in the CSV files. Valid values are "COMMA", "TAB", or "SPACE".
csvLineBreak	string	Optional	"LF"	Line break character used in CSV files. Valid values are "LF", "CR", or "CRLF".

Model parameters when `initModelFormat="CSV2"` are as follows:

Table 4- 98 Training Parameters When `initMethod="MODEL_READING"` and `initModelFormat="CSV2"`

Parameter	Type	Required or Optional?	Default Value	Description
itemTableWithBias	string	Required	""	Name of the CSV file that contains the item model with biases.
itemTableWithBiasIndex	string	Optional	"N"	Specifies whether the item model file has row indexes.
userTableWithBias	string	Optional	""	Name of the CSV file that contains the user model with biases.
userTableWithBiasIndex	string	Optional	"N"	Specifies whether the user model file has row indexes.
csvDelimiter	string	Optional	"COMMA"	Delimiter used in the CSV files. Valid values are "COMMA", "TAB", or "SPACE".
csvLineBreak	string	Optional	"LF"	Line break character used in the CSV files. Valid values are "LF", "CR", or "CRLF".

The model is initialized with streaming data when `initMethod="ONLINE_TRAINING"`.

Here is a Train window that uses online training. Notice that the only reference to external CSV files is the value of `initModelFormat`. You can use an offline model to initialize a model that you trained online.

```
<window-train name='w_train' algorithm='recommender'>
  <parameters>
    <properties>
      <property name="nFactors">3</property>
      <property name="method">RMF</property>
      <property name="initMethod">ONLINE_TRAINING</property>
      <property name="initModelFormat">CSV4</property> <property
name="maxUsers">200000</property>
```

4.10 Using Recommender Systems

```

<property name="maxItems">200000</property>
<property name="maxInitIters">20</property>
<property name="maxIters">10</property>
<property name="commitInterval">1000</property>
<property name="trainInterval">500</property>
<property name="nInit">1000</property>
<property name="updateItemFactor">1</property>
<property name="hasCommonBias">0</property>
<property name="hasUserBias">1</property>
<property name="hasItemBias">1</property>
<property name="scale">0</property>
<property name="maxRatings">100000</property>
<property name="regL2">1e-2</property>
</properties>
</parameters>
<input-map>
<properties>
...
</properties>
</input-map>
</window-train>
    
```

4.10.6 Scoring Data with Recommender Models

Score windows use the following properties for both KNN and RMF recommender models

Table 4- 99 Parameters

Name	Type	Required or Optional?	Default Value	Description
nTopRecoms	int32	Optional	50	The number of top recommendations to make to each user.

Table 4- 100 Input Mapping

Name	Type	Variable Type	Required or Optional?	Default Value	Description
user	variable	string	Required	No default value	Specifies the targeted user ID to make recommendations <b>Note:</b> Specify this as a key field.

Table 4- 101 Output Mapping

Name	Type	Variable Type	Required or Optional?	Default Value	Description
itemOut	variable	string	Required	No default value	Specifies the IDs of each recommended item <b>Note:</b> Specify this as a key field.
ratingOut	variable	double	Required	No default value	Specifies the predicted rating for each recommended item
rankOut	variable	double	Required	No default value	Specifies the relative ranking values for each recommended item. The highest ranking is specified as 1, the second-highest as 2, and so on.



# Accessibility Features

## 5.1 Accessibility Features of Edge Streaming Analytics

### 5.1.1 Overview

Edge Streaming Analytics has not been tested against U.S. Section 508 standards and W3C Web Content Accessibility Guidelines (WCAG).

We recommend the following software for a better experience using our products with assistive technologies:

- On Microsoft Windows, use the latest version of JAWS.

### 5.1.2 User Interface Layout

The primary user interface to Edge Streaming Analytics for building streaming application models is Edge Streaming Creator. It enables you to draw an event stream processing model as a data flow diagram and to control how windows within a model relate and flow into one another.

In Edge Streaming Creator, toolbars and pages are displayed as primary navigation tools, and different options are available depending on the active page.

### 5.1.3 Keyboard Shortcuts

In the user interface, the following keyboard shortcuts are available:

Table 5- 1 Keyboard Shortcuts

Editor	Action	Keyboard Shortcut
Modeler	Remove a selected window from the model diagram	Delete key
XML Editor	Revert your previous change	Ctrl + Z
XML Editor	Revert the effects of the undo action	Ctrl + Y
XML Editor	Remove the selected code from its original position	Ctrl + X
XML Editor	Copy the selected code to the clipboard	Ctrl + C

*5.1 Accessibility Features of Edge Streaming Analytics*

<b>Editor</b>	<b>Action</b>	<b>Keyboard Shortcut</b>
XML Editor	Paste the code from the clipboard at the cursor's position	Ctrl + V
XML Editor	Search for specific text	Ctrl + F

