



PUBLIC

SAP Data Intelligence

Document Version: 3.1.7 – 2021-10-15

Repository Objects Reference

Content

1	Repository Objects Reference for SAP Data Intelligence.	11
2	SAP Data Intelligence Operators.	12
2.1	ABAP.	29
	ABAP Converter.	32
	ABAP CDS Reader.	33
	SLT Connector.	38
	Cluster Table Splitter.	43
	Custom ABAP Operator.	45
	ABAP ODP Reader.	46
	SAP ABAP Operator.	50
2.2	Connectivity.	51
	AWS SNS Consumer.	52
	AWS SNS Producer.	55
	Google Pub/Sub Consumer.	58
	Google Pub/Sub Producer.	62
	HANA Client 2.	65
	HANA Monitor.	80
	HTTP Client 2.	82
	HTTP Server.	86
	Kafka Consumer 2.	90
	Kafka Producer.	96
	MQTT Consumer.	102
	MQTT Producer.	104
	NATS Consumer.	107
	NATS Producer.	108
	OpenAPI Client.	109
	OpenAPI Server.	116
	Receive Email.	121
	Send Email.	124
	WAMP Consumer.	129
	WAMP Producer.	130
2.3	Connectivity (via Flowagent).	131
	Azure SQL Database SQL Consumer.	132
	Azure SQL Database Table Consumer.	134
	DB2 SQL Consumer.	136
	DB2 Table Consumer.	138

	Flowagent CSV Producer.	140
	Flowagent File Producer.	143
	Flowagent SQL Executor.	147
	Flowagent Table Producer.	150
	Google BigQuery Producer.	154
	Google BigQuery SQL Consumer.	156
	HANA Table Consumer.	157
	MySQL SQL Consumer.	158
	MySQL Table Consumer.	160
	OData Query Consumer.	162
	ODBC SQL Consumer.	163
	ODBC Table Consumer.	166
	Open Connectors SQL Consumer.	169
	Open Connectors Table Consumer.	171
	Oracle Table Consumer.	173
	Oracle SQL Consumer.	175
	SAP IQ Table Consumer.	177
	SAP IQ SQL Consumer.	179
	SQL Server SQL Consumer.	181
	SQL Server Table Consumer.	182
	Table Replicator.	184
2.4	Converter.	194
	StreamToString Converter.	195
	StringToStream Converter.	196
	ToBlob Converter.	196
	ToMessage Converter.	197
	ToNumber Converter.	198
	ToString Converter.	199
	ToStringType Converter.	200
2.5	Data Quality.	202
	Anonymization.	202
	Data Mask.	215
	DQMm Address Cleanse.	236
	DQMm Client.	245
	DQMm Reverse Geo.	248
	Person and Firm Cleanse.	251
	Validation Rule.	259
	Data Quality SQL Reference.	264
2.6	Data Workflows.	280
	BW Process Chain.	280
	Data Transfer.	282

	HANA Flowgraph.	284
	Notification.	285
	Pipeline.	287
	SAP Data Services Job.	289
	Workflow Merge (and).	291
	Workflow Merge (or).	292
	Workflow Split.	293
	Workflow Terminator.	294
	Workflow Trigger.	295
2.7	Files.	295
	From File.	296
	List Files.	296
	Monitor Files.	300
	Read File.	303
	Remove File.	306
	To File.	308
	Write File.	309
2.8	Hadoop and Spark.	318
	Livy Spark Submit.	318
	Submit Hadoop Job.	322
2.9	Multiplexer.	327
2.10	Processing.	328
	Command Executor.	329
	Constant Generator.	330
	Data Generator.	331
	Go Message Operator.	332
	Go Operator.	333
	Go String Operator.	339
	JavaScript Operator.	340
	JavaScript (v2).	346
	JS Message Operator.	352
	JS String Operator.	353
	Message Generator.	354
	Process Executor.	355
	Python 3 Operator.	356
	R Client 3.	364
	Stdin/out Port Muxer.	376
	Streaming Analytics Projects.	377
2.11	SAP HANA.	388
	Initialize HANA Table.	388
	Read HANA Table.	392

	Run HANA SQL.	397
	Write to HANA Table.	399
2.12	SAP Integration.	401
	Cloud Data Integration Consumer.	402
	SAP Analytics Cloud Formatter.	403
	SAP Analytics Cloud Producer.	406
	SAP CP EM Consumer.	408
	SAP CP EM Producer.	411
	SAP CPI-PI iFlow.	414
2.13	SAP Machine Learning Core Operators.	417
	Artifact Consumer.	419
	Artifact Producer.	420
	HANA ML Forecast Operator.	424
	HANA ML Inference Operator.	432
	HANA ML Training Operator.	437
	Model Serving Operator.	444
	Submit Metrics.	453
	ML Training.	454
2.14	SAP Vora.	456
	Vora Avro Decoder.	457
	Vora Avro Encoder.	461
	Vora Avro Ingestor.	462
	Vora Client.	470
	Vora Ingestor.	473
	Vora Loader.	477
	Vora Table Consumer.	481
2.15	Streaming Analytics.	483
	Streaming Analytics Projects.	483
2.16	Structured Data Operators.	494
	Data Transform.	495
	SAP Application Consumer.	496
	SAP Application Producer.	497
	SQL Consumer.	498
	Structured File Consumer.	501
	Structured File Producer.	503
	Table Consumer.	507
	Table Producer.	511
2.17	Tables.	512
	Decode Table.	512
	Generate Table.	516
2.18	Utilities.	518

	Avro Serializer.	519
	Avro Deserializer.	520
	Flowgraph Compiler.	521
	Format Converter.	522
	Graph Terminator.	525
	Header Remover.	526
	Histogram.	527
	Histogram Plotter.	529
	HTML Viewer.	529
	Message Aggregator.	530
	Message Key Store.	533
	Py3 Data Generator.	537
	Py3 Type to Blob.	537
	Py3 Type to String.	538
	Terminal.	539
	Wiretap.	541
2.19	Others.	543
2.20	Deprecated Operators.	543
	CDC Graph Generator (Deprecated).	543
	Copy File (Deprecated).	548
	Create TA Request (Deprecated).	553
	Functional Services (Deprecated).	556
	Move File (Deprecated).	574
	PA Automated (Deprecated).	579
	Read File (Deprecated).	580
	Remove File (Deprecated).	590
	SAP ABAP ODP Object Consumer (Deprecated).	594
	Spark Data Transform (Deprecated).	598
	Text Analysis Connector (Deprecated).	599
	Write File (Deprecated).	603
3	SAP Data Intelligence Graphs.	609
3.1	Connectivity.	623
	Example for OpenAPI Greeter Demo (Go).	623
	Example for OpenAPI Plain Greeter Demo (Go).	623
	Kafka.	624
	OpenAPI GreeterClient Demo.	624
	OpenAPI PlainClient Demo.	625
	OpenAPI Greeter Demo.	625
	OpenAPI Plain Greeter.	626
	S3.	626
	Google Pub/Sub.	627

3.2	Examples.	628
	Anonymization.	629
	Basic Validation.	630
	Data Generator.	631
	Data Mask.	632
	Data Workflow.	632
	DQMm Address.	633
	DQMm Reverse.	634
	Example Counter.	634
	Example Notification.	635
	File System.	636
	Functional Services.	636
	GCS.	637
	Go Data Generator.	638
	Go Message Generator.	638
	Go Operator and File.	639
	Google Dataproc.	640
	HDFS.	640
	HTML Viewer.	641
	Histogram Simple Example.	641
	IoT Validation.	642
	Kafka.	643
	Load into SAP HANA.	644
	Load into SAP Vora.	644
	Message Generator.	645
	Person and Firm.	646
	RClient Example.	646
	Simple Wiring.	649
	Storage.	649
	S3.	650
	Vora SQL Console.	651
3.3	Graph Snippets.	651
	Copy Files (Full Load).	652
	Copy Files (Incremental Load).	652
	Load Files into HANA (Full Load).	653
	Load Files into VORA (Incremental Load).	653
	Simple Javascript DB-to-File Data Manipulation.	654
	Simple Javascript File Data Manipulation.	655
	Simple Javascript File-to-DB Data Manipulation.	655
	Simple Node.js File Data Manipulation.	656
	Simple Python File Data Manipulation.	657

	Simple Python File-to-DB Data Manipulation.	657
	Simple R File Data Manipulation.	658
	Simple R File-to-DB Data Manipulation.	658
3.4	Ingestion via Flowagent.	659
	CDC Graph Generator Sample Graph.	659
	CSV Producer.	660
	Flowagent Consumer.	661
	Ingest OData Query	662
	Ingest to Table via Flowagent.	663
	Ingest Google BigQuery SQL.	663
	Ingest Google BigQuery Table Producer.	664
	MySQL Consumer.	664
	Open Connectors SQL Consumer.	670
	Open Connectors Table Consumer.	671
	Table Consumer via Flowagent.	672
3.5	ML Scenario Manager Templates and Examples.	677
	Auto ML Inference Template.	678
	Auto ML Training Template.	678
	Basic Training (TensorFlow Training Template).	678
	HANA ML Inference on Dataset.	680
	HANA ML Inference on API Requests.	680
	HANA ML Training Template.	681
	HANA ML Forecast Template.	682
	ML Batch Inference.	683
	ML Multi-Model Inference.	684
	Python Consumer Template.	687
	Python Data Preprocessing Example.	689
	Python Producer Template.	691
	Pytorch Text Classification.	693
	R Consumer - Consuming an ML Model via API using R Template.	694
	R Consumer - Exposing a trained ML model via API Example.	696
	R Producer - Training an ML Model Using R Template.	697
	R Producer - Training a classification model using R Example.	698
	R Producer - Training a regression model using R.	700
	TensorFlow MNIST Training Example.	701
	Tensorflow Serving Template.	702
	TensorFlow 2 and Scikit.	702
3.6	Others.	703
	Example for Vora ingestion using avro data from kafka and Go operator.	703
3.7	SAP Integration.	704
	SAP Analytics Cloud Push Demo.	704

	SAP CP EM Consumption Example.	706
	SAP CP EM Production Example.	706
	SAP CPI-PI iFlow Example.	707
3.8	SAP Vora.	708
	Direct Avro Ingestion.	709
	Avro Ingest via Disk.	709
	Avro Ingest Mod via Disk.	710
	CSV Ingest2 via Disk.	711
	CSV Ingest2 via Disk.	712
	CSV Ingest3 via Disk.	712
	Direct CSV Ingest2.	713
	JSON Ingest2 via Disk.	714
	Record Ingest via Disk.	715
	Vora Table Consumer.	716
	Example for Vora Ingestion Using CSV Data 2.	717
	Example for Vora Ingestion Using CSV Data 3.	717
	Example for Vora Ingestion Using CSV Data 5.	718
	Example for Vora Ingestion Using CSV Data 6.	719
	Example for Vora Ingestion Using JSON Data 5.	720
	Example for Vora Ingestion Using JSON Data 6.	720
	Example for Vora Ingestion Using Record Object Data 2 in Go.	721
	Example for Vora Ingestion Using Record Object Data 2 in Python.	722
3.9	Scenario Templates.	722
	BW ODP to Vora (Initial + Delta Load).	723
	BW HANA View to File.	724
	Data Extraction from SAP S/4HANA CDS View to a File Store.	725
	Data Extraction from SAP S4/HANA CDS View to KAFKA.	726
	Data Extraction using SLT to a File Store.	728
	Data Extraction using SLT to KAFKA.	729
	Ingest Files into HANA.	730
	Ingest Files into Vora.	731
	Initial Load + Delta Extraction from AnyDB.	733
	Initial Load from Oracle (parallel).	733
	Javascript File Data Manipulation.	734
	Load Files into HANA.	735
	Load Files into Vora.	736
	Node.js File Data Manipulation.	737
	Python File Data Manipulation with Pandas.	738
	R File Data Manipulation.	739
	Simple Javascript File Data Manipulation.	740
	Simple Javascript File-to-DB Data Manipulation.	741

	Simple Javascript DB-to-File Data Manipulation.	742
	Simple Node.js File Data Manipulation.	743
	Simple Python File Data Manipulation.	744
	Simple Python File-to-DB Data Manipulation.	745
	Simple R File Data Manipulation.	746
	Simple R File-to-DB Data Manipulation.	747
3.10	Streaming Analytics.	747
	Using Adapters in Streaming Analytics Project CCL.	748
	Streaming Analytics Freezer Monitor Demo.	749
	Connecting to Kafka in Streaming Analytics Project CCL.	750
	Sending Datatypes to Streaming Analytics Projects.	751
3.11	Utilities.	752
	Go Plugin Generator/Bootstrapper.	752
	Wiretap.	753
	Wiretap 2.	754
	Wiretap 3.	754

1 Repository Objects Reference for SAP Data Intelligence

The Repository Objects Reference contains information about the built-in operators and graphs that SAP Data Intelligence provides for use in SAP Data Intelligence Modeler.

Related Information

[SAP Data Intelligence Operators \[page 12\]](#)

[SAP Data Intelligence Graphs \[page 609\]](#)

2 SAP Data Intelligence Operators

SAP Data Intelligence provides built-in operators, that can be directly used in a graph or used as the basis for a custom operator.

An event from the environment is a message delivered to the operator through its input ports. The operator can interact with the environment through its output ports. The operators are unaware of the graph in which it is defined and the source and target of its incoming and outgoing connections.

For information about using SAP Data Intelligence operators and creating your own custom operators, see [Operators](#) in the *Modeling Guide for SAP Data Intelligence*.

Note

- Some of the SAP Data Intelligence operators have configuration type property. It is recommended to choose [Configuration Manager](#) and select an existing connection id or define a new connection in the Connection Management application. The [Manual](#) option must be used only for testing purposes.

Category	Operator	ID	Primary Use Case	Status	Replaced By
ABAP [page 29]	ABAP Converter [page 32]	com.sap.abap.toS-tringConverter	Convert a generic ABAP data type to string using the CSV, JSON, or XML format.	Active	
ABAP [page 29]	ABAP CDS Reader [page 33]	com.sap.abap.cds_reader	Read data from a Core Data Services (CDS) view.	Active	
ABAP [page 29]	SLT Connector [page 38]	com.sap.abap.slt_reader	Establish connections between SAP Landscape Transformation Replication Server (SLT) and SAP Data Intelligence.	Active	
ABAP [page 29]	Cluster Table Splitter [page 43]	com.sap.abap.cluster_splitter	Use SLT Connector to replicate one or more (but not all) logical tables for a cluster table.	Active	
ABAP [page 29]	Custom ABAP Operator [page 45]	com.sap.abap.custom	Adapt to the selected custom operator implementation in the remote ABAP system.	Active	

Category	Operator	ID	Primary Use Case	Status	Replaced By
ABAP [page 29]	ABAP ODP Reader [page 46]	com.sap.abap.odp.read	Read data from an object for operational data processing (ODP).	Active	
ABAP [page 29]	SAP ABAP Operator [page 50]	com.sap.abap.sap	Adapt to the selected standard operator implementation in the remote ABAP system.	Active	
Deprecated Operators [page 543]	Copy File (Deprecated) [page 548]	com.sap.storage.copy2	Copy files in a storage service.	Deprecated	Please check SAP Note 2739161  for more information.
Connectivity [page 51]	AWS SNS Consumer [page 52]	com.sap.cloud.aws.sns.consumer	Consume publications from an Amazon Web Services Simple Notification Service (AWS SNS) topic and output them as a stream of messages.	Active	
Connectivity [page 51]	AWS SNS Producer [page 55]	com.sap.cloud.aws.sns.producer	Receive a message from the input port and publish it to an AWS SNS topic.	Active	
Connectivity [page 51]	Google Pub/Sub Consumer [page 58]	com.sap.cloud.gcp.pubsub.consumer	Consume publications from a Google Pub/Sub topic and output them as a stream of messages.	Active	
Connectivity [page 51]	Google Pub/Sub Producer [page 62]	com.sap.cloud.gcp.pubsub.producer	Receive a message from the input port and publish it to a Google Pub/Sub topic.	Active	
Connectivity [page 51]	HANA Client 2 [page 65]	com.sap.hana.client2	Execute SQL statements and inserts CSV or JSON data into an SAP HANA instance.	Active	

Category	Operator	ID	Primary Use Case	Status	Replaced By
Connectivity [page 51]	HANA Monitor [page 80]	com.sap.hana.monitor2	Create a trigger on a given table. For each row inserted, save a copy in a temporary table and poll it for those new rows.	Active	
Connectivity [page 51]	HTTP Client 2 [page 82]	com.sap.http.client2	Send arbitrary HTTP requests, polling a URL, and posting JSON data.	Active	
Connectivity [page 51]	HTTP Server [page 86]	com.sap.http.server2	Start an HTTP server on a configurable port. It handles GET and POST requests received from a configurable handler path.	Active	
Connectivity [page 51]	Kafka Consumer 2 [page 90]	com.sap.kafka.consumer2	Consume records and messages from a Kafka cluster.	Active	
Connectivity [page 51]	Kafka Producer [page 96]	com.sap.kafka.producer	Send records and messages to topics on a Kafka cluster.	Active	
Connectivity [page 51]	MQTT Consumer [page 102]	com.sap.mqtt.consumer	Subscribe to MQTT topics, consume its messages, and output them.	Active	
Connectivity [page 51]	MQTT Producer [page 104]	com.sap.mqtt.producer	Receive MQTT messages from a port and publish them in a topic.	Active	
Connectivity [page 51]	NATS Consumer [page 107]	com.sap.nats.consumer	Consume messages from NATS and forward them to the output port.	Active	
Connectivity [page 51]	NATS Producer [page 108]	com.sap.nats.producer	Receive a message from the input port and publish it into NATS.	Active	
Connectivity [page 51]	OpenAPI Client [page 109]	com.sap.openapi.client	Invoke services described in a swagger/openAPI document.	Active	

Category	Operator	ID	Primary Use Case	Status	Replaced By
Connectivity [page 51]	OpenAPI Server [page 116]	com.sap.open-api.server	Provide services described in swagger/openAPI or unspecified HTTP services in Modeler.	Active	
Connectivity [page 51]	Receive Email [page 121]	com.sap.email.receive	Receive emails.	Active	
Connectivity [page 51]	Send Email [page 124]	com.sap.email.send	Send emails.	Active	
Connectivity [page 51]		com.sap.social.tweetStream2	Receive a stream of tweets filtered by some keywords specified in the configuration using the Twitter Streaming API.	Active	
Connectivity [page 51]	WAMP Consumer [page 129]	com.sap.wamp.consumer	Access records from Web Application Messaging Protocol (WAMP).	Active	
Connectivity [page 51]	WAMP Producer [page 130]	com.sap.wamp.producer	Write files to Web Application Messaging Protocol (WAMP).	Active	
Deprecated Operators [page 543]	Move File (Deprecated) [page 574]	com.sap.storage.move2	Move (rename) files in a file service.	Deprecated	Please check SAP Note 2739161  for more information.
Deprecated Operators [page 543]	Read File (Deprecated) [page 580]	com.sap.storage.read	Read a file or periodically polls a directory for its contents in a storage service.	Deprecated	Please check SAP Note 2739161  for more information.
Deprecated Operators [page 543]	Remove File (Deprecated) [page 590]	com.sap.storage.remove2	Remove files in a storage service.	Deprecated	Please check SAP Note 2739161  for more information.
Deprecated Operators [page 543]	Write File (Deprecated) [page 603]	com.sap.storage.write	Write files to a storage service.	Deprecated	Write File [page 309] com.sap.file.write
Connectivity (via Flowagent) [page 131]	Azure SQL Database SQL Consumer [page 132]	com.sap.dh.ds.azure.sql.consumer	Perform a SQL query in an Azure SQL Database.	Active	
Connectivity (via Flowagent) [page 131]	Azure SQL Database Table Consumer [page 134]	com.sap.dh.ds.azure.sql.table.consumer	Read from an Azure SQL Database table or view.	Active	

Category	Operator	ID	Primary Use Case	Status	Replaced By
Connectivity (via Flowagent) [page 131]	DB2 SQL Consumer [page 136]	com.sap.dh.ds.db2.sql.consumer	Perform a SQL query in a DB2 database.	Active	
Connectivity (via Flowagent) [page 131]	DB2 Table Consumer [page 138]	com.sap.dh.ds.db2.table.consumer	Read from a DB2 table or view.	Active	
Connectivity (via Flowagent) [page 131]	Flowagent CSV Producer [page 140]	com.sap.dh.ds.csv.producer	Read any Flowagent-based consumer operator and output in a CSV format conforming to the RFC-4180 specification.	Active	
Connectivity (via Flowagent) [page 131]	Flowagent File Producer [page 143]	com.sap.dh.ds.storage.producer	Consume Flowagent-based consumer operators	Active	
Connectivity (via Flowagent) [page 131]	Flowagent SQL Executor [page 147]	com.sap.dh.ds.sql.executor	Perform an SQL Statement to any supported database.	Active	
Connectivity (via Flowagent) [page 131]	Flowagent Table Producer [page 150]	com.sap.dh.ds.table.producer	Read from any Flowagent-based consumer operator and write in a table of any supported database.	Active	
Connectivity (via Flowagent) [page 131]	Google BigQuery Producer [page 154]	com.sap.dh.ds.gbq.producer	Write to a Google BigQuery table.	Active	
Connectivity (via Flowagent) [page 131]	Google BigQuery SQL Consumer [page 156]	com.sap.dh.ds.gbq.sql.consumer	Perform a SQL query to a Google BigQuery table.	Active	
Connectivity (via Flowagent) [page 131]	HANA Table Consumer [page 157]	com.sap.dh.ds.hanaodbc.table.consumer	Read from a HANA table or view using a Flowagent subengine.	Active	
Connectivity (via Flowagent) [page 131]	MySQL SQL Consumer [page 158]	com.sap.dh.ds.mysql.sql.consumer	Perform a SQL query in a MySQL database.	Active	
Connectivity (via Flowagent) [page 131]	MySQL Table Consumer [page 160]	com.sap.dh.ds.mysql.table.consumer	Read from a MySQL table or view.	Active	
Connectivity (via Flowagent) [page 131]	OData Query Consumer [page 162]	com.sap.dh.sdi.odata.table.consumer	Read from an OData RESTful API.	Active	

Category	Operator	ID	Primary Use Case	Status	Replaced By
Connectivity (via Flowagent) [page 131]	ODBC SQL Consumer [page 163]	com.sap.dh.ds.odbc.sql.consumer	Perform a SQL query to any database that provides an Open Database Connectivity (ODBC) connector.	Active	
Connectivity (via Flowagent) [page 131]	ODBC Table Consumer [page 166]	com.sap.dh.ds.odbc.table.consumer	Read a table from any database that provides an Open Database Connectivity (ODBC) connector.	Active	
Connectivity (via Flowagent) [page 131]	Open Connectors SQL Consumer [page 169]	com.sap.dh.sdi.openconnectors.sql.consumer	Get and filter data by the given SQL statement from SAP Cloud Platform Open Connectors.	Active	
Connectivity (via Flowagent) [page 131]	Open Connectors Table Consumer [page 171]	com.sap.dh.sdi.openconnectors.table.consumer	Get all data from the given table name from SAP Cloud Platform Open Connectors.	Active	
Connectivity (via Flowagent) [page 131]	Oracle Table Consumer [page 173]	com.sap.dh.ds.oracle.table.consumer	Read from an Oracle table or view.	Active	
Connectivity (via Flowagent) [page 131]	Oracle SQL Consumer [page 175]	com.sap.dh.ds.oracle.sql.consumer	Perform a SQL query in an Oracle database.	Active	
Deprecated Operators [page 543]	CDC Graph Generator (Deprecated) [page 543]	com.sap.dh.cdc.graphGenerator	Generate three graphics by doing static analyses of the metadata of the source table.	Deprecated	Table Replicator [page 184] com.sap.database.table.replicator
Connectivity (via Flowagent) [page 131]	SAP IQ Table Consumer [page 177]	com.sap.dh.ds.sapiq.table.consumer	Read from an SAP IQ table or view.	Active	
Connectivity (via Flowagent) [page 131]	SAP IQ SQL Consumer [page 179]	com.sap.dh.ds.sapiq.sql.consumer	Perform an SQL query to an SAP IQ database.	Active	
Connectivity (via Flowagent) [page 131]	SQL Server SQL Consumer [page 181]	com.sap.dh.ds.microsoftsqlserver.sql.consumer	Perform an SQL query to an SQL Server database.	Active	

Category	Operator	ID	Primary Use Case	Status	Replaced By
Connectivity (via Flowagent) [page 131]	SQL Server Table Consumer [page 182]	com.sap.dh.ds.microsoftsql-server.table.consumer	Read from an SQL Server table or view.	Active	
Connectivity (via Flowagent) [page 131]	Table Replicator [page 184]	com.sap.data-base.table.replicator	Capture the delta and handle applying the changes to a target table, allowing replication of tables in an efficient and secure way, through an in-built recovery mechanism.	Active	
Converter [page 194]	StreamToString Converter [page 195]	com.sap.util.streamToString-Converter	Parse a contiguous stream and output strings that are separated by platform-dependent line delimiters.	Active	
Converter [page 194]	StringToStream Converter [page 196]	com.sap.util.stringToStream-Converter	Convert incoming messages of type 'string' to a contiguous stream.	Active	
Converter [page 194]	ToBlob Converter [page 196]	com.sap.util.toBlobConverter	Convert the input into a byte array.	Active	
Converter [page 194]	ToMessage Converter [page 197]	com.sap.util.toMessageConverter	Convert the input to a Modeler message.	Active	
Converter [page 194]	ToNumber Converter [page 198]	com.sap.util.toNumberConverter	Convert types int64, uint64, float64, and string into other numeric types.	Active	
Converter [page 194]	ToString Converter [page 199]	com.sap.util.toStringConverter	Convert the input to a string.	Active	
Converter [page 194]	ToStringType Converter [page 200]	com.sap.vgui.conversion.toString-Type	Convert regular types to the specialized string types used for Scenes.	Active	
Data Quality [page 202]	Anonymization [page 202]	com.sap.demo.anonymization	Protect the privacy of individuals by creating groups of similar records.	Active	

Category	Operator	ID	Primary Use Case	Status	Replaced By
Data Quality [page 202]	Data Mask [page 215]	com.sap.dh.dq.dataMask	Protect the personally identifiable or sensitive information by covering all or a portion of the data.	Active	
Data Quality [page 202]	DQMm Address Cleanse [page 236]	com.sap.dh.dq.dqmmAddressCleanse	Prepare address cleanse requests to be sent to SAP Data Quality Management, micro-services for location data.	Active	
Data Quality [page 202]	DQMm Client [page 245]	com.sap.dh.dq.dqmmClient	Send requests to Data Quality Management, micro-services for location data.	Active	
Data Quality [page 202]	DQMm Reverse Geo [page 248]	com.sap.dh.dq.dqmmReverseGeo	Prepare address cleanse requests to be sent to SAP Data Quality Management, micro-services for location data.	Active	
Data Quality [page 202]	Person and Firm Cleanse [page 251]	com.sap.dh.dq.personAndFirmCleanse	Identify the names of people and firms even when both types of names are in the same column.	Active	
Data Quality [page 202]	Validation Rule [page 259]	com.sap.dh.dq.validationRule	Create rules and routes records that pass through the Pass output port.	Active	
Data Workflows [page 280]	BW Process Chain [page 280]	com.sap.dh.bwprocesschain	Help execute an SAP BW process chain in an SAP BW system.	Active	
Data Workflows [page 280]	Data Transfer [page 282]	com.sap.dh.bwdatatransfer	Transfer data from an SAP BW system to an SAP Vora system.	Active	
Data Workflows [page 280]	HANA Flowgraph [page 284]	com.sap.dh.hanaflowgraph	Execute an SAP HANA flowgraph in an SAP HANA system.	Active	

Category	Operator	ID	Primary Use Case	Status	Replaced By
Deprecated Operators [page 543]	Spark Data Transform (Deprecated) [page 598]	com.sap.dh.data-transform	Provide capabilities to meet your data transformation requirement.	Deprecated	Data Transform [page 495] com.sap.data-transform
Data Workflows [page 280]	Notification [page 285]	com.sap.dh.notification	Send email notifications at certain points in the data workflow execution.	Active	
Data Workflows [page 280]	Pipeline [page 287]	com.sap.dh.vflow-pipeline	Execute a Modeler flow in an SAP Data Intelligence system.	Active	
Data Workflows [page 280]	SAP Data Services Job [page 289]	com.sap.dh.dsjob	Execute an SAP Data Services job in a remote system.	Active	
Data Workflows [page 280]	Workflow Merge (and) [page 291]	com.sap.dh.logical.AND	Control the flow of execution in a data workflow.	Active	
Data Workflows [page 280]	Workflow Merge (or) [page 292]	com.sap.dh.logical.OR	Control the flow of execution in a data workflow.	Active	
Data Workflows [page 280]	Workflow Split [page 293]	com.sap.dh.split	Duplicate an input message into two output messages.	Active	
Data Workflows [page 280] Data Workflows [page 280]	Workflow Terminator [page 294]	com.sap.dh.terminator	Shut down the data workflow execution.	Active	
Data Workflows [page 280]	Workflow Trigger [page 295]	com.sap.dh.trigger	Send a start message, which you can use to start a data workflow.	Active	
Files [page 295]	From File [page 296]	com.sap.file.from-File	Extract the path attribute from a file message.	Active	
Files [page 295]	List Files [page 296]	com.sap.file.list	List files and directories from various storage services.	Active	
Files [page 295]	Monitor Files [page 300]	com.sap.file.monitorFiles	Monitor file events in a directory from various storage services.	Active	

Category	Operator	ID	Primary Use Case	Status	Replaced By
Files [page 295]	Read File [page 303]	com.sap.file.read	Read the content of files from different storage services.	Active	
Files [page 295]	Remove File [page 306]	com.sap.file.removeFile	Remove files and directories from various storage services.	Active	
Files [page 295]	To File [page 308]	com.sap.file.toFile	Convert the input to a File type.	Active	
Files [page 295]	Write File [page 309]	com.sap.file.write	Write files to various services.	Active	
Hadoop and Spark [page 318]	Livy Spark Submit [page 318]	com.sap.livy.submit	Submit jobs to a cluster using the Livy REST API.	Active	
Hadoop and Spark [page 318]	Submit Hadoop Job [page 322]	com.sap.hadoop.submitJob	Submit jobs to Hadoop clusters provided by different cloud providers.	Active	
Deprecated Operators [page 543]	PA Automated (Deprecated) [page 579]	com.sap.ml.pa.automatedAnalytics	Apply the PA Automated Analytics product for prediction of values.	Active	
Deprecated Operators [page 543]		com.sap.ml.model.consumer	Consume a model in the model repository.	Deprecated	Artifact Consumer [page 419] com.sap.ml.artifact.consumer
Deprecated Operators [page 543]		com.sap.ml.model.producer	Push a model to the model repository.	Deprecated	Artifact Producer [page 420] com.sap.ml.artifact.producer
Processing [page 328]	Command Executor [page 329]	com.sap.system.commandExecutor	Execute the given command for each arrival of a message on the stdin port.	Active	
Processing [page 328]	Constant Generator [page 330]	com.sap.util.constantGenerator	Represent a constant string that may be emitted into the graph.	Active	
Processing [page 328]	Data Generator [page 331]	com.sap.util.dataGenerator	Generate random sample data every 500 ms.	Active	

Category	Operator	ID	Primary Use Case	Status	Replaced By
Processing [page 328]	Go Message Operator [page 332]	com.sap.system.golangengine2	Use the message type with a Go script.	Active	
Processing [page 328]	Go Operator [page 333]	com.sap.system.golangExecutor2	Executes Go code within a graph.	Active	
Processing [page 328]	Go String Operator [page 339]	com.sap.system.golangengine2	Execute Go code within a graph.	Active	
Processing [page 328]	JavaScript Operator [page 340]	com.sap.system.jsoperator	Execute JavaScript snippets within a graph.	Active	
Processing [page 328]	JavaScript (v2) [page 346]	com.sap.system.jsengine.v2	Execute JavaScript snippets within a graph and allow another part of the code to be executed until the blocking call returns.	Active	
Processing [page 328]	JS Message Operator [page 352]	com.sap.system.jsengine	Use the message type with the JavaScript operator.	Active	
Processing [page 328]	JS String Operator [page 353]	com.sap.system.jsengine	Handle inputs and generate outputs.	Active	
Processing [page 328]	Message Generator [page 354]	com.sap.util.data-MessageGenerator	Generate a message with random sample data every 500 ms.	Active	
Processing [page 328]	Process Executor [page 355]	com.sap.system.processExecutor	Start a process and provide given contiguous streams to it.	Active	
Processing [page 328]	Python 3 Operator [page 356]	com.sap.system.python3Operator	Define a script that offers some convenience functions provided by the API object.	Active	
Processing [page 328]	R Client 3 [page 364]	com.sap.system.rClient3	Run R code defined by you in the Rserve.	Active	
Processing [page 328]	Stdin/out Port Muxer [page 376]	com.sap.system.process.stdmuxer	Map multiple input ports to stdin of a process and multiplex the output of stdout to multiple output ports.	Active	

Category	Operator	ID	Primary Use Case	Status	Replaced By
Processing [page 328]	Streaming Analytics Projects [page 483]	com.sap.streamingproject	Run a Streaming Analytics project.	Active	
SAP HANA [page 388]	Initialize HANA Table [page 388]	com.sap.hana.init-Table	Initialize one or more tables on an SAP HANA database	Active	
SAP HANA [page 388]	Read HANA Table [page 392]	com.sap.hana.readTable	Read data from a table in SAP HANA.	Active	
SAP HANA [page 388]	Run HANA SQL [page 397]	com.sap.hana.runSQL	Execute user-provided SQL statements on an SAP HANA database.	Active	
SAP HANA [page 388]	Write to HANA Table [page 399]	com.sap.hana.writeTable	Ingest data into one or more tables on an SAP HANA database.	Active	
SAP Integration [page 401]	Cloud Data Integration Consumer [page 402]	com.sap.dh.cloud_data_integration	Use Flowagent for execution.	Active	
SAP Integration [page 401]	SAP Analytics Cloud Formatter [page 403]	com.sap.sac.formatter	Convert message.table input data to message format before sending to SAP Analytics Cloud.	Active	
SAP Integration [page 401]	SAP Analytics Cloud Producer [page 406]	com.sap.sac.producer	Send data to the SAP Analytics Cloud application.	Active	
SAP Integration [page 401]	SAP CPI-PI iFlow [page 414]	com.sap.cloud.cpi.pi	Provide the possibility to trigger iFlows in an SAP CPI system.	Active	
Deprecated Operators [page 543]		com.sap.ml.leonardo.MLFFunctionalServices	Prepare a request message for the Leonardo Machine Learning Foundation (Leonardo MLF) services.	Deprecated	Please check SAP Note 2739161  for more information.
Deprecated Operators [page 543]		com.sap.ml.leonardo.objectDetectionInference	Prepare a request message for the Object Detection Inference service.	Deprecated	Please check SAP Note 2739161  for more information.

Category	Operator	ID	Primary Use Case	Status	Replaced By
Deprecated Operators [page 543]	Functional Services (Deprecated) [page 556]	com.sap.ml.functionalServices	Use ML pre-defined functional services.	Deprecated	Please check SAP Note 2739161  for more information.
SAP Machine Learning Core Operators [page 417]	Artifact Consumer [page 419]	com.sap.ml.artifact.consumer	Retrieve and output a registered artifact.	Active	
SAP Machine Learning Core Operators [page 417]	Artifact Producer [page 420]	com.sap.ml.artifact.producer	Register the artifact given as input and return the artifact's generated ID.	Active	
SAP Machine Learning Core Operators [page 417]	HANA ML Forecast Operator [page 424]	com.sap.ml.hana.forecast	Connect to an SAP HANA database and leverages algorithms from SAP HANA Predictive Analytics Library (PAL) or SAP HANA Automated Predictive Library (APL) that do not require a model to be persisted.	Active	
SAP Machine Learning Core Operators [page 417]	HANA ML Inference Operator [page 432]	com.sap.ml.hana.inference	Connect to a HANA database and apply the specified trained Machine Learning model leveraging SAP HANA Predictive Analytics Library (PAL).	Active	
SAP Machine Learning Core Operators [page 417]	HANA ML Training Operator [page 437]	com.sap.ml.hana.training	Connect to a SAP HANA database and generate a Machine Learning model using the specified algorithm leveraged by SAP HANA Predictive Analytics Library (PAL).	Active	
SAP Machine Learning Core Operators [page 417]	Model Serving Operator [page 444]	com.sap.ml.modelServingOperator	Deploy a Machine Learning (ML) model and handle inference requests for the model.	Active	

Category	Operator	ID	Primary Use Case	Status	Replaced By
SAP Machine Learning Core Operators [page 417]	Submit Metrics [page 453]	com.sap.ml.submitMetrics	Submit your model metrics to the ML Tracking API. The metrics can be viewed in ML Scenario Manager.	Active	
SAP Machine Learning Core Operators [page 417]	ML Training [page 454]	com.sap.ml.training	Dispatch Machine Learning training jobs for execution.	Active	
SAP Vora [page 456]	Vora Avro Decoder [page 457]	com.sap.vora.pre-ingestor	Decode coded Avro, CSV, and JSON messages and offer the possibility to process the decoded messages before storing them in SAP Vora using SAP Vora Ingestor.	Active	
SAP Vora [page 456]	Vora Avro Encoder [page 461]	com.sap.vora.post-assembler	Create Avro, CSV, or JSON messages from the Vora messages containing record objects.	Active	
SAP Vora [page 456]	Vora Avro Ingestor [page 462]	com.sap.vora.avro-ingestor	Ingest data into SAP Vora based on the incoming Avro or other text messages in CSV and JSON.	Active	
SAP Vora [page 456]	Vora Client [page 470]	com.sap.vora.client2	Run statements on the SAP Vora Transaction Coordinator.	Active	
SAP Vora [page 456]	Vora Ingestor [page 473]	com.sap.vora.ingestor	Ingest data into SAP Vora based on incoming records messages.	Active	
SAP Vora [page 456]	Vora Loader [page 477]	com.sap.vora.vora-loader	Load data into SAP Vora.	Active	
Structured Data Operators [page 494]	Data Transform [page 495]	com.sap.data-transform	Provide wide variety of options to meet your data transformation needs.	Active	
Structured Data Operators [page 494]	SQL Consumer [page 498]	com.sap.data-base.sql.consumer	Read from a specified database using native SQL.	Active	

Category	Operator	ID	Primary Use Case	Status	Replaced By
Structured Data Operators [page 494]	Structured File Consumer [page 501]	com.sap.storage.consumer	Read from any cloud storage.	Active	
Structured Data Operators [page 494]	Structured File Producer [page 503]	com.sap.storage.producer	Read from any structured data operators and produce a file (CSV, ORC, or PARQUET) in the specified storage.	Active	
Structured Data Operators [page 494]	Table Consumer [page 507]	com.sap.database.table.consumer	Read from an specified database table or view, and produce a structured output.	Active	
Structured Data Operators [page 494]	Table Producer [page 511]	com.sap.database.table.producer	Read from any structured operators, and produce a table in the specified database.	Active	
Deprecated Operators [page 543]	Text Analysis Connector (Deprecated) [page 599]	com.sap.textanalysis.taconnector	Connect to a text analysis server using an external connector binary.	Deprecated	Please check SAP Note https://launchpad.support.sap.com/#/notes/2739161 for more information.
Tables [page 512]	Decode Table [page 512]	com.sap.table.decode	Decode tabular data from one of the supported formats into table messages.	Active	
Tables [page 512]	Generate Table [page 516]	com.sap.table.generate	Generate Table Messages with a user-defined schema and random data.	Active	
Utilities [page 518]	Avro Serializer [page 519]	com.sap.util.avro.Serializer	Convert CSV encoded lines and produce the contents as Avro encoded messages.	Active	
Utilities [page 518]	Avro Deserializer [page 520]	com.sap.util.avro.Deserializer	Convert Avro-encoded messages and produce the contents as CSV-encoded lines.	Active	

Category	Operator	ID	Primary Use Case	Status	Replaced By
Utilities [page 518]	Flowgraph Compiler [page 521]	com.sap.bdh.flowgraphCompiler	Compile incoming flowgraph XML strings into jars, which can be executed via Spark.	Active	
Utilities [page 518]	Format Converter [page 522]	com.sap.util.formatConverter	Convert blobs between CSV, JSON, and XML.	Active	
Utilities [page 518]	Graph Terminator [page 525]	com.sap.util.graphTerminator	Terminate the current graph execution.	Active	
Utilities [page 518]	Header Remover [page 526]	com.sap.util.headerRemover	Remove specified headers from an SAP Vora message.	Active	
Utilities [page 518]	Histogram [page 527]	com.sap.util.histogram	Receive a numeric value as input and increment the counter of the bin that corresponds to the range in which the value lies.	Active	
Utilities [page 518]	Histogram Plotter [page 529]	com.sap.util.histogramPlotter	Receive a string-converted Modeler message as input with an array of integers in the body and rangeMin and rangeMax fields in the header.	Active	
Utilities [page 518]	HTML Viewer [page 529]	com.sap.util.htmlViewer	Render HTML code on the browser as it arrives at the input port.	Active	
Utilities [page 518]	Message Aggregator [page 530]	com.sap.util.aggregator	Aggregate input messages in a custom manner specified by JavaScript functions that you can enter.	Active	
Utilities [page 518]	Message Key Store [page 533]	com.sap.vora.messagekeystore	Replicate messages received on the input ports to the output ports based on message history.	Active	

Category	Operator	ID	Primary Use Case	Status	Replaced By
Utilities [page 518]	Py3 Data Generator [page 537]	com.sap.util.data-generatorpy3	Generate random sample data every 500ms.	Active	
Utilities [page 518]	Py3 Type to Blob [page 537]	com.sap.util.python3TypeToBlob	Serialize any Python3 data received in its input port and send the result to the output port.	Active	
Utilities [page 518]	Py3 Type to String [page 538]	com.sap.util.python3TypeToString	Convert any Python 3 data received in its input to string by feeding it into the standard Python str(.) function.	Active	
Utilities [page 518]	Terminal [page 539]	com.sap.util.terminal	Represent a terminal window in the browser.	Active	
Utilities [page 518]	Wiretap [page 541]	com.sap.util.wiretap	Wiretap a connection between two operators in a Modeler graph and display the traffic to the browser window or to an external web-socket client that connects to this operator.	Active	
Deprecated Operators [page 543]		com.sap.blob.consumer	Consume a blob in the blob repository.	Deprecated	Please check SAP Note 2739161  for more information.
Deprecated Operators [page 543]		com.sap.blob.producer	Push a blob to the blob repository.	Deprecated	Please check SAP Note 2739161  for more information.
Deprecated Operators [page 543]	SAP ABAP ODP Object Consumer (Deprecated) [page 594]	com.sap.dh.ds.abap.odp.consumer	Use Flowagent for execution.	Deprecated	ABAP ODP Reader [page 46] com.sap.abap.odp.read
Deprecated Operators [page 543]	Create TA Request (Deprecated) [page 553]	com.sap.textanalysis.tarequestcreator	Create a request for the Text Analysis Connector operator in JSON format.	Deprecated	Please check SAP Note 2739161  for more information.

Related Information

[ABAP \[page 29\]](#)
[Connectivity \[page 51\]](#)
[Connectivity \(via Flowagent\) \[page 131\]](#)
[Converter \[page 194\]](#)
[Data Quality \[page 202\]](#)
[Data Workflows \[page 280\]](#)
[Files \[page 295\]](#)
[Hadoop and Spark \[page 318\]](#)
[Multiplexer \[page 327\]](#)
[Processing \[page 328\]](#)
[SAP HANA \[page 388\]](#)
[SAP Integration \[page 401\]](#)
[SAP Machine Learning Core Operators \[page 417\]](#)
[SAP Vora \[page 456\]](#)
[Streaming Analytics \[page 483\]](#)
[Structured Data Operators \[page 494\]](#)
[Tables \[page 512\]](#)
[Utilities \[page 518\]](#)
[Others \[page 543\]](#)
[Deprecated Operators \[page 543\]](#)

2.1 ABAP

Use the ABAP operators to access data in ABAP-based remote systems from SAP Data Intelligence.

General Concepts

The following general concepts apply to some or all of the ABAP operators. For operator-specific aspects, see the descriptions of the individual operators.

Authorizations

The user specified in the ABAP connection needs authorizations for the operator and the data that is selected. For details, see the section about user authorizations in SAP Note [2981615](#). This authorization concept is valid as of the following releases:

- DMIS 2011 SP20
- DMIS 2018 SP05
- SAP S/4HANA Cloud 2102
- SAP S/4HANA 2020 FPS01

For older releases, see SAP Note [2831756](#).

Versioning

As of SAP Data Intelligence 3.0 and SAP Data Intelligence Cloud 2003, a versioning concept for ABAP operators is in place. This means that each operator may be available in more than one version. A new version is shipped, for example, when an operator gets a new output type or has been equipped with additional features. You can find a list of new operator versions and their features in the SAP Note for the corresponding DMIS SP or SAP S/4HANA release, for example [2860331](#) for DMIS 2018 SP03.

This versioning concept applies to ABAP operators and is different from the one that is in place for other types of operators within SAP Data Intelligence: On the SAP Data Intelligence side, you only have one version of each ABAP operator - the reason being that the ABAP operators you see in SAP Data Intelligence are just shells, while the actual operators reside in the ABAP-based system and are pulled from there when needed.

As a consequence, you may not see the latest ABAP operator versions in the most recent version of the Modeler for SAP Data Intelligence if your ABAP-based source system does not have the prerequisites to provide the latest operator versions. (This is true, for example, for DMIS 2011 SP17.) In this case, an upgrade to the latest DMIS version would be required for you to get the most recent version of the operators.

As a rule, you should use the latest version that is available in the connected SAP system. If, however, you are working with an older version and want to keep using it for consistency reasons, that's also possible.

There may also be other reasons for using an older version. For example, one version may use the output type `String` in the output port, while the newer version uses the output type `Message`, and you want to use `String`. For more information about the differences between available versions of an operator, see the [New Features](#) section in the operator description. To find out which configuration parameters are relevant for a version of an operator, see the [Relevant in Version](#) column in the list of configuration parameters for the operator.

Subscriptions - and How to Use Them to Resume Operator Execution

Versions V1 and V2 of the SLT Connector, ABAP ODP Reader, and ABAP CDS Reader operators now have a feature called subscription, which enables you to resume execution of a graph containing any of these operators after it has been stopped - no matter whether it was stopped automatically due to an error or manually by a user. For details regarding availability of this feature, see the individual operator descriptions.

If you have a graph that includes one of these operators and want to start it for the first time, you choose subscription type [New](#) and enter a unique meaningful name for the subscription. The system then automatically creates a technical ID (subscription ID) for this subscription in the ABAP-based source system. (This subscription ID is needed, for example, to follow up on technical issues with a subscription.)

When a graph is stopped (be it manually or due to an error), you can start the data load process again by resuming the graph. To do so, change the subscription type to [Existing](#) and choose the subscription ID for the subscription name you entered when starting the graph for the first time. Note that in some cases it is technically not possible to resume the data load process at exactly the point where it stopped, which can result in missing or duplicate records in the target environment.

As a rule, a subscription and its subscription ID are not deleted automatically, and you have to remove them manually as described below. Exceptions to this rule are explained in the individual operator descriptions. Once you have deleted a subscription, you can no longer resume the related data load.

Every time a graph is stopped, the system does a check for subscriptions whose graphs haven't been updated for 30 days or longer and deletes these subscriptions. To adjust the default value for the retention period, in transaction SM30, open the maintenance view for table DHBAS_CONFIG (or LTBAS_CONFIG for DMIS, respectively) and either create a new value or overwrite the existing value for the parameter SUBSCR_RETENTION_IN_DAYS.

Note: Due to UI limitations, the fields are not dynamically filled with the corresponding values when you resume a graph by choosing an existing subscription. However when the graph starts to run again it uses the available values for the subscription.

How To Remove Subscriptions

After a graph has been stopped, the subscription is kept intact so that you can continue executing the graph at a later point in time. If you do not want to resume a graph anymore, you can use the Resilience Subscription Eraser operator to discard the subscription. For details on where and how this functionality is available, see the SAP Note for the DMIS Support Package or SAP S/4HANA release you are using, for example [2860331](#)  for DMIS 2018 SP03.

Note

After you erase a subscription ID, you may still be able to see it in the subscription list for a short period. This is because SAP Data Intelligence caches the subscription list for one minute. The subscription ID will disappear from the list when the cache period has elapsed.

To remove a subscription using the Resilience Subscription Eraser operator, proceed as follows:

1. In the Modeler, search for the SAP ABAP operator and drag it to the graph.
2. For the ABAP connection, use the same connection that you used with the operator (SLT Connector, ABAP ODP Reader, or ABAP CDS Reader) in the data loading graph.
3. Select Resilience Subscription Eraser V1.
4. Select the corresponding reader object:
 - All (subscriptions generated by all ABAP operators)
 - Tables (subscriptions generated by the SLT Connector operator)
 - CDS Views (subscriptions generated by the ABAP CDS Reader operator)

- ODP Objects (subscriptions generated by the ABAP ODP Reader operator)
- 5. In the dropdown list of subscription IDs for stopped graphs, choose the one that you want to erase.
- 6. Add a Graph Terminator to the graph and connect the out port of the Eraser operator with the in port of the Graph Terminator.
- 7. Run the graph.

Related Information

[ABAP Converter \[page 32\]](#)

[ABAP CDS Reader \[page 33\]](#)

[SLT Connector \[page 38\]](#)

[Cluster Table Splitter \[page 43\]](#)

[Custom ABAP Operator \[page 45\]](#)

[ABAP ODP Reader \[page 46\]](#)

[SAP ABAP Operator \[page 50\]](#)

2.1.1 ABAP Converter

Use the ABAP Converter operator to convert a generic ABAP data type to string using the CSV, JSON, or XML format.

General Concepts

The following general concepts are relevant for this operator:

- Authorizations

For a description of these concepts, see [ABAP \[page 29\]](#).

Configuration Parameters

Name	ID	Type	Description
ABAP Connection	connectionID	string	Mandatory. The connection information for the remote ABAP system. You can select a predefined ABAP connection from the Connection Manager. Default: ""
Format	format	string	The output format. Supported formats are CSV, JSON, and XML. Default: "CSV"

Input

Input	Type	Description
in	abap.*	Contents to be converted.

Output

Output	Type	Description
out	string	Contents that were converted.

2.1.2 ABAP CDS Reader

Use the ABAP CDS Reader operator to read data from a CDS view.

You can specify which action the system should take:

- Read all the data from the CDS view once (initial load)
- Initial load and then follow up on subsequent changes (replication)
- Follow up on changes only without doing an initial load first (delta load)

The output data type depends on the type of the CDS view that is being read. It is extended with two additional fields for the CDS view name and the operation ("I" for Insert, "U" for Upsert and "D" for Delete).

General Concepts

The following general concepts are relevant for this operator:

- Authorizations
- Versioning
- Subscriptions

For a description of these concepts, see [ABAP \[page 29\]](#).

New Features

As of V2, the output type for this operator has changed from `abap.*` to `message`, which also means that ABAP metadata is available for CDS Reader. For more information, see below under *Output*

i Note

When you do an initial load using V0 or V1 of the ABAP CDS Reader operator, it sends out an empty string when all the data has been transferred. The subsequent operator takes this empty string as the signal that the initial load is finished, and proceeds with the next actions (for example, 'stop the graph'). Since the output port type is `abap.*`, you can use the ABAP Converter operator to format the output data. For the empty string, the ABAP Converter operator can format the empty string to the empty content. The CDS view with `dataExtraction` annotations must be enabled for the ABAP Converter operator according to the format value of the property `format`.

csv: ""

json: {"DATA":""}

xml: <?xml version="1.0" encoding="utf-16"?> <asx:abap xmlns:asx="http://www.sap.com/abapxml" version="1.0" > gt; gt; gt; gt;

The CDS view with `dataExtraction` annotations must be enabled for replication (including initial load) and replication (delta load) extraction. As of SAP S/4HANA 2020, you can use the CDS view `I_DataExtractionEnabledView` to check if the actions replication (including initial load) and replication (delta load) are possible for a CDS view.

Configuration Parameters

Name	ID	Type	Description	Relevant in Version
ABAP Connection	connectionID	string	Mandatory. The connection information for the remote ABAP system. You can select a predefined ABAP connection from the Connection Manager. Default: ""	V0, V1, V2
Version	operatorID	string	Mandatory. The specific version of the operator. Default: ""	V1, V2
ABAP CDS Name	cdsname	string	The name of the CDS view. Note that depending on authorizations not all CDS views are available. Default: ""	V0, V1, V2
Subscription Type	subscriptionType	string	Can take one of the following values: - New - new subscription - Existing - reuse existing subscription Default: "New"	V1, V2
Subscription Name	subscriptionName	string	The name of the subscription Default: ""	V1, V2
Subscription ID	subscriptionID	string	The ID of the subscription. Default: ""	V1, V2

Name	ID	Type	Description	Relevant in Version
Transfer Mode	action	string	Can take one of the following values: - Initial Load - initial load only - Replication - replication of delta information (including initial load) - Delta Load - replication of delta information only (no initial load) - Default: "Initial Load"	V0, V1, V2
Records per Roundtrip	Chunk size	integer	Maximum number of records to be output in one go *)	V0, V1, V2

*) The actual number of records per roundtrip depends on a variety of factors, but cannot be larger than the value you specify here. Possible values are all integer numbers from 1 to 1,000,000. Default: 50,000.

Additional Information - Subscriptions and Related Topics

Availability Information

- The subscription feature is available for this operator as of SAP S/4HANA 2020 FPS00.
- The Resilience Subscription Eraser operator does not work for CDS Reader subscriptions in SAP S/4HANA OP 2020 FPS00. For more information, see [2967162](#).
- V2 of the ABAP CDS Reader is available as of SAP Data Intelligence 3.0, but not with older on-premise releases of SAP Data Intelligence.

Function

When you create a graph that includes the ABAP CDS Reader, you need to set the subscription type to [New](#) and enter a subscription name in the configuration panel.

❁ Example

You want to replicate a CDS view called EXAMPLECDV for the first time. You enter the following values in the configuration panel:

Subscription Type	New

Subscription Name	20200101EXAMPLECDSV (you can enter any name here)
ABAP CDS Name	EXAMPLECDSV
Transfer Mode	Replication

To resume data loading for a graph after it was stopped, set the subscription type to [Existing](#) and choose the subscription ID that was generated automatically for your subscription name as well as the CDS name and the transfer mode from the configuration panel.

You can find the subscription ID for your subscription name in either of the following ways:

- Check the process logs of your executed pipeline in the Modeler
- Look it up in the Resilience Subscription Eraser operator.
- Find it in table DHAPE_SUBSCR (or LTAPE_SUBSCR, respectively - depending on your ABAP system) in the source ABAP-based system.

Note

- If the transfer mode is [Initial Load](#) and the initial load process was completed successfully, the subscription is removed automatically.
- If the transfer mode is [Initial Load](#) and the graph is stopped before the initial load finishes, or if the transfer mode is [Replication](#) or [Delta Load](#), the data loading is stopped when the graph is stopped, but the subscription is **not** removed.
- [CDS View](#) and select the subscription ID that you want to use. For more information about how to proceed, see [ABAP \[page 29\]](#).

Input

- None.

Output

- For V0 and V1: outData: Output the CDS data; type: `abap.*`
- As of V2: outMessageData: Output the CDS data; type: `message`

If you use V2 of the CDS Reader, the output operator provides a message output with metadata information in the header of the message including information about the column names with respective data types as well as the lastBatch attribute (type boolean) that indicates if the current batch is the last one.

Attributes:

- Fields: object metadata from ABAP source system including field names and data types provided in json format
- message.lastBatch (type boolean): boolean that indicates if the current batch is the last one in the data ingestion process

Note: The `lastBatch` attribute is currently meant to be used in initial load scenarios only. If you run the graph in delta load or replication mode, the graph is expected to run permanently for capturing changes, and therefore the `lastBatch` attribute will never switch to 'true'.

- `message.batchIndex` (type integer): number that indicates which batch is being processed, starting from 0.

2.1.3 SLT Connector

Use the SLT Connector operator to establish a connection between SAP Landscape Transformation Replication Server (SLT) and SAP Data Intelligence.

You can then use SLT to replicate tables from a source system into SAP Data Intelligence by creating a graph that includes SLT Connector for every table that you want to replicate.

Before you use this operator in a graph, you first need to create an SLT configuration in the ABAP-based system in which SLT is installed. To do so, run transaction code `LTRC` in the SLT system and specify the scenario type as *SAP Data Hub / SAP Data Intelligence*. For more information about replicating data to SAP Data Intelligence, see https://help.sap.com/viewer/product/SAP_LANDSCAPE_TRANSFORMATION/2.0.14/en-US.

The part that you need later (when using the operator) is the mass transfer ID that is automatically assigned to your configuration: You have to specify it together with the table name and connection information.

When you then run the related graph, the table is automatically assigned to the mass transfer ID you defined previously. When the initial load is completed, when you choose transfer mode `Replication`, the system replicates the delta data of the source table and provides you with information about the original operation for each record (I for Insert, D for Delete, U for Update).

i Note

When you stop a graph in the Modeler (in SAP Data Intelligence), SLT Connector suspends the replication process for the corresponding table in SLT.

General Concepts

The following general concepts are relevant for this operator:

- Authorizations
- Versioning
- Subscriptions

For a description of these concepts, see [ABAP \[page 29\]](#).

New Features

As of V2, the output type for this operator has changed from `abap.*` to `message`, which also means that ABAP metadata is available. For more information, see below under *Output*.

Configuration Parameters

Name	ID	Type	Description	Relevant in Version
ABAP Connection	connectionID	string	Mandatory. The connection information for the ABAP-based remote system. You can select a predefined ABAP connection from the Connection Manager. Default: ""	V0, V1, V2
Version	operatorID	string	Mandatory. The specific version of the operator to use. Default: ""	V1, V2
Mass Transfer ID	Mass Transfer ID	string	The name of the mass transfer ID Default: ""	V0, V1, V2
Table Name	Table Name	string	The name of the relevant table. Note: For logical tables that are part of a cluster table (such as BSEG), specify the name of the cluster table (physical table), for example RFBLG. Use the Cluster Table Splitter operator and link the output port of SLT Connector to the corresponding port of the Cluster Table Splitter. (In the example, this would be bseg.) If you do so, you only get the logical tables for which you specified ports. Otherwise, you get all logical tables that belong to the cluster table.	V0, V1, V2

Name	ID	Type	Description	Relevant in Version
Subscription Type	subscriptionType	string	Can take one of the following values: • New - new subscription • Existing - reuse existing subscription Default: "New"	V1, V2
Subscription Name	subscriptionName	string	The name of the subscription. Default: ""	V1, V2
Subscription ID	subscriptionID	string	The ID of the subscription. Default: ""	V1, V2
Chunk Size	Chunk size	string	See below for possible values Default: ""	V0
Records per Roundtrip	chunkSize	integer	Maximum number of records to be output in one go *)	V1, V2
Transfer Mode	action	string	Can take the following values: • Initial Load - initial load only • Replication - replication of delta information (including initial load) • Delta Load - replication of delta information only (no initial load) Default: "Initial Load"	V0, V1, V2

*) The actual number of records per roundtrip depends on a variety of factors, but cannot be larger than the value you specify here. Possible values are all integer numbers from 1 to 500,000. Default: 50,000.

The Chunk Size parameter (relevant for V0 only) can take the following values:

- 1 record: Output the data record by record

- 1 portion: Output the source data in batches, one portion at a time. Depending on the transfer mode, it can take different values:
 - Initial Load: The package size (in bytes) used for the initial load. By default, the package size is 8,000,000 bytes.
 - Delta Load: The portion size is a number between 1 and 5,000 records.
 - Replication: For the initial load, the portion size has the same meaning as specified above for initial load. For the replication itself, the portion size is a number between 1 and 5,000 records.

Additional Information - Subscriptions and Related Topics

Availability Information

- The subscription feature for this operator is available as of DMIS 2018 SP04, DMIS 2011 SP19, and SAP S/4HANA 2020 FPS00.
- SAP S/4HANA 2020 FPS01:
 - Resuming a dead graph is possible only if SLT Connector caught the error and forced the graph to die.
 - If a graph containing the SLT Connector is stopped by means of the Stop button in the Modeler, the subscription is cleared automatically, and the table is stopped in the SLT system. If an error occurs and SLT Connector catches the error, the graph will be dead, and the subscription remains in the system. You can resume the graph by choosing subscription type *Existing* and the subscription ID for the subscription name you entered earlier, and you can remove the subscription as described below.
- DMIS 2018 SP05 and DMIS 2011 SP20: If a graph is stopped after the initial load process has been completed successfully, the system automatically removes the subscription and removes the table in SLT. If, however, a running graph with transfer mode *Initial Load* is stopped before the data loading finishes, or if a running graph with transfer mode *Replication* or *Delta Load* is stopped, the subscription is not removed, and the table is suspended (but not stopped) in the SLT system. If you use the Cluster Table Splitter followed by SLT Connector, the subscription is **not** removed even after the data loading is finished.
- V2 of the SLT Connector is available as of SAP Data Intelligence 3.0, but not with older on-premise releases of SAP Data Intelligence.
- In combination with some DMIS releases, a minimum release of SAP Data Intelligence is required for the functionality to work. For details, see SAP Note [2892183](#) 

Function

When you create a graph that includes the SLT Connector, you need to set the subscription type to *New* and enter a subscription name in the configuration panel.

Example

You want to replicate the table SFLIGHT for the first time. You have created a configuration in the SLT system. Its mass transfer ID (MT ID) is 001. You enter the following values in the SLT Connector configuration panel:

Subscription Type	New
Subscription Name	20200101SFLIGHT (you can enter any name here)
Mass Transfer ID	001

Table Name	SFLIGHT
Transfer Mode	Replication

To resume data loading for a graph after it was stopped, set the subscription type to [Existing](#) and choose the subscription ID that was generated automatically for your subscription name as well as the MT ID and the table name from the configuration panel.

You can find the subscription ID for your subscription name in either of the following ways:

- Check the process logs of your executed pipeline in the Modeler
- Look it up in the Resilience Subscription Eraser operator.
- Find it in table DHAPE_SUBSCR (or LTAPE_SUBSCR, respectively - depending on your ABAP system) in the source ABAP-based system.

Note

- As of DMIS 2018 SP05 and DMIS 2011 SP20, you use the Resilience Subscription Eraser operator to remove a subscription that you don't need anymore: Set the [Reader](#) object in the configuration panel to [Tables](#), choose the subscription ID that you want to remove, and run the graph. The graph stops the table and removes the subscription from the SLT system.
- To load a table for which you removed a subscription earlier, set the subscription type to [New](#) to ensure that the table is loaded from scratch.
- As long as a graph has been stopped, but not yet removed, you cannot run another graph with subscription type [New](#) for the same table and MT ID.
- If you use the SLT Connector operator to integrate data from an ABAP source system, and your graph in SAP Data Intelligence terminates, you may get an error message like "Subscription <subscription_name> is being used by another graph" when you try to restart your failed graph. For more information about this error scenario, see SAP Note [3057246](#).

Input

- None.

Output

- For V0 and V1: outData: Output the CDS data; type: `abap.*`
- As of V2: outMessageData: Output the CDS data; type: `message`

If you use V2 of the SLT Connector, the output operator provides a message output with metadata information in the header of the message including information about the column names with respective data types as well as the `lastBatch` attribute (type `boolean`) that indicates if the current batch is the last one.

Attributes:

- Fields: object metadata from ABAP source system including field names and data types provided in json format
- message.lastBatch (type boolean): boolean that indicates if the current batch is the last one in the data ingestion process
Note: The lastBatch attribute is currently meant to be used in initial load scenarios only. If you run the graph in delta load or replication mode, the graph is expected to run permanently for capturing changes, and therefore the lastBatch attribute will never switch to 'true'.
- message.batchIndex (type integer): number that indicates which batch is being processed, starting from 0.

2.1.4 Cluster Table Splitter

Use the Cluster Table Splitter to replicate one or more (but not all) logical tables for a cluster table.

A cluster table in the ABAP-based remote system can consist of more than one logical table. The SLT Connector operator can read only the complete cluster table and provides you with the data of all logical tables. If you are interested only in some of the logical tables, you can use the Cluster Table Splitter operator to split a cluster table into its individual logical tables and then replicate only the logical tables you actually need.

To do so, you create an output port for each of the relevant logical tables, using the name of the logical table as the name of the output port. An output port can be connected to a subsequent operator. For V0 and V1 of the Cluster Table Splitter operator (that have output type abap.*), you can use the ABAP Converter operator to convert the data into string, XML, or JSON according to the settings of the output format.

Example

You want to replicate the logical table named `BSEG`, which is a logical table in cluster table `RFBLG`. To do so, you enter `RFBLG` as the table name for SLT Connector, then link the SLT Connector to the Cluster Table Splitter operator, and finally create an output port named `bseg`.

i Note

If there are special characters like `_` or `/` in the logical table names, remove these when creating the additional output port. For example, logical table `EXAMPLE_TABLE` should be mapped to port `exampletable`, and `/1LT/EXAMPLE` to `1ltexample`

General Concepts

The following general concepts are relevant for this operator:

- Authorizations
- Versioning

For a description of these concepts, see [ABAP \[page 29\]](#).

New Features

As of V2, the output type for this operator has changed from `abap.*` to `message`.

Configuration Parameters

Name	ID	Type	Description	Relevant in Version
ABAP Connection	connectionID	string	Mandatory. The connection information for the remote ABAP system. You can select a predefined ABAP connection from the Connection Manager. Default: ""	V0, V1, V2
Version	operatorID	string	The specific version of the operator. Default: ""	V1, V2

Input

Input	Type	Description
in	<code>abap.*</code> for V0 and V1 and <code>message</code> as of V2	SLT Connector transfers all the logical tables data into Cluster Table Splitter.

Output

Output	Type	Description
<output port name>	<code>abap.*</code> for V0 and V1 and <code>message</code> as of V2	Create one output port for each relevant logical table, using the table name as the port name.

2.1.5 Custom ABAP Operator

You can create your own custom operators in your ABAP-based remote system. To make these operators available in SAP Data Intelligence, you use the Custom ABAP Operator. The Custom ABAP Operator dynamically adapts to the custom operator you select in the remote system.

For more information, about how to integrate ABAP function modules with SAP Data Intelligence using the Custom ABAP Operator, see this [blog post](#) .

i Note

Only operators that were **not** created in the SAP namespace in the remote ABAP-based system are displayed for selection with the Custom ABAP Operator.

Procedure

1. Add the Custom ABAP operator to a graph;
2. Open the [Configuration Panel](#):
 1. Select a predefined connection for the remote ABAP system.
 2. Select an operator implementation from the remote ABAP system. The UI will adapt the Custom ABAP operator to the selected operator implementation.
 3. If required, enter the additional configuration values.
3. Connect the operator to other operators as usual.

For more details about how to create a custom ABAP operator, see the [User Guide for ABAP Integration in SAP Data Intelligence](#).

The following general concepts are relevant for this operator:

- Authorizations

For a description of these concepts, see [ABAP \[page 29\]](#).

Configuration Parameters

Name	ID	Type	Description
ABAP Connection	connectionID	string	Mandatory. The connection information for the ABAP-based remote system. You can select a predefined ABAP connection from the Connection Manager. Default: ""

Name	ID	Type	Description
ABAP Operator	operator	string	Mandatory. A predefined operator implementation from the remote ABAP system. Default: ""

Input

- None.

Output

- None.

2.1.6 ABAP ODP Reader

Use this operator to read data from an object for operational data processing (ODP object).

The following extraction modes are supported:

- FULL - one-time load of the entire ODP object:
As soon as it has output all data from the ODP queue, the ABAP ODP Reader (V0 and V1) sends an empty string to the subsequent operator to let it know that it can start. (As of V2, this is no longer necessary as the corresponding information is included in the metadata.)
- DELTA - Replication of delta information (including initial load)

As of DMIS 2011 SP19 and DMIS 2018 SP04, the ABAP ODP Reader supports projection and filtering if used together with the Data Transfer operator.

General Concepts

The following general concepts are relevant for this operator:

- Authorizations
- Versioning
- Subscriptions

For a description of these concepts, see [ABAP \[page 29\]](#).

New Features

As of V2, the output type for this operator has changed from `abap.*` to `message`, which also means that ABAP metadata is available. For more information, see below under *Output*.

Configuration Parameters

Name	ID	Type	Description	Relevant in Version
ABAP Connection	connectionID	string	Mandatory. The connection information for the ABAP-based remote system. You can select a predefined ABAP connection from the Connection Manager. Default: ""	V0, V1, V2
Version	operatorID	string	Mandatory. The specific version of the operator. Default: ""	V1, V2
Context	context	string	The name of the ODP context. Can take one of the following values: • ODP_BW • ODP_SAPI Default: ""	V0, V1, V2
ODP Object Name	odpname	string	The name of the ODP object - only used if the ODP object is not specified in the adapted dataset. Default: ""	V0, V1, V2
Extraction Mode	extractionMode	string	Can take one of the following values: • FULL • DELTA	V0, V1, V2

Name	ID	Type	Description	Relevant in Version
Polling Period	pollingPeriod	string	Only relevant for DELTA. The polling period specifies the amount of time in seconds until the operator will make the next attempt to fetch data after an attempt that did not return any data. Default: ""	V0, V1, V2
Subscription Type	subscriptionType	string	Can take one of the following values: • New - new subscription • Existing - reuse existing subscription Default: "New"	V1, V2
Subscription Name	subscriptionName	string	The name of the subscription. Default: ""	V1, V2
Subscription ID	subscriptionID	string	The unique identifier of the subscription. Default: ""	V1, V2

Additional Information - Subscriptions and Related Topics

Availability Information

- The subscription feature is available for this operator as of DMIS 2018 SP04 and DMIS 2011 SP19.
- V2 of the SLT Connector is available as of SAP Data Intelligence 3.0, but not with older on-premise releases of SAP Data Intelligence.

Function

When you create a graph that includes the ABAP ODP Reader, you need to set the subscription type to `New` and enter a subscription name in the configuration panel.

❖ Example

You want to do a delta load for an object called `EXAMPLEODP` under context `ODP_SAPI` for the first time. You enter the following values in the ODP Reader configuration panel:

Subscription Type	New
Subscription Name	20200101EXAMPLEODP (you can enter any name here)
Context	ODP_SAPI
ODP Object Name	EXAMPLEODP
Extraction Mode	DELTA

To resume data loading for a graph after it was stopped, set the subscription type to `Existing` and choose the corresponding subscription ID for your subscription name, ODP context, and ODP name from the configuration panel.

Note

- If the extraction mode is `FULL` and the load process was completed successfully, the subscription is removed automatically.
- If the extraction mode is `FULL` and the graph is stopped before the load process finishes, or if the extraction mode is `DELTA`, the data loading is stopped when the graph is stopped, but the subscription is **not** removed.
- To remove a subscription that you don't need anymore, you use the Resilience Subscription Eraser operator: Set the Reader object in the configuration panel to [ODP Objects](#) and choose the subscription ID that you want to remove.

Input

- None.

Output

- For V0: outRecord: output the data record by record; type: `abap.*`
- For V0: outTable: output the data in batches; note that the default batch size is 10 MB and cannot be changed. Type: `abap.*`
- For V1: outData: Output the ODP data in batches of a fixed size of 10 MB each; type: `abap.*`
- As of V2: outMessageData: Output the ODP data in batches of a fixed size of 10 MB each; type: `message`

If you use V2 of the ABAP ODP Reader, the output operator provides a message output with metadata information in the header of the message including information about the column names with respective data types as well as the `lastBatch` attribute (type `boolean`) that indicates if the current batch is the last one.

Attributes:

- Fields: object metadata from ABAP source system including field names and data types provided in json format

- `message.lastBatch` (type boolean): boolean that indicates if the current batch is the last one in the data ingestion process
Note: The `lastBatch` attribute is currently meant to be used in initial load scenarios only. If you run the graph in delta load or replication mode, the graph is expected to run permanently for capturing changes, and therefore the `lastBatch` attribute will never switch to 'true'.
- `message.batchIndex` (type integer): number that indicates which batch is being processed, starting from 0.

2.1.7 SAP ABAP Operator

Use the SAP ABAP Operator to make a standard operator implementation from your ABAP-based remote system available in SAP Data Intelligence. The SAP ABAP Operator dynamically adapts to the selected operator in the remote system.

i Note

Only operators that were created in the SAP namespace in the remote ABAP-based system are displayed for selection with the SAP ABAP operator.

Procedure

1. Add the SAP ABAP Operator to a graph.
2. Open the [Configuration Panel](#):
 1. Select a predefined connection for the ABAP-based remote system.
 2. Select an operator implementation from the ABAP-based remote system. The UI will adapt the SAP ABAP Operator to the selected operator implementation.
 3. If required, enter the additional configuration values.
3. Connect the operator to other operators as usual.

General Concepts

The following general concepts are relevant for this operator:

- Authorizations

For a description of these concepts, see [ABAP \[page 29\]](#).

Configuration Parameters

Name	ID	Type	Description
ABAP Connection	connectionID	string	Mandatory. The connection information for the remote ABAP system. You can select a predefined ABAP connection from the Connection Manager. Default: ""
ABAP Operator	operator	string	Mandatory. A predefined operator implementation from the remote ABAP system. Default: ""

Input

- None.

Output

- None.

2.2 Connectivity

Related Information

[AWS SNS Consumer \[page 52\]](#)

[AWS SNS Producer \[page 55\]](#)

[Google Pub/Sub Consumer \[page 58\]](#)

[Google Pub/Sub Producer \[page 62\]](#)

[HANA Client 2 \[page 65\]](#)

[HANA Monitor \[page 80\]](#)

[HTTP Client 2 \[page 82\]](#)
[HTTP Server \[page 86\]](#)
[Kafka Consumer 2 \[page 90\]](#)
[Kafka Producer \[page 96\]](#)
[MQTT Consumer \[page 102\]](#)
[MQTT Producer \[page 104\]](#)
[NATS Consumer \[page 107\]](#)
[NATS Producer \[page 108\]](#)
[OpenAPI Client \[page 109\]](#)
[OpenAPI Server \[page 116\]](#)
[Receive Email \[page 121\]](#)
[Send Email \[page 124\]](#)
[WAMP Consumer \[page 129\]](#)
[WAMP Producer \[page 130\]](#)

2.2.1 AWS SNS Consumer

Consumes publications from an Amazon Web Services Simple Notification Service topic and outputs them as a stream of messages.

Configuration Parameters

Name	ID	Type	Description
Connection	<code>connection</code>	object	Mandatory. An AWS SNS connection consisting of an AWS access key, account ID, key ID, and a region (defaults to <code>eu-central-1</code>).
Topic	<code>topic</code>	string	Name or Amazon Resource Name (ARN) of an SNS topic. If only a name is given, an ARN is created using the given region and account ID (in connection). The ARN is looked up on SNS. If it does not exist, the consumer fails regardless of the value of <code>Error handling</code>.

Name	ID	Type	Description
Subscription Name	subscriptionName	string	Name of the subscription to create or reuse on the AWS SNS service. The subscription name is used to create a subscription as well as a Simple Queue Service (SQS) standard queue which holds matching publications. If the subscription already exists, it is reused.
Error Handling	errorHandling	string	How the operator handles errors. Forward: does not fail the graph. Instead the error message is forwarded to output. The output messages have the attribute <code>message.error</code> set to true and the attribute <code>originalMessage</code> containing the input message that caused the error. Fail: fails the graph upon error. Ignore: only prints a warning in the log and ignores the error.
Maximum retries	maxRetries	integer	Refers to the number of retries performed by the operator when sending a request to the SNS API.
Maximum acknowledgment time	maxAckTime	integer	Maximum time period (in seconds) between when the consumer receives the message and it sends the acknowledgment to the AWS SNS. If not acknowledged within this period, the SNS service delivers the publication again to the consumer. If the Messages to acknowledge input port is not used, the consumer automatically sends the acknowledgment for a message upon receiving it from the SNS service.

Name	ID	Type	Description
Maximum message retention time	<code>maxRetentionPeriod</code>	integer	The length of time, in seconds, for which AWS SQS retains a message. Value can be between 60 seconds to 1,209,600 seconds (14 days).
Unsubscribe to topic on shutdown	<code>unsubscribeOnShutdown</code>	boolean	If true, when the graph is shut down, the subscription is removed, which results in no more matching publications being copied to the queue from the topic.
Remove SQS queue on shutdown	<code>removeQueueonShutdown</code>	boolean	If true, when the graph is shut down, the SQS queue, containing matching publications not consumed yet, is removed.
Proxy	<code>proxy</code>	string	HTTP proxy to be used by the HTTP client issuing SNS API calls.

Input

Input	ID	Type	Description
Messages to acknowledgment	<code>messageToAck</code>	message	The stream of Message IDs. Each publication received by the consumer has a unique ID assigned by the AWS SNS service. Using this ID, the publication can be explicitly acknowledged. The operator reads the message ID from the attribute named <code>msg.MessageID</code> in the received message.

Output

Output	ID	Type	Description
Output Messages	outputMessage	message	The stream of publications received by the consumer from the topic on AWS SNS. Each output message contains at least the following three attributes: • <code>msg.MessageID</code>: a unique ID assigned to each message published to AWS SNS. This ID can be used to explicitly acknowledge a received publication. • <code>msg.TopicArn</code>: The ARN of the topic that the message was published to. • <code>msg.PublishTime</code>: The timestamp assigned to the message by AWS SNS.

2.2.2 AWS SNS Producer

Receives a message from the input port and publishes it to an Amazon Web Services Simple Notification Service topic.

SNS accepts only string as the body of the message. If the input message has a non-string body, it will be encoded using Base64, encoding to a string, and an attribute named `msg.IsSnsBodyEncoded` with the value `true`, is added to the SNS message. If the SNS message is consumed via the corresponding consumer, decoding the body and removing the `msg.IsSnsBodyEncoded` attribute is done automatically.

Configuration Parameters

Parameter	ID	Type	Description
Connection	<code>connection</code>	object	Mandatory. An AWS SNS connection consisting of an AWS access key, account ID and key ID.
Topic	<code>topic</code>	string	Name or Amazon Resource Name (ARN) of an SNS topic. If only a name is given, an ARN is created using the given region and account ID (in connection). The ARN is looked up on SNS. If it already exists, it gets reused. If not, it will be created. The provided topic is the default topic that received messages will be published to. Alternatively, if there exists a <code>msg.TopicArn</code> attribute in the message, the value of this attribute is used as topic name and the message is published to this topic rather than topic. This attribute will also exist in the message received by the consumer. If topic is empty and the producer cannot get the topic name from the message attributes, the producer fails regardless of the value of <code>Error handling</code>.

Parameter	ID	Type	Description
Error handling	<code>errorHandling</code>	string	How the operator handles errors. Forward: does not fail the graph, instead the error message is forwarded to output. The output messages have the attribute <code>message.error</code> set to true and the attribute <code>originalMessage</code> containing the input message that caused the error. Fail: fails the graph upon error. Ignore: only prints a warning in the log and ignores the error.
Maximum retries	<code>maxRetries</code>	integer	Refers to the number of retries performed by the operator when sending a request to the SNS API.
Delete topic on shutdown	<code>deleteTopicOnShutdown</code>	boolean	If true, when the graph is shut down, the SNS topic is deleted. Currently, SNS deletes all attached subscriptions as well.
Proxy	<code>proxy</code>	string	HTTP proxy to be used by the HTTP client issuing SNS API calls.

Input

Input	ID	Type	Description
Messages to publish	messageToPublish	message	The stream of records to publish. If the input messages have <code>message.error</code> set to true, the behavior is based on the value of <code>Error Handling</code>. Messages must be UTF-8 encoded strings at most 256 KB in size. Message must have a body. An empty body is replaced with "". An SNS message with an empty string as its body, will be converted to a Modeler message with nil as its body. The message body must either a string or a byte array.

Output

Output	ID	Type	Description
Publishing results	publishResult	message	The result of publishing an input message is written to this output either as a successful publication, which would include assigned message ID for the publication (by SNS) and the topic it was published to, under the attributes <code>msg.MessageID</code> and <code>msg.TopicArn</code>, respectively.

2.2.3 Google Pub/Sub Consumer

Consumes publications from a Google Pub/Sub topic and outputs them as a stream of messages.

- If *Subscription name* refers to an existing subscription name on the Google Pub/Sub service, *Topic* can be empty, as an existing subscription is already attached to a topic. If subscription is to be created, *Topic* must

be provided and [Topic](#) must exist on the Pub/Sub service. Otherwise, the graph fails regardless of the value of [Fail on error](#).

- If [Fail on error](#) is false, upon error, the consumer only outputs the error message on [Error](#) output port and does not fail the graph.
- The [messageToAck](#) input port is optional. If connected, no automatic acknowledgment is sent and the user has to explicitly acknowledge received publications by submitting their message ID to this port. This allows using the Google Pub/Sub service as a message queue and ensures that publications that are not consumed successfully get delivered again to the same consumer or other consumers using the same [Subscription name](#).
- When using explicit acknowledgment, receiving more than one ack for the same publication or requesting acknowledgment for a publication that is not received by the consumer results in an error.

Configuration Parameters

Name	ID	Type	Description
Connection	connection	string	Mandatory. A Google Pub/Sub connection consisting of a GCP project ID and a JSON key file to access the Pub/Sub service.
Subscription Name	subscriptionName	string	Mandatory. The name of the subscription on the Google Pub/Sub service.
Topic	topic	string	Google Pub/Sub topic name.
Fail on Error	failOnError	boolean	Whether to fail the whole graph if the consumer encounters an error at runtime. Default: false

Name	ID	Type	Description
Maximum Acknowledgment Time (sec)	maxAckExtension	integer	Maximum time period between when the consumer receives the message and it sends the acknowledgment to the Google Pub/Sub service. If not acknowledged within this period, the Pub/Sub service delivers the publication again to the same consumer or any other consumer that is configured to use the same Subscription name. If the messageToAck input port is not used, the consumer automatically sends the acknowledgment for a message upon receiving it from the Pub/Sub service. Default: 600
Maximum Outstanding Publications	maxOutsandingMessages	integer	Maximum number of received but unacknowledged messages on the consumer. Upon reaching this maximum, the consumer stops pulling publications from the Pub/Sub service until one or more of the existing messages expire (ack deadline is passed) or acknowledged. A negative value indicates no limit for the number of unprocessed messages. Default: 1000

Name	ID	Type	Description
Synchronously pull messages	synchronousPull	boolean	If set, pull messages synchronously ensuring the buffer does not exceed the maximum outstanding messages. Otherwise, asynchronously pull messages as determined by the operator, not guaranteeing the maximum outstanding messages limit. Therefore, setting this configuration to true is the only way to guarantee that buffered unprocessed messages on the consumer do not exceed maximum outstanding messages. Default: False

Input

Input	Type	Description
messageToAck	message	Messages to be acknowledged. The stream of Message IDs. Each publication received by the consumer has a unique ID assigned by the Google Pub/Sub service. Using this ID, the publication can be explicitly acknowledged. The operator reads the message ID from the attribute named <code>msg.MessageID</code> in the received message.

Output

Output	Type	Description
<code>receivedMessage</code>	message	The stream of publications received by the consumer from the Topic on Google Pub/Sub service. Each output message contains the following extra two attributes: • <code>gcp.pubsub.message.ID</code>: a unique ID assigned to each message published to Google Pub/Sub. This ID can be used to explicitly acknowledge a received publication. • <code>gcp.pubsub.message.publishTime</code>: The timestamp assigned to the message by the Google Pub/Sub service upon its publication.
<code>error</code>	message	Error messages occurring during runtime are output on this port as a stream of messages (one for each error). The error message has the attribute <code>msg.KeyError</code> with the value <code>true</code> and the body of the message is the error message.

2.2.4 Google Pub/Sub Producer

Receives a message from the input port and publishes it to a Google Pub/Sub topic.

- The provided *Topic* is the default topic that received messages will be published to. Alternatively, if a `gcp.pubsub.topicName` attribute exists in the message, the value of this attribute is used as topic name and the message is published to this topic rather than *Topic*. This attribute will not be removed by the producer upon publish. If *Topic* is empty and the producer cannot get the topic name from the message attributes, the producer fails regardless of the value of *Fail on error*.
- *Create topic if it does not exist* applies only to *Topic*, not topic names mentioned in the message attributes.
- If *Fail on error* is false, upon error, the producer only outputs the error message on *Error* output port and does not fail the graph. The message to be published is discarded.
- *Subscription name to create* applies only to *Topic* and not topic names mentioned in the message attributes. If specified, the producer makes sure the subscription to *Topic* exists (if not, it creates the subscription) on Google Pub/Sub before processing input messages. Creating a subscription before publishing can ensure publications published to *Topic* are stored on Google Pub/Sub, even if at the time of publishing, no subscription to *Topic* exists. Google Pub/Sub does not guarantee storing undelivered publications when there are on matching subscriptions.

- If the number of publications received by the producer, but not yet published to the Google Pub/Sub service is more than *Maximum outstanding publications*, the producer does not process messages from *messageToPublish* input until one or more messages are finished publishing. Publishing a message is finished when the producer outputs its message ID on the *publishedMessage* output port, or an error occurs and the error is written to the *error* output port.
- Upon publish timeout, an error message is written to the *error* output port and if *Fail on error* is true, the graph fails.
- The encoding of the received input message is stored in an attribute named `message.encoding` in the message published to the Google Pub/Sub service. The Google Pub/Sub consumer uses this attribute to reconstruct the same SAP Data Intelligence message.
- As Google Pub/Sub supports only string as a key or value type of attributes, any attribute included in the input message to this operator must have strings as key and value.
- To improve the throughput of the operator, publications are batched. The value of *Publication batch size* must be smaller than *Maximum outstanding publications*. This is checked by the producer upon initialization.

Configuration Parameters

Name	ID	Type	Description
Connection	connection	object	Mandatory. A Google Pub/Sub connection consisting of a Google Cloud Platform (GCP) project ID and a JSON key file to access the Pub/Sub service.
Topic	topic	string	Google Pub/Sub topic name.
Create topic if it does not exist	createTopicIfNotExists	boolean	Whether to create <i>Topic</i> if it does not exist on the given Google Pub/Sub project. Default: true
Fail on Error	failOnError	boolean	Whether to fail the whole graph if the producer encounters an error at runtime. Default: false
Subscription Name to Create	subscriptionName	string	Ensure that the given subscription name exists for <i>Topic</i> .

Name	ID	Type	Description
Maximum Outstanding Publications	maxOutstandingPubs	integer	Maximum number of publications that are submitted to the operator but are not finished being published. Default: 1000
Publish Timeout (sec)	publishTimeout	integer	Maximum timeout in seconds for publishing a publication. Default: 60
Publication Batch Size	publishBatchSize	integer	Minimum number of publications that the producer needs to receive before sending a batch of publications to the Google Pub/Sub service. Default: 100
Delay Between Retrying Failed Publications (msec)	retryDelay	integer	Time to wait before retrying a failed publication. Default: 1000
Number of Retry Attempts for Failed Publications	retryAttempts	integer	Maximum number of retries for a failed publication. Upon reaching this limit, an error is generated and handled according to the value of <i>Fail on error</i> . Default: 5

Input

Input	Type	Description
messageToPublish	message	The stream of messages to publish.

Output

Output	Type	Description
publishedMessage	message	Upon successful publication, Google Pub/Sub assigns a unique ID to each publication. The producer outputs these IDs as a stream of messages on this port, with the message ID as value of the attribute <code>msg.MessageID</code> .
error	message	Error messages occurring during run-time are output on this port as a stream of message (one for each error). The error message has the attribute <code>msg.KeyError</code> with the value <code>true</code> and for errors relating to publishing a message, the original message can be retrieved using the attribute <code>originalMessage</code> . The body of the message is the error message.

2.2.5 HANA Client 2

Executes SQL statements and inserts CSV or JSON data into an SAP HANA instance. It supports three fundamental operations:

- **Executing arbitrary SQL code:**
SQL code arrives at port `sql`.
When executing stored procedures with a `CALL` statement, please note that only scalar output parameters are supported.
- **Executing prepared statements:**
A message arrives at port `data` with the `hana.preparedStatement` attribute, which names one of the prepared statement configured in [Prepared statements](#) to be executed. The message body must be of type:
 - `[] [] interface{};`
 - `[] interface{}()` containing `[] interface{};`
 - `[] [] string`, in case all statement parameters accept strings;
 - `[] [] float64`, in case all statement parameters accept integers or floating-point; or
 - `nil`, in which case the prepared statement is executed without arguments.If the body is none of the above types, an error is raised.
- **Inserting data into a predefined table:**
A message arrives at port `data` **without** the `hana.preparedStatement` attribute. The message body is parsed as either CSV or JSON (see [Input format](#)) and the resulting rows are inserted into the table identified by [Table name](#), according to the column names and types in [Table columns](#).
You can use a single HANA Client operator to perform all three operations, because the input ports work independently from one another.

Configuration Parameters

Title	ID	Type	Description
Connection	connection	object	Mandatory. The object containing the connection parameters to a HANA instance. i Note It may be necessary to enable TLS in your connection configuration. Error messages such as <code>i/o timeout</code>, <code>connection reset by peer</code> and <code>EOF</code> may indicate that a network firewall is dropping insecure packets. When TLS is enabled, the operator uses the "Validate hostname in certificate" (<code>hostnameInCertificate</code>) connection property for the TLS Server Name extension. If this property is empty, the operator falls back to <code>host</code>, which must be a hostname and not an IP address. Default: {}
Output in Batches	outputInBatches	bool	Determines whether output messages should be broken by the number of rows specified in <i>Output batch size</i>. If enabled, output messages resulting from queries contain the standard set of batching attributes, except for the optional <code>message.batchCount</code>. Default: false

Title	ID	Type	Description
Output Batch Size	outputBatchSize	integer	The number of rows that should be placed in each output message when executing queries. This value is used only when <i>Output in batches</i> is true and it must be an integer greater than zero. Default: 10
Init Statements	initStatements	string	The SQL code to be executed when the operator is initialized. The result of running these statements is not written to any ports. The given statements are executed after <i>Table initialization</i>. Placing a HANA Client operator inside a group with multiplicity larger than 1 causes these initialization statements to be executed multiple times. Default: ""
Prepared Statements	preparedStatements	array	A JSON array of SQL statements. Those statements are prepared during operator initialization and can be invoked using the <code>hana.preparedStatement</code> message attribute for port <code>data</code>. For efficient insertion, prefer <code>BULK INSERT</code> over regular <code>INSERT</code>, because the latter inserts rows one by one whereas the former batches them to reduce network usage. Note that <code>BULK INSERT</code> statements without arguments (question-mark placeholders) produce no effect. Default: []

Title	ID	Type	Description
Table Name	tableName	string	The name of the table to insert data into. If empty, port data can receive only messages with the <code>hana.preparedStatement</code> attribute. Refer to the Case Sensitivity section below for applicable naming rules. Default: ""
Table Columns	tableColumns	array	An array describing the columns of the table identified by Table name. If empty, port data can receive only messages with the <code>hana.preparedStatement</code> attribute. Refer to the Case Sensitivity section below for applicable naming rules. The size, precision and scale fields are used only if Table Initialization is not <code>None</code>. Default: []

Title	ID	Type	Description
Input Format	inputFormat	string	Mandatory. The format used to parse the data received at the data port. Accepted values are: • CSV: input is broken into records using CSV record delimiter; records are broken into fields using CSV field delimiter. Each field is parsed according to the type defined for its column in Table columns. • JSON: input must come as a JSON array. Each of its elements can be either: ◦ a JSON array representing a table row; its elements must be in the same order as Table columns; ◦ a JSON object whose keys are the column names in Table columns; spare keys are ignored. When using JSON, each column value is treated in its JSON-native type, unconverted. Default: "CSV"

Title	ID	Type	Description
CSV Mode	csvMode	string	Specifies how the operator should interpret the CSV data that arrives at the data port for insertion (without the <code>hana.preparedStatement</code> attribute). Accepted values: • Batch: Each chunk from the input is parsed as a self-contained CSV string. The records are inserted before reading the next input from data. • Stream: Each chunk is considered as the next piece of a single stream, so each individual input does not need to be complete. When <i>CSV bulk size records</i> are buffered (or <i>CSV bulk timeout</i> milliseconds go by without any new records), they are inserted into the table in a single bulk and an output message is sent to <code>result</code> (if applicable). When the graph stops, the stream is ended, so all buffered records are inserted, even if fewer than <i>CSV bulk size</i>, and output is not produced. i Note The end of the stream may truncate the last record. So if it was not complete (had fewer fields than required), it is discarded, the error is logged, and the pending records that came before are inserted as usual. Default: Batch

Title	ID	Type	Description
CSV Bulk Size	csvBulkSize	int	When in CSV stream mode, this property determines how many CSV records should be accumulated to be inserted in a single bulk. Default: 100
CSV Bulk Timeout	csvBulkTimeoutMs	int	When in CSV stream mode, if this value is positive and no new record was received for this duration, all buffered records are inserted as a bulk. Default: 10000
CSV Empty Field Value	csvEmptyFieldValue	string	When input is in CSV format, this property specifies how empty fields are interpreted for string-like columns: VARCHAR, ALPHANUM, and SHORTTEXT. Accepted values are "NULL" or "Empty string". For all other column types, empty CSV fields are converted to the NULL value. Default: "Empty string"
CSV Record Delimiter	csvBreak	string	The character used as record delimiter in CSV input. If set to "\n", both LF and CRLF are accepted. Default: "\n"
CSV Field Delimiter	csvComma	string	The character used as field delimiter in CSV input. Default: ","
CSV Quote Character	csvQuote	string	The character used as quote. Default: ""

Title	ID	Type	Description															
CSV Lazy Quotes	csvLazyQuotes	bool	If true, allows fields to contain unescaped quotes. With this option, your field need only be quoted if its value starts with a literal quote. Single-field examples and how to encode them with and without lazy quotes <table><thead><tr><th>Desired Value</th><th>CSV</th><th>CSV (lazy quotes)</th></tr></thead><tbody><tr><td>ab"</td><td>"ab"""</td><td>ab"</td></tr><tr><td>a"b</td><td>"a""b"</td><td>a"b</td></tr><tr><td>"ab</td><td>""""ab"</td><td>""ab"</td></tr><tr><td>"ab"</td><td>""""ab""""</td><td>""ab""""</td></tr></tbody></table> In the last example with lazy quotes, the second literal quote has to be escaped lest it is regarded as a close-quote due to it being at the end of a quoted field. Default: true	Desired Value	CSV	CSV (lazy quotes)	ab"	"ab"""	ab"	a"b	"a""b"	a"b	"ab	""""ab"	""ab"	"ab"	""""ab""""	""ab""""
Desired Value	CSV	CSV (lazy quotes)																
ab"	"ab"""	ab"																
a"b	"a""b"	a"b																
"ab	""""ab"	""ab"																
"ab"	""""ab""""	""ab""""																
CSV Comment Character	csvComment	string	The character used as comment indicator. When this configuration is non-empty, lines starting with this character are ignored. Default: ""															

Title	ID	Type	Description
CSV Header	csvHeader	string	Determines whether CSV contains a header and how to use it. The header is regarded as being the first record of each input (in <code>batch</code> mode) or of the entire lifetime (in <code>stream</code> mode). Accepted values are: • <code>None</code>: The first record is regular data, not a header. • <code>Ignore</code>: The first record is a header that should be discarded. • <code>Use as schema</code>: The first record is a header from which to extract column names. These names are matched against the table schema as reported by HANA so that spare columns in the CSV are ignored and missing ones will have a <code>NULL</code> value. In this mode, Table columns is not used for insertion (but may still be required by the Table initialization option). Default: "None"
Insert Mode	insertMode	string	Mandatory. Specifies whether the <code>INSERT</code> or the <code>UPSERT</code> command should be used for inserting data. When using <code>UPSERT</code>, the target table must have a primary key. For other uses of <code>UPSERT</code>, such as <code>WHERE</code> clauses, use Prepared statements. Default: "INSERT"

Title	ID	Type	Description
Table Initialization	initTable	string	Mandatory. Determines whether and how the table should be initialized. It is executed before the <i>Init statements</i>. If different than <i>None</i>, it ensures the schema exists (if explicitly given in <i>Table name</i>) and requires parameter <i>Table name</i> and <i>Table columns</i>. Accepted values are: • <i>None</i>: no table initialization is done. • <i>Create</i>: ensures table exists. • <i>Truncate</i>: ensures table exists and is empty, truncating the table. • <i>Delete Rows</i>: ensures table exists and is empty, deleting all rows. • <i>Drop (Cascade)</i>: ensures table exists and is empty, dropping table with <i>cascade</i> option. • <i>Drop (Restrict)</i>: ensures table exists and is empty, dropping table with <i>restrict</i> option. Default: "None"
Table Type	tableType	string	If <i>Table initialization</i> is not "None", this option determines whether to create a row or column table. Default: "Column"

Title	ID	Type	Description
Decimal Output Format	decimalOutputFormat	string	Determines how the operator should output the values <code>DECIMAL</code>- and <code>SMALLDECIMAL</code>-type columns. Accepted values are: • <code>Fraction (string)</code>: Decimals are the form <code><numerator>/<denominator></code> in a string. • <code>Floating-point</code>: Decimals are converted to double precision floating-point numbers. The resulting number may not represent the exact value of the division. Default: "Fraction (string)"
Transaction Level	transactionLevel	string	When running SQL statements with the <code>sql</code> input, this option determines how the operator groups statements into database transactions. Accepted values: • <code>Per statement</code>: a separate transaction is used for each statement. • <code>Per input</code>: a separate transaction is used for each input, and each transaction encompasses all statements in that input. Default: "Per statement"

Title	ID	Type	Description
Network Batch Size (rows)	networkBatchSize	int	The number of rows to fetch from the server in each batch when executing queries. This value can be adjusted to affect the granularity of network requests the operator performs. Default: 512
Connection Timeout (ms)	connectionTimeoutInMs	int	The number of milliseconds to wait for in each network operation: • establishing the connection • receiving query results • sending data All communication with the server is broken down into packets of at most 4KiB. The timeout applies to the reading and writing of each such packet. The value zero disables the timeout and allows operations to run indefinitely. Default: 50000

Title	ID	Type	Description
Terminate on Error	terminateOnError	bool	A flag that indicates whether or not the graph should stop if this operator encounters an error. If set to <code>false</code>, an output may represent an error, in which case its attribute <code>message.error</code> is <code>true</code> and its body contains a string describing the error. i Note This setting does not apply to errors during initialization, such as compiling <i>Prepared statements</i> or executing the <i>Init statements</i>. In such cases, errors always cause the graph to fail. Default: true

Input

Input	Type	Description
sql	message	Accepts SQL code in a message's body to be executed by the SAP HANA instance.
data	message	Accepts CSV or JSON data in a message's body. If the attribute <code>hana.preparedStatement</code> is present, its value is used to identify a prepared statement to which this data is submitted (see <i>Executing prepared statements</i> for the accepted data format). Otherwise, data is inserted into the table identified by <i>Table name</i> .

Output

Output	Type	Description
result	message	A message containing the result of an operation. The original input message attributes are preserved in the output, except when in CSV stream mode, because the output is decoupled from the input. If the operation retrieved rows, they are in the message body as an array of objects. Each key in an object is the uppercase name of the corresponding table column.

Supported Types

Insertion

The following data types are supported by this operator when using the `data` port:

- Numeric Types
 - TINYINT
 - SMALLINT
 - INTEGER
 - BIGINT
 - FLOAT
 - DOUBLE
 - DECIMAL and SMALLDECIMAL. These values must be represented as:
 - a number in decimal notation (for example, 4.12, 23); or
 - a fraction of the form `<numerator>/denominator>` (for example, 43/10).In JSON, they can be either a string in one of the above formats (for example, `[[1.3], ["2.1"], ["3/4"]]` is a valid input to insert three rows into a decimal-type column.
- Boolean Type:
 - BOOLEAN
- Character String Types:
 - VARCHAR
 - NVARCHAR
- Datetime Types: the following types are supported, along with HANA's default format for each of them:
 - DATE: YYYY-MM-DD (for example, '1957-06-13')
 - TIME: HH24:MI:SS (for example, '23:32:18')
 - TIMESTAMP: YYYY-MM-DD HH24:MI:SS.FF7 (for example, '1957-06-13 23:32:18.8261933')
 - SECONDDATE: YYYY-MM-DD HH24:MI:SS (for example, '1957-06-13 23:32:18')

Querying

When a query is sent to port `sql`, the type of each value in the output matches the table below.

Column Type	Output Type
TINYINT, SMALLINT, INTEGER, BIGINT	int64
FLOAT, REAL, DOUBLE	float64
DECIMAL, SMALLDECIMAL	See Decimal output format
ALPHANUM, SHORTTEXT, VARCHAR, NVARCHAR	string
SECONDDATE	string ("YYYY-mm-dd HH:MM:SS")
TIMESTAMP	string ("YYYY-mm-dd HH:MM:SS.NNNNNNN")
DATE	string ("YYYY-mm-dd")
TIME	string ("HH:MM:SS")
BOOLEAN	int64 (0 or 1)
VARBINARY, TEXT, BLOB, CLOB, NCLOB	[]byte

NULL values are always output as `nil`, regardless of the column type.

Note

Due to internal technical limitations, `CALL` statements that have output parameters do not yet obey the table above, except for integer, floating-point, and boolean types.

Case Sensitivity

Identifiers in SAP HANA are case-insensitive by default, so they are implicitly capitalized. When written in double-quotes, however, they become case-sensitive and must always be spelled exactly the same as when they were created. When using this operator, you must take case sensitivity into account: If a schema, table, or column was created using quotes, then its name in the operator's configuration must also be in quotes and in the same case.

The snippet below shows some examples and explanations on case sensitivity:

```
-- Because we're not using quotes, mySchema will be capitalized: MYSCHEMA.
CREATE SCHEMA mySchema;
-- myTable, B and c are all case-sensitive.
CREATE TABLE MYSCHEMA."myTable" (A INTEGER, "B" VARCHAR(1), "c" TIMESTAMP);
-- This statement will run successfully because b is not in quotes, so it
-- will be capitalized, becoming B, which is the correct name of the column.
INSERT INTO myschema."myTable" (a, b, "c") VALUES (0, 'x', NOW());
-- This statement will fail because c has to be quoted and lowercase.
INSERT INTO myschema."myTable" (a, b, c) VALUES (1, 'y', NOW());
```

2.2.6 HANA Monitor

This operator watches a HANA table and outputs newly inserted rows as a message.

It works by creating a trigger on the table to copy each inserted row into a temporary table. The temporary table is periodically queried and cleared so that it contains the new rows since the last poll. No output is produced if there were no new rows since the last query.

Configuration Parameters

Name	ID	Type	Description
Connection	Connection	object	Mandatory. The object containing the connection parameters to a HANA instance.
			i Note it may be necessary to enable TLS in your connection configuration. Error messages such as <code>i/o timeout</code>, <code>connection reset by peer</code> and <code>EOF</code> may indicate that a network firewall is dropping insecure packets When TLS is enabled, the operator uses the "Validate hostname in certificate" (<code>hostnameInCertificate</code>) connection property for the TLS Server Name extension. If this property is empty, the operator falls back to <code>host</code> , which must be a hostname and not an IP address.
Schema name	Schema name	string	Mandatory. The schema of the monitored table. Default: "

Name	ID	Type	Description
Table name	Table name	string	Mandatory. The table to be monitored. Default: "
Table columns	Table columns	string	Mandatory. A comma-separated list of column names and types. Sized types (such as VARCHAR and FLOAT) must include their sizes in parentheses. For example: <pre>ID INTEGER, NAME VARCHAR (64) , WAGE DECIMAL (11 , 2)</pre> Default: "
Poll period	Poll period	int	Mandatory. The time in milliseconds on which the temporary table will be polled for new rows. Default: 1000
Connection timeout	Connection timeout	int	The number of milliseconds to wait for when connecting to HANA before timing out. Default: 10000

Input

None.

Output

Output	Type	Description
outResult	message	A message whose body contains the new rows as an array of objects. Each key in an object is the uppercase name of the corresponding table column.

2.2.7 HTTP Client 2

This operator performs HTTP requests.

Note

OAuth2 authentication is not supported.

Operations

The client is capable of three main operations:

- Poll: if *Polling Enabled* is set to **True**, the operator sends GET requests to the server specified in *Poll Connection*. You can control the interval between requests with the *Polling Period* property.
- Post: data received on the in port is sent as POST requests to the server specified in *Post Connection*.
- Custom requests: each message received on the `inRequest` port specifies one request to be made.

The response to any of these operations is sent to the `outResponse` port, regardless of its status code. If this port is not connected, a status code outside the 2×× range raises an error. When the status code is in this range, the body of the response is also sent to the out port, if connected.

Configuration Parameters

Name	ID	Type	Description
Poll Period (ms)	<code>pollingPeriodInMs</code>	int	The period in milliseconds between requests of the poll operation.
Post Connection	<code>postConnection</code>	object	Connection information for the post operation. If a gateway is associated to this connection, it takes precedence over the <i>Proxy</i> setting.
Retry Attempts	<code>numRetryAttempts</code>	int	The number of times to retry a failed request. This applies to all three operations. Default: 0

Name	ID	Type	Description
Custom Request Connection	customConnection	object	An optional connection to be used by port inRequest. If a gateway is associated to this connection, it takes precedence over the Proxy setting.
Proxy	proxy	string	A URL to be used as proxy server. If empty, the system's configuration is used as fall-back (environment variables HTTP_PROXY, http_proxy, HTTPS_PROXY and https_proxy). This configuration applies to the operations that don't have a gateway associated to their connection. Default: ""
Request Timeout (ms)	requestTimeoutInMs	int	The number of milliseconds to wait for a response before timing out. This applies to all three operations. Default: 5000
Retry Period (ms)	retryPeriodInMs	int	The number of milliseconds to wait between consecutive retry attempts. This applies to all three operations. Default: 0

Input

Input	Type	Description
in	blob	The data that is sent in a POST request to Post Connection. An error is raised if data is received on this port and the connection is not set. The Content-Type header is automatically set using the MIME Sniffing algorithm.

Input	Type	Description
<code>inRequest</code>	message	A message specifying a custom HTTP request to be sent. The message must contain attribute <code>http.method</code> (the HTTP method for the request). If attribute <code>http.url</code> is not provided, Custom request Connection is used as fallback. HTTP Header Fields: - You can set any RFC 7230-compliant HTTP header field by prepending its name with <code>http.</code> as a message attribute. For example, when creating a message in a JavaScript operator: <pre>message.Attributes["http.Content-Type"] = "text/json";</pre> The HTTP Client automatically converts this to <pre>Content-Type: text/json</pre> - Fields <code>Content-Length</code>, <code>Transfer-Encoding</code>, and <code>Connection</code> are added automatically, as needed. - Field names are case-insensitive (RFC 7230, section 3.2).

Output

Output	Type	Description
<code>out</code>	blob	The output data in JSON format.

Output	Type	Description
outResponse	message	The complete response given by the remote peer. The message will contain the following attributes: • <code>http.status</code> (type string): the returned status (for example, 200 OK). • <code>http.statusCode</code> (type int): the code of the returned status (for example, 200). • <code>http.url</code> (type string): the URL of the remote peer. • <code>http.method</code> (type string): the HTTP method (for example GET). • All HTTP header fields sent by the server, prepended by <code>http.</code>, for example, field <code>Content-Type</code> becomes message attribute <code>http.Content-Type</code>.

Error Handling

In some cases, there can be expected errors that should not cause the graph to stop. Observe the following behavior:

- If port `outResponse` is connected:
 - If an error occurred and no response was received, an error is thrown.
 - If a response was received, it will be written to this port regardless of the contained HTTP status.
 - The user is free to decide on the appropriate treatment for each HTTP status by reading it from the `http.status` and `http.statusCode` message attributes.
- If port `outResponse` is not connected, then this operator throws an error itself if the request failed or if the status in the remote peer's response is anything other than `success (2xx)`.
- Port `out` does not affect the above error handling policy, and the body of a response is written to this port if the status reports `success (2xx)`.

TLS Certificates

Due to limited filesystem manipulation capabilities, the HTTP Client automatically loads and uses any certificate files in its directory with a `.cert` extension. To upload a certificate, go to the [Repository](#) tab, browse to the [HTTP Client](#)'s editor page and click the [Upload Auxiliary File](#) button.

2.2.8 HTTP Server

This operator allows a graph to accept HTTP requests on user-defined endpoints (routes) and respond to them based on messages produced on the fly by other operators in the graph.

Output ports must be created when designing the graph to accommodate the desired route multiplexing.

The sample graph `com.sap.demo.http.server` demonstrates some of the usage of this operator.

All configured routes will exist under a prefix that depends on the *graph handle* and the *operator ID*:

```
/service/v1/graphs/<graph-handle>/operator/<operator-id>/
```

This means that the URL will automatically be unique for every HTTP server in the graph as well as for every running instance of it, so the graph may run multiple times simultaneously. You can find the graph handle from the *Status* panel, and the operator ID from the Configuration panel.

Configuration Parameters

Title	ID	Type	Description
Routes	<code>routes</code>	array	The list of routes the operator accepts. See Route Specification below for details. Default: []

Title	ID	Type	Description
Max Simultaneous Requests	<code>maxRequests</code>	int	The maximum number of pending requests this operator can hold. A request is considered pending from the moment it is sent to the output port until a response is received from the input and sent to the client. This limit applies to requests globally, without taking into account whether they came from the same client or not. When this number is reached, all requests will be automatically responded to with status 503 Service Unavailable and the error message "simultaneous request limit reached" until at least one response arrives on the input for a pending request. Default: 1

Input

Input	Type	Description
response	message	A message describing how to respond to a pending request. The attribute <code>message.request.id</code> must be present and hold the value that was sent to the output port when the request was received.

The body of the message, if present, will be used as the response payload. Objects and arrays will be encoded as JSON.

The HTTP status can be set with the `http.status` attribute (an integer) and defaults to 200 OK when the attribute is not present.

Output

By default, this operator comes without any outputs. Users must add as many output ports as they wish to multiplex the routes they configure. All outputs must be of type message. Each port will be used to output the requests for the route(s) associated to it in the route's configuration.

If the HTTP request has a payload, it will become the output message body.

The output message will have the following attributes:

Output	Description
<code>message.request.id</code>	A unique ID generated by the operator for this request. To respond this request, an input message must arrive at port <code>response</code> containing this ID.
<code>http.method</code>	The request's HTTP method. This will necessarily be one of the route's <code>methods</code> .
<code>http.path</code>	The complete path of the request, including the common prefix described in <code>basePath</code> .
<code>http.route</code>	The route's own path, as set in the <code>routes</code> configuration.
<code>http.peer</code>	The address of the client that made the request.
<code>http.vars</code>	An object mapping URL variable names to their values.

Route Specification

Each element in the route's array is an object with the following properties:

- **Path:** the route's path, which will be accessible under a prefix as described in `stableURL`. No two routes can have the same path.
- **Outputs:** an object that maps each supported HTTP method (GET, POST, DELETE, and PUT) to the name of the output port to which the operator will send the requests received for this endpoint with that method. A blank port name means the endpoint does not accept requests with that method, so those will be automatically replied to with status 405 Method Not Allowed.

If the port does not exist, an error will be raised during initialization.

If the port exists but is not connected (so there is no operator to generate a response), all requests to it will be automatically responded to with status 204 No Content.

Route Variables

Route paths can have variables of two kinds: regular variables and catch-all variables. In both cases, the concrete value can be found in a message attribute named `http.vars`. This attribute is a map whose keys are the

variables' names. The value for a variable is always a string. In a Go operator, the `http.vars` attribute has to be type-asserted to `map[string]string`:

```
func OnInput(val interface{}) {
    msg := val.(map[string]interface{})
    attributes := msg["Attributes"].(map[string]interface{})
    httpVars := attributes["http.vars"].(map[string]string)
    // ...
}
```

Regular variables take the form `:myvar` in the route's path. Each variable of this kind matches a single path segment.

For example, if a route has the path `/people/:name` and a request is made to `/people/adam`, a JavaScript operator can catch the output from HTTP Server and retrieve the requested person's name:

```
function onInput(ctx, msg) {
    console.log(msg.Attributes['http.vars']['name']); // Prints 'adam'
}
```

Catch-all variables take the form `*myvar` in the route's path. They must come at the end of the route's path and will match everything to the end of the requested path. They can also be empty. Retrieving its concrete value from the message attribute is done just like with regular variables.

Cross-Site Request Forgery Protection

HTTP requests using methods GET, HEAD and OPTIONS are not considered vulnerable to CSRF (cross-site request forgery) and therefore no CSRF protection is in place.

For all other HTTP methods CSRF protection is active and needs to be bypassed by one of two ways:

- Set the HTTP header X-Requested-With to a non-empty value (this is what an AJAX request would do without losing the ability to authenticate via session cookie).
- Authenticate with Authorization: Bearer <token> header (this is what a client library would likely use).

2.2.9 Kafka Consumer 2

The Kafka Consumer operator works as a Kafka client that consumes records or messages from a Kafka cluster. It is compatible with Kafka versions 0.9.x and later.

Configuration Parameters

Name	ID	Type	Description
Kafka Version	kafkaVersion	string	Mandatory. The version of Kafka server being used. Default: "1.0.0"
Connection Type	connectionType	string	Manual: gets connection information from each property. Connection management: uses connection property for connection information. Default: "manual"
Kafka Consumer Connection	connection	object	Information about the connection for the service.
Topics	topics	string	Name of the topics from which messages are read and sent to the pipeline (consumed). Topic names are separated by , . Default: "test_topic"

Name	ID	Type	Description
Auto Set Group ID	autoGroupId	bool	When true, the group ID is set according to the multiplicity index of the operator. For instance, when the operator is inside a group with multiplicity = 3, each node consumes from group-0,group-1 and group-2, where group is the value set in groupId config and - * is the multiplicity index. Default: false
Auto Commit	autoCommit	bool	Whether to automatically commit the message (mark message as processed). Default: true
Max Message (bytes)	maxMessageBytes	int	The maximum number of message bytes to fetch from the broker in a single request. Default: 0
Max Wait Time	maxWaitTime	int	The maximum amount of time in milliseconds that the broker waits for the consumer. Default: 500
Retry Attempts	numRetryAttempts	int	The number of attempts to retry. Default: 0
Offset	offset	string	The initial offset to use if no offset was previously committed. Default: "newest"
Terminate on Error	terminateOnError	bool	Whether to terminate the consumer in case the Kafka Consumer throws an error. Default: true

Connection Parameters

Name	ID	Type	Description
Configuration Type	configurationType	string	Type of connection: Manual (user input) or retrieved by the Connection Management Service. Default: "Configuration manager"
Connection ID	connectionID	string	The ID of the connection information to be retrieved from the Connection Management Service. Default: ""

Name	ID	Type	Description
Kafka Consumer Connection	According to connection property	object	All the connection properties for manual input: <i>Brokers</i> (ID brokers, type string). Mandatory. The list of nodes in the cluster following the format 'host:port' and separated by commas. Default: "localhost:9092" <i>Group ID</i> (ID groupId, type string). Mandatory. All information for a consumer that is part of the group. Default: "test_group" <i>Use TLS</i> (ID useTLS, type bool). Flag that enables TLS/SSL authentication. Default: false <i>Insecure Skip Verify</i> (ID insecureSkipVerify, type bool). If a client verifies the server's certificate chain and host name. Default: false <i>Use Certificate Authority</i> (ID useCA, type bool). If this property is true, the TLS CA file is requested. i Note only enabled when 'Use TLS' is true. Default: true <i>Certificate Authority Path</i> (ID tlsCAFile, type string). Server's certificate authority file path (extension has to be .pem). Default: "" <i>Client Certificate Path</i> (ID tlsClientCertFile, type string). Client certificate file that is loaded with public or private key pair if applicable (exten-

Name	ID	Type	Description
			sion has to be .pem). Default: "" <i>Client Key Path</i> (ID tlsClientKeyFile, type string). Client key file that is loaded with client certificate file if applicable (extension has to be .pem). i Note the supported encryption key types are: rsa, ecdsa and ed25519. Default: "" <i>Authentication</i> (ID authenticate, type bool). Manual configuration: flag that enables SASL authentication. Default: false <i>Username</i> (ID username, type string). Username if using Kafka's SASL/PLAIN authentication. Default: "" <i>Password</i> (ID password, type string). Password if using Kafka's SASL/PLAIN authentication. Default: "" <i>SASL Authentication</i> (ID saslAuthentication, type string). Authentication mechanism to be used. i Note PLAIN, SCRAM-256, SCRAM-512, and GSSAPI for Kerberos are supported (the property is considered only if username and password are set, except for GSSAPI).

Name	ID	Type	Description
			Default: "PLAIN"
			<i>Kafka Kerberos Service Name</i> (ID serviceNameGSSAPI, type string). Service name defined for the kerberized Kafka server. Default: "kafka"
			<i>Kafka Kerberos Realm</i> (ID realmGSSAPI, type string). Realm defined for the kerberized Kafka server. Default: ""
			<i>Authentication Type for Kafka/GSSAPI</i> (ID authTypeGSSAPI, type string). Authentication type for Kafka/GSSAPI, when "User/password" is used, username and password must be set, when "Keytab file authentication" is used, keytab path must be set. Default: "User/Password Authentication"
			<i>Kerberos Config Path</i> (ID kerberosConfigPath, type string). Kerberos configuration path (krb5.conf file). Default: ""
			<i>Keytab File Path</i> (ID keytabPath, type string). Path for keytab file for Kafka client. Default: ""

Input

Input	Type	Description
markoffsets	message	An offset (record) of the provided topic or partition as processed (committed) the header message.commit.token must be set.

Output

Output	Type	Description
message	message	The stream of records consumed.
event	message	The stream of events produced by the consumer such as consumer rebalance notifications.

2.2.10 Kafka Producer

The Kafka Producer operator allows applications to send records or messages to topics on a Kafka cluster. It supports Kafka versions 0.8 and later.

Configuration Parameters

Name	ID	Type	Description
Kafka Version	kafkaVersion	string	Version of Kafka server being used. Default: "1.0.0"
Connection Type	connectionType	string	Manual: gets connection information from each property. Connection management: uses connection property for connection information. Default: "manual"
Kafka Producer Connection	connection	object	Holds information about the connection for the service.
Topic	topic	string	Mandatory. The category name to which records are subscribed. A new topic name can be set using the <code>kafka.topic</code> message attribute for port Message. Default: "test_topic"

Name	ID	Type	Description
Manual Partitioning	manualPartitioning	bool	Specifies whether the partition is configured manually, it means that the user chooses the partition of the message. Default: "false"
Key	key	string	The key to be included in the record. A new key can also be set using the <code>kafka.key</code> message attribute for port messages when the property <code>manualPartitioning</code> is false. Default: ""
Retry Attempts	retryAttempts	int	The number of attempts to retry. Default: 0
Retry Period (ms)	retryPeriodInMs	int	The retry waiting time in milliseconds. Default: 0
Partition	partition	int	The partition count. Default: 0
Async	async	bool	Specifies whether the producer should work on asynchronous mode. Default: false
Max Message (bytes)	maxMessageInBytes	int	The maximum permitted size of a message. Limited to at most 100 MB, which is the largest request the broker will attempt to process. Default: 1000000
Max Message Batch	maxMessageBatch	int	The maximum number of messages the producer will send in a single broker request. Default: 0

Name	ID	Type	Description
Request Timeout (ms)	timeout	int	The maximum duration the broker will wait the receipt of the number of required acks (defaults to 10 seconds). Default: 30000

Connection Parameters

Name	ID	Type	Description
Configuration Type	configurationType	string	Type of connection information that is used: Manual (user input) or retrieved by the Connection Management Service. Default: "Configuration manager"
Connection ID	connectionID	string	The ID of the connection information to be retrieved from the Connection Management Service. Default: ""

Name	ID	Type	Description
Kafka Producer connection	According to connection properties.	object	All the connection properties for manual input: <i>Brokers</i> (ID brokers, type string). Mandatory. The list of nodes in the cluster following the format 'host:port' and separated by commas. Default: "localhost:9092" <i>Use TLS</i> (ID useTLS, type bool). Flag that enables using TLS/SSL authentication. Default: false <i>Insecure Skip Verify</i> (ID insecureSkipVerify, type bool). If a client verifies the server's certificate chain and host name. Default: false <i>Use Certificate Authority</i> (ID useCA, type bool). If this property is true, the TLS CA file is requested. i Note only enabled when 'Use TLS' is true. Default: true <i>Certificate Authority Path</i> (ID tlsCAFile, type string). Server's certificate authority file path (extension has to be .pem). Default: "" <i>Client Certificate Path</i> (ID tlsClientCertFile, type string). Client certificate file that is loaded with public or private key pair if applicable (extension has to be .pem). Default: "" <i>Client Key Path</i> (ID tlsClientKeyFile, type string). Client key file that will be loaded with client certificate file if

Name	ID	Type	Description
			applicable (extension has to be .pem). i Note the supported encryption key types are: rsa, ecdsa and ed25519. Default: "" <i>Authentication</i> (ID authenticate, type bool). Manual configuration: flag that enables SASL authentication. Default: false <i>Username</i> (ID username, type string). Username if using Kafka's SASL/PLAIN authentication. Default: "" <i>Password</i> (ID password, type string). Password if using Kafka's SASL/PLAIN authentication. Default: "" <i>SASL Authentication</i> (ID saslAuthentication, type string). Authentication mechanism to be used. i Note PLAIN, SCRAM-256, and SCRAM-512 are supported (the property is considered only if username and password are set). Default: "PLAIN" <i>Kafka Kerberos Service Name</i> (ID serviceNameGSSAPI, type string). Service name defined for the kerberized Kafka server. Default: "kafka" <i>Kafka Kerberos Realm</i> (ID realmGSSAPI, type string).

Name	ID	Type	Description
			Realm defined for the kerber-ized Kafka server. Default: "" Authentication Type for Kafka/GSSAPI (ID authTypeGSSAPI, type string). Authentication type for Kafka/GSSAPI, when \"User/password\" is used, username and password must be set, when \"Keytab file authentication\" is used, keytab path must be set. Default: \"User/Password Authentication\" Kerberos Config Path (ID kerberosConfigPath, type string). Kerberos configuration path (krb5.conf file). Default: "" Keytab File Path (ID keytabPath, type string). Path for keytab file for Kafka client. Default: ""

Input

Input	Type	Description
message	message	The stream of records to publish.

Output

Output	Type	Description
markoffsets	message	Message containing the offsets marked as part of the current transaction and set of headers.

2.2.11 MQTT Consumer

The MQTT Consumer operator subscribes to MQTT topics, consumes its messages, and outputs them.

Configuration Parameters

Name	ID	Type	Description
MQTT brokers	mqttbroker	string	List of MQTT addresses, each separated by a comma. Each broker format should be <code>scheme://host:port</code> , where <code>scheme</code> is one of the following: • <code>tcp</code> • <code>ssl</code> • <code>ws</code> Default: "tcp://localhost:1883"
Topic	topic	string	The topic name to which consumer will subscribe. Default: "testmqtt/test1"
Clean session	cleanSession	bool	Will set the "clean session" flag in the connect message. This flag indicates that no messages saved by the broker for this client should be delivered. Default: true
Insecure skip verify	insecureSkipVerify	bool	Controls whether a client verifies the server's certificate chain and host name. Default: false
Client ID	mqttClientID	string	The client identifier. Default: "mqttConsumer"

Name	ID	Type	Description
Retry attempts	numRetryAttempts	int	The number of attempts to retry. Default: 0
Retry period (ms)	retryPeriodInMs	int	The time period between each retry attempt in milliseconds. Default: 1000
TLS certificate file	tlsCertificateFile	string	The TLS certificate file path. Default: ""
TLS key file	tlsKeyFile	string	The TLS key file path. Default: ""
Username	username	string	The MQTT optional user-name Default: ""
Password	password	string	The MQTT optional password Default: ""
Explicit request mode	explicitRequestMode	bool	When set to <code>true</code> , this property changes the way that consumer deals with messages. It will wait for an input of <code>singlemessagerequest</code> port to connect a client, consume only one message from it, and then disconnect the client. This cycle happens whenever a signal comes through <code>singlemessagerequest</code> . Default: <code>false</code>
Timeout (s)	timeout	int	In case that explicit request mode is being used, this property is the time to wait to receive a message. Default: 15

Input

Input	Type	Description
singlemessagerequest	message	This input port should be used when <code>explicitRequestMode</code> is set to true. The input message can be empty or include the attribute <code>mqtt.topic</code> . After receiving the signal, the MQTT client will connect, subscribe to topic, consume one message, and disconnect.

Output

Output	Type	Description
outmessage	message	The stream of records published.

2.2.12 MQTT Producer

The MQTT Producer operator receives MQTT messages from a port and publishes them in a topic.

Configuration Parameters

Name	ID	Type	Description
MQTT brokers	mqttBroker	string	List of MQTT addresses, each separated by a comma. Each broker format should be <code>scheme://host:port</code>, where <code>scheme</code> is one of the following: • <code>tcp</code> • <code>ssl</code> • <code>ws</code> Default: "tcp://localhost:1883"

Name	ID	Type	Description
Topics	topic	string	The category name to which records are published. Default: "testmqtt/test1"
Client ID	mqttClientID	string	The client identifier. A clientID must be set. Otherwise, the graph will not run and the ID must be a unique to avoid collision. Default: ""
Clean session	cleanSession	bool	Set the "clean session" flag in the connect message. This flag indicates that no messages saved by the broker for this client should be delivered. Default: true
Insecure skip verify	insecureSkipVerify	bool	Controls whether a client verifies the server's certificate chain and host name. Default: false
Retry attempts	retryAttempts	int	The number of attempts to retry. Default: 0
Retry period (ms)	retryPeriodInMs	int	The time period between each retry attempt in milliseconds. Default: 1000
Client connection timeout	timeout	int	Timeout to stabilize connection with client ID. Default: "500"
TLS certificate file	tlsCertificateFile	string	The TLS certificate file path. Default: ""
TLS key file	tlsKeyFile	string	The TLS key file path. Default: ""

Name	ID	Type	Description
Username	username	string	The MQTT optional user-name. Default: ""
Password	password	string	The MQTT optional password Default: ""
Retain message	retainMessage	bool	When set to <code>true</code> , the publisher will tell the broker to keep the last message on that topic after consumed. Default: false
Explicit message acknowledgement	acknowledgeMessage	bool	When set to <code>true</code> , the producer will reconnect every time it receives a new message from the input port. Then it publishes the message to the topics defined and disconnect, sending the status to the <code>status</code> output port. Default: false

Input

Input	Type	Description
inmessage	message	The stream of records to publish. If the message contains the attribute <code>mqtt.topic</code> the message will be published to this topic.

Output

Output	Type	Description
status	message	The status after publishing a message (used when <code>Explicit message acknowledgement</code> is true). When there is an error, the attribute <code>error</code> will have it.

2.2.13 NATS Consumer

The NATS Consumer operator consumes messages from NATS and forwards them to the output port.

Configuration Parameters

Name	ID	Type	Description
NATS URL	natsUrls	string	Mandatory. The NATS address. Default: "nats://localhost:4222"
Queue	queue	string	The queue name to form the messages queue group. For more information, check the NATS documentation. Default: ""
Subject	subject	string	Mandatory. The subject name to subscribe to. Default: "subject"
Number of Retry Attempts	numRetryAttempts	int	The number of attempts to retry. Default: 0

Name	ID	Type	Description
Retry Period (ms)	retryPeriodInMs	int	The time period for each re-try in milliseconds. Default: 0

Input

None.

Output

Output	Type	Description
message	message	A message with the body containing the data (as a blob) and with the attributes containing the <code>nats.subject</code> and <code>nats.reply</code> fields.

2.2.14 NATS Producer

The NATS Producer operator receives a message from the input port and publishes it into NATS.

Configuration Parameters

Name	ID	Type	Description
NATS URL	natsUrls	string	Mandatory. The NATS address. Default: "nats://localhost:4222"
Subject	subjects	string	Mandatory. The subject name to subscribe to. Default: "subject"

Name	ID	Type	Description
Number of Retry Attempts	numberRetryAttempts	int	The number of attempts to retry. Default: 0
Retry Period (ms)	retryPeriodInMs	int	The time period between each retry in milliseconds. Default: 0

Input

Input	Type	Description
message	message	Message with the content to be published on its body. The body type should either be blob (<code>\[\] byte</code>) or string.

Output

None.

2.2.15 OpenAPI Client

The OpenAPI client operator is a convenient client that is suitable for invoking services described in a OpenAPI document. However, its use is not limited to those services described in OpenAPI documentation, and it can be used to invoke arbitrary REST services.

This client is expected to be configured with some default parameters. Some of these properties can be customized per call when the corresponding headers of the input message are set accordingly.

In particular, parameters defined for the operation such as header, path, query, form, and file parameters can be set in the message headers using header names in form `openapi.type_params.name`, where `type` indicates one of the parameter types and `name` identifies the parameter name, respectively.

In other words, headers `openapi.header_params.name`, `openapi.path_params.name`, `openapi.query_params.name`, `openapi.form_params.name`, and `openapi.file_params.name` are used to represent the corresponding name parameter for the operation. Parameter values are interpreted as string for scalar values and set to the corresponding parameters of the request message except for file parameters. The value of a file parameter can be set either to the path to a file or to the content of a file.

For example, to assign file `/opt/pictures/dog12.png` or its content to file parameter `foto`, header `openapi.file_params.foto` can be set to `/opt/pictures/dog12.png`. Alternatively, this header can be set to the base64 representation of the content of this file while setting another parameter `openapi.file_params.foto#name` to its name `dog12.png` to indicate that the value of this file parameter is the content and not the file path. Parameters values for vector values are represented as arrays. All parameters except for the header parameters support vector values (for example, `["sunny", "day"]` for two values `"sunny"` and `"day"`).

In addition to the above parameters, if header `message.request_id` is set in the input message, this header will be included in the output message so that each response message can be correlated to its corresponding request message. Furthermore, the status code is set in `openapi.status_code` in the response message.

Recognized Message Headers

Header	Description
<code>message.request_id</code>	The ID for correlating a request and its response messages.
<code>openapi.header_params.x</code>	Header parameter x.
<code>openapi.path_params.x</code>	Path parameter x.
<code>openapi.query_params.x</code>	Query parameter x.
<code>openapi.form_params.x</code>	Form parameter x.
<code>openapi.file_params.x</code>	File parameter x, where its value may represent the path to the file or the base64 encoded content of the file.
<code>openapi.file_params.x#name</code>	File name parameter for file parameter x, if set, represents the file name and the value of its file parameter represents the base64 encoded content.
<code>openapi.path_pattern</code>	Overrides the default <code>pathPattern</code> parameter configured at the .
<code>openapi.method</code>	Overrides the default <code>method</code> parameter configured at the operator.
<code>openapi.produces</code>	Overrides the default <code>produces</code> parameter (for example, the accepted response content types) configured at the operator.
<code>openapi.consumes</code>	Overrides the default <code>consumes</code> parameter (for example, the request content type) configured at the operator.
<code>openapi.auth_scheme</code>	Overrides the default <code>authScheme</code> parameter configured at the operator.
<code>openapi.user</code>	Overrides the default <code>user</code> configured at the operator.
<code>openapi.password</code>	Overrides the default <code>password</code> configured at the operator.

Header	Description
openapi.api_key_name	Overrides the default apiKeyName configured at the operator.
openapi.api_key_value	Overrides the default apiKeyType configured at the operator.
openapi.status_code	Status code given in the response message.

Several authentication schemes are supported by the OpenAPI client operator. These include basic, APIKey, and OAuth authentication schemes. When OAuth is chosen and its oauth2Flow is set to accessCode or implicit, the user must use the UI option of the OpenAPI client operator to authorize the client instance so that it can retrieve an access token. In this case, the correct redirection URL for this client instance must be registered at the authorization provider. The redirection URL has the service path `/service/v1/runtime/internal/redirect`.

You can use this operator as is, on its own, by manually configuring its parameters and using a custom operator that forwards appropriate input messages to this operator. If you have an OpenAPI file, you can generate such custom operator and a demo graph using the `generateOpenAPIClient` REST operation of the engine. Refer to the API document (swagger.yaml) available at the installed Modeler instance for details.

See [OpenAPI Server \[page 116\]](#) for the server-side operator.

Configuration Parameters

Name	ID	Type	Description
Host	host	string	The service's host optionally including the port. Default: localhost:8090
Schemes	schemes	string	The transport protocols. Default: http
Base Path	basePath	string	The service's base path. Default: /service
Path Pattern	pathPattern	string	The path pattern with optional path variables. Overridden by header <code>openapi.path_pattern</code> . Default: ""

Name	ID	Type	Description
Method	method	string	The method name. Overridden by header open-api.method. Default: "GET"
Produces	produces	string	A comma-separated list of media types that can be produced by the operation. Overridden by header open-api.produces. Default: "application/json"
Consumes	consumes	string	A comma-separated list of media types that can be consumed by the operation. Overridden by header open-api.consumes. Default: ""
Include Response Headers	includeResponseHeaders	string	A comma-separated list of response header names that should be included in the output message. Overridden by header openapi.include_response_headers. Default: ""
Authentication Scheme	authScheme	string	The security scheme. The possible values are "", "basic", "apiKey", "oauth2". Overridden by header open-api.auth_scheme. Default: ""
User	user	string	The user name used by "basic" security scheme. Overridden by header open-api.user. Default: ""

Name	ID	Type	Description
Password	password	string	The password used by "basic" security scheme. Overridden by header openapi.user. Default: ""
API Key Name	apiKeyName	string	The API key name used by "apiKey" security scheme. Overridden by header openapi.api_key_name. Default: ""
API Key Type	apiKeyType	string	The API key type used by "apiKey" security scheme. The possible values are "header" or "query". Overridden by header openapi.api_key_type. Default: ""
API Key Value	apiKeyValue	string	The API key value used by "apiKey" security scheme. Overridden by header openapi.api_key_value. Default: ""
OAuth2 Flow	oauth2Flow	string	The flow used by "oauth2" security scheme. This value corresponds to one of the OAuth2 grant types. The possible values are "implicit," "password," "application," and "accessToken," which correspond to grant types "implicit grant," "password grant," "client credentials grant," and "authorization code grant," respectively. Default: ""
OAuth2 Scope	oauth2Scope	string	The scope used by "oauth2" security scheme. Default: ""

Name	ID	Type	Description
OAuth2 Token URL	oauth2TokenUrl	string	The token URL used by "oauth2" security scheme when the flow is set to "password," "application," or "accessCode." Default: ""
OAuth2 Authorization URL	oauth2AuthUrl	string	The authorization URL used by "oauth2" security scheme when the flow is set to "implicit" or "accessCode". Default: ""
OAuth2 Client ID	oauth2ClientId	string	The client ID used by "oauth2" security scheme". Default: ""
OAuth2 Client Secret	oauth2ClientSecret	string	The client secret used by "oauth2" security scheme". Default: ""
OAuth2 Default Token Expiration	oauth2DefaultTokenExpires	integer	The default token expiration time in seconds. This value is used only when the token returned by the authorization server does not specify the expiration time (<code>expires_in</code>). The value of 0 or less denotes no expiration.
OAuth2 Additional Headers	oauth2AdditionalHeaders	string	Optional headers added in a token request, given as <code>\{"header1": "value", ...\"}</code>. Default: ""
Use CSRF Token	useCsrfToken	bool	If set to true, the client automatically retrieves a csrf-token at its first request and uses it in its subsequent requests. Default: false
TLS Certificate	tlsCertificate	string	The file path to the TLS certificate file. Default: ""

Name	ID	Type	Description
TLS Key	tlsKey	string	The file path to the TLS private key file. Default: ""
TLS CA	tlsCa	string	The file path to the TLS certificate authority file. Default: ""
TLS Skip Verify	tlsSkipVerify	bool	If set to true, the client does not verify the server's certificate chain nor the host name. Default: false
Max Concurrency	maxConcurrency	integer	The maximum number of concurrent requests that are handled per operator instance. If the message order must be preserved, set this value to 1. Default: 8
Timeout	timeout	int	The timeout value in milliseconds to wait for a response. Default: 300000

Input

Input	Type	Description
in	message	Input message where parameters are set in the headers and the optional content in body.

Output

Output	Type	Description
out	message	Output message where parameters are set in the headers and the optional content in body.

2.2.16 OpenAPI Server

The OpenAPI server operator is a convenient server-side component that is suitable for providing services described in OpenAPI documentation or unspecified HTTP services. The component runs behind the web container of the hosting platform and inherits its security configuration. More concretely, this operator is configured to start its REST endpoint relative to the Pipeline engine's service path at `/openapi/service/basePath/`.

When a request matching this path arrives, an output message is generated and sent to the connected operator. This operator supports both request-response and one-way message exchange patterns (MEPs). Currently, this choice must be configured per operator. When the operator is running in the request-response mode, the caller waits for the response up to the specified timeout. If the response message is returned to this operator over its port `ret`, it is returned to the caller. Otherwise, an error is returned to the caller.

Note that the service resides not at `/openapi/service/basePath` but within `/openapi/service/basePath`. In other words, a request to `/openapi/service/basePath` will return an HTTP 301 response redirecting to `/openapi/service/basePath/`.

The output message generated by this operator depends on whether it is configured as the plain mode (for example, no OpenAPI document is associated) or the OpenAPI-driven mode (for example, an OpenAPI document is associated). In the plain mode, the headers and body of the incoming message will be directly transferred to the output message. In contrast, in the OpenAPI-driven mode, the matching operation to the incoming message will be determined and the corresponding parameters for that operation will be extracted. Subsequently, these operation specific parameters will be transferred to the output message.

See [OpenAPI Client \[page 109\]](#) for the client-side operator.

Configuration Parameters

Name	ID	Type	Description
Base Path	basePath	string	The service's base path. Default: /untitled

Name	ID	Type	Description
Timeout	timeout	int	The timeout value in milliseconds to wait for a response. Default: 300000
One-Way	oneway	bool	If set to true, this operator does not wait for a response and returns an HTTP 204 immediately. If set to false, this operator waits for a response up to the timeout value. Default: false

Name	ID	Type	Description
Swagger Specification	swaggerSpec	string	The optional OpenAPI document for this service endpoint. If this parameter is specified, a GET request to <code>/openapi/service/_basePath_/swagger.json</code> will return the OpenAPI spec document. (Note that this <code>_basePath_</code> value corresponds to the value of the above <code>basePath</code> parameter and not the value of the <code>basePath</code> property configured in the OpenAPI document.) Furthermore, the request message will be processed according to this OpenAPI document and the corresponding parameters will be extracted and injected into the output message. For the header names of the injected parameters, refer to the documentation for <code>openapi.client</code>. All the operations declared in the OpenAPI document are accessible. If this parameter is not specified, the incoming messages are processed in the generic mode. Default: ""
Web Socket	websocket	bool	If set to true, this operator's <code>swaggersocket</code> mode is enabled and the operator accepts requests over websocket and returns responses asynchronously. Default: false

Name	ID	Type	Description
Max Concurrency	maxConcurrency	integer	The maximum number of concurrent requests that are handled per operator instance. If the message order must be preserved, set this value to 1. Default: 8
Max Accepted	maxAccepted	integer	The maximum number of accepted requests at a time. Requests are accepted up to this number and being processed. If additional requests arrive while this number is reached, those requests are rejected. Default: 128

Input

Input	Type	Description
ret	message	Return message produced by processing the request message in the pipeline. This message must contain the original request ID attribute <code>message.request.id</code> from the request message. The default HTTP status code of the resulting HTTP response is 200, which can be overridden by setting attribute <code>openapi.status.code</code>. In addition, a technical error with HTTP 500 may be returned by setting attribute <code>message.response.error</code> with either an error text or the JSON representation of JavaScript error (for example, "my error" or <code>{"message": "my error", "name": "Error"}</code>).

Output

Output	Type	Description
out	message	Output message where parameters are set in the headers, and the optional content in body.

Cross-Site Request Forgery Protection

HTTP requests using methods GET, HEAD and OPTIONS are not considered vulnerable to CSRF (cross-site request forgery) and therefore no CSRF protection is in place.

For all other HTTP methods CSRF protection is active and needs to be bypassed by one of two ways:

- Set the HTTP header X-Requested-With to a non-empty value (this is what an AJAX request would do without losing the ability to authenticate via session cookie).
- Authenticate with Authorization: Bearer <token> header (this is what a client library would likely use).

Related Information

[Swagger/OpenAPI](#) ➡

2.2.17 Receive Email

Receives emails. The protocol used is IMAP Version 4rev1. The operator fetches emails automatically if the in port is not connected. When the port is connected, the operator fetches emails whenever a new entry arrives. The output is received as single messages, each related to a single email.

Configuration Parameters

Configuration Parameters

Name	ID	Type	Description
Maximum Entries	Maximum entries	integer	The maximum number of entries the system will fetch. If <code>maxEntries</code> is higher than the mailbox amount of emails, it will send only the emails contained in the mailbox. If <code>maxEntries</code> is smaller than the mailbox occupancy, it will send only the '<code>maxEntries</code>' newest items. This property is also considered when input port is not connected. Default: 10
Mailbox	Mailbox	string	Mandatory: The mailbox from which emails will be retrieved. This will be used only when in Polling mode. Default: "inbox"
Poll Period	Poll period	integer	The poll period (in Milliseconds). Meaning that every '<code>pollPeriod</code>' seconds, a new email fetch will be done. This will be used only when in Polling mode. Default: 1000

Name	ID	Type	Description
Only Output Unread Emails	Only output unread emails	boolean	When set to true, the system will output only emails that have not been previously read. Default: true

Note

Use `Connection Management` for defining connections over the manual option.

Connection Parameters

Parameter	Type	Description
connection	object	Information about connection details for the services.
configurationType	string	Which type of connection information will be used: Manual (user input), or retrieved by the Connection Management Service. Note It is recommended to choose <i>Configuration Manager</i> as the configuration type and select an existing <i>Connection ID</i>. The <i>Manual</i> configuration type must be used only for testing purposes. Default: ""
connectionID	string	The ID of the connection information to retrieve from the Connection Management Service. Default: ""

Parameter	Type	Description
connectionProperties	object	All the connection properties for manual input: <i>IMAP Server</i> (type string). Mandatory. The IMAP server where emails will be received from. Default: "imap.example.com" <i>IMAP Port</i> (type integer). Mandatory: The IMAP port number used by the email server. Default: 993 <i>User</i> (type string). Mandatory. User email from whom messages will be retrieved. Default: "" <i>Password</i> (type string). Mandatory. User password. Default: "" <i>Use TLS</i> (type boolean). Flag that controls whether the IMAP Server uses TLS or not. If the Server uses TLS and this is set to false or vice-versa, the operator will not work. Default: true

Input

Input	Type	Description
in	string	This port will inform what mailbox should be fetched. When this port is used, it disables the Polling mode and performs an email fetch for every single entry arriving at this port.

Output

Output	Type	Description
out	message	A message containing the Message Attributes and the Body of an email. The email body can be either in plain-text format or HTML format. Message Attributes: <i>email.to</i> (type Address). The recipient of the email. <i>email.from</i> (type Address). The sender of the email. <i>email.subject</i> (type string). The subject of the email. <i>email.cc</i> (type Address). The carbon copy recipients of the email. <i>email.date</i> (type time.Time). The date when the email was received at the server.

Additional Object Information

Address

An object that has two fields, Name(type string) and Address(type string). The Name field is used to describe the name of the person (or entity) who sent the email, which may be empty. The Address field contains the email which the email was received from, this is always filled.

2.2.18 Send Email

Sends emails. It uses an SMTP protocol. Every time data comes through the input port, a new email is built and sent.

Configuration Parameters

Configuration Parameters

Name	ID	Type	Description
Delete Attachments After Sending	Delete attachments after sending	bool	A flag that controls whether the files attached to the email should be deleted after sending or not. Default: false
Default From Value	Default from value	string	The value of <code>email.from</code>, which will be used when none is received through the incoming message. If not set, the value of user will be considered. Default: ""
Default to Value	Default to value	[]string	The value of <code>email.to</code>, which will be used when none is received through the incoming message. Default: []
Default Subject Value	Default subject value	string	The value of <code>email.subject</code>, which will be used when none is received through the incoming message. Default: ""

Note

Use `Connection Management` for defining connections over the manual option.

Connection Parameters

Name	ID	Type	Description
connection	connection	object	Holds information about connection information for the services.

Name	ID	Type	Description
configurationType	configurationType	string	Which type of connection information will be used: Manual (user input), or retrieved by the Connection Management Service. i Note It is recommended to choose <i>Configuration Manager</i> as the configuration type and select an existing <i>Connection ID</i>. The <i>Manual</i> configuration type must be used only for testing purposes. Default: ""
connectionID	connectionID	string	The ID of the connection information to retrieve from the Connection Management Service. Default: ""

Name	ID	Type	Description
connectionProperties	connectionProperties	object	All the connection properties for manual input: <i>Host</i> (type string, ID host): mandatory. Address of the SMTP server used to authenticate. Default:"smtp.example.com" <i>Port</i> (type integer, ID port): mandatory. The SMTP port number used by the Email Server. Default:"smtp.example.com" <i>User</i> (type string, ID user): mandatory. User email address used for authentication. Default: "email@example.com" <i>Password</i> (type string, ID password): mandatory. User password used for authentication. Default: "123456" <i>Insecure Skip Verify</i> (type bool, ID insecureSkipVerify): mandatory. A flag that indicate whether the certificate verification should be skipped.

Input

Input	Type	Description
in	message	A message whose attributes will be used to fill the email information and whose body will be used as the email body. In case there are multiple entries for a same header, they can be separated by a comma. Message Attributes: <i>email.to</i> (type string). Mandatory. The recipient of the email. <i>email.from</i> (type string). Mandatory. The sender of the email. <i>email.subject</i> (type string). Mandatory. The subject of the email. <i>email.cc</i> (type string). The carbon copy (additional recipients) of the email. <i>email.attachments</i> (type string). The attachments of the email. It expects the name of the files that should be attached. The operator will read those files and add as attachment.

Output

Output	Type	Description
out	string	A string containing a list of all the recipients that received the email, separated by semicolon (;).

Limitations

- The following characters cannot appear in message header names: <>\${ }.

Basic Usage

❖ Example

Suppose `alice@example.com` wants to send an email to `bob@example.com` containing the message: "Hi Bob! How are you doing? Best, Alice", and with the subject being "Greetings". The message arriving at the operator should look like this:

```
{"email.from": "alice@example.com", "email.to": "bob@example.com",  
"email.subject": "Greetings"}Hi Bob! How are you doing? Best, Alice
```

If attachments were to be included, the attribute "email.attachments" should be used.

Security

The Send Email operator can automatically detect if the SMTP server uses transport layer security (TLS).

2.2.19 WAMP Consumer

The WAMP Consumer operator is used to access records from Web Application Messaging Protocol (WAMP).

Configuration Parameters

Parameter	Type	Description
wampServer	string	Mandatory. The WAMP address. Default: "ws://localhost:8000/ws"
realm	string	Mandatory. The routing namespace and administrative domain. Default: "turnpike.chat.realm"
topic	string	Mandatory. The category name to which records are published. Default: "testwamp"
numRetryAttempts	int	The number of attempts to retry. Default: 0

Parameter	Type	Description
retryPeriodInMs	int	The time period in milliseconds for each retry. Default: 0

Input

None

Output

Output	Type	Description
outmessage	message	The stream of records published.

2.2.20 WAMP Producer

The WAMP Producer operator writes files to Web Application Messaging Protocol (WAMP).

Configuration Parameters

Parameter	Type	Description
wampServer	string	Mandatory. The WAMP address. Default: "ws://localhost:8000/ws"
realm	string	Mandatory. The routing namespace and administrative domain. Default: "turnpike.chat.realm"
topic	string	Mandatory. The category name to which records are published. Default: "testwamp"

Parameter	Type	Description
numRetryAttempts	int	The number of attempts to retry. Default: 0
retryPeriodInMs	int	The time period in milliseconds for each retry. Default: 0

Input

Input	Type	Description
inmessage	message	The stream of records to publish.

Output

None

2.3 Connectivity (via Flowagent)

Related Information

[Azure SQL Database SQL Consumer \[page 132\]](#)
[Azure SQL Database Table Consumer \[page 134\]](#)
[DB2 SQL Consumer \[page 136\]](#)
[DB2 Table Consumer \[page 138\]](#)
[Flowagent CSV Producer \[page 140\]](#)
[Flowagent File Producer \[page 143\]](#)
[Flowagent SQL Executor \[page 147\]](#)
[Flowagent Table Producer \[page 150\]](#)
[Google BigQuery Producer \[page 154\]](#)
[Google BigQuery SQL Consumer \[page 156\]](#)
[HANA Table Consumer \[page 157\]](#)

[MySQL SQL Consumer \[page 158\]](#)
[MySQL Table Consumer \[page 160\]](#)
[OData Query Consumer \[page 162\]](#)
[ODBC SQL Consumer \[page 163\]](#)
[ODBC Table Consumer \[page 166\]](#)
[Open Connectors SQL Consumer \[page 169\]](#)
[Open Connectors Table Consumer \[page 171\]](#)
[Oracle Table Consumer \[page 173\]](#)
[Oracle SQL Consumer \[page 175\]](#)
[SAP IQ Table Consumer \[page 177\]](#)
[SAP IQ SQL Consumer \[page 179\]](#)
[SQL Server SQL Consumer \[page 181\]](#)
[SQL Server Table Consumer \[page 182\]](#)
[Table Replicator \[page 184\]](#)

2.3.1 Azure SQL Database SQL Consumer

Performs a SQL query in an Azure SQL Database.

It uses Flowagent subengine for execution, and you need to connect it to a Flowagent-based producer operator (for example, [Flowagent CSV Producer \[page 140\]](#)).

Configuration Parameters

Name	ID	Type	Description
Azure SQL Database Connection	microsoftsqlserverConnection	object	Mandatory. The object containing the connection parameters to an Azure SQL database.
Native SQL	native_sql_statement	string	SQL statement that is used to read from the source. The syntax depends on the underlying source.

Name	ID	Type	Description
Partition Type	partitionType	string	Partitioning used to enable data reading in parallel. In this case, the producer must be inside a group with multiplicity equal to the number of partitions (only the producer, for example, Flowagent File Producer). Default: None
Logical Partition Specification	partitionSpecification	object	For information about partitioning, see Partition.
Additional Session Parameters	additionalSessionParameters	string	Session parameters that affect the connection with the source.
Fetch Size	fetchSize	integer	The number of rows that will be fetched from the source in each request. For wide tables, smaller values are better. For tables with fewer columns, larger values are better. Default: 100

Input

Input	Type	Description
inSQL	string	SQL statement that is used to read from the source. The syntax depends on the underlying source. i Note This is optional. It can be overridden by Native SQL, so this port can also be used as a trigger.

Output

Output	Type	Description
outConfig	string.flowagent.config	Flowagent type port, which only Flowagent producers can consume (for example, Flowagent File Producer). To get the data as string, Flowagent CSV Producer can be used as producer.
outError	string	Operator error. If mapped, the operator error is sent to the operator this port is mapped to and the graph remains running. If not mapped, the graph terminates with the operator error.

2.3.2 Azure SQL Database Table Consumer

Reads from an Azure SQL Database table or view.

It uses Flowagent subengine for execution, and you need to connect it to a Flowagent-based producer operator (for example, [Flowagent CSV Producer \[page 140\]](#))

Configuration Parameters

Name	ID	Type	Description
Azure SQL Database Connection	azuresqldbConnection	object	Mandatory. The object containing the connection parameters to an Azure SQL Database.
Source Table	adapted_dataset	object	Table from which the content will be read. It accepts two formats: adapted dataset of the target table, which can be selected via the table browsing action, or qualified name in <code><Schema>.<TableName></code> format.

Name	ID	Type	Description
Partition Type	partitionType	string	Partitioning used to enable data reading in parallel. In this case, the producer must be inside a group with multiplicity equal to the number of partitions (only the producer, for example, Flowagent File Producer). Default: None
Logical Partition Specification	partitionSpecification	object	For information about partitioning, see Partition.
Additional Session Parameters	additionalSessionParameters	string	Session parameters that affect the connection with the source.
Fetch Size	fetchSize	integer	The number of rows that will be fetched from the source in each request. For wide tables, smaller values are better. For tables with fewer columns, larger values are better. Default: 100

Input

Input	Type	Description
inTableName	string	SQL Server table name from which the data will be read, usually in <Schema>.<TableName> format. i Note This is optional. It can be overridden by Source Table, so this port can also be used as a trigger.

Output

Output	Type	Description
outConfig	string.flowagent.config	Flowagent type port, which only Flowagent producers can consume (for example, Flowagent File Producer). To get the data as string, Flowagent CSV Producer can be used as producer.
outError	string	Operator error. If mapped, the operator error is sent to the operator this port is mapped to and the graph remains running. If not mapped, the graph terminates with the operator error.

2.3.3 DB2 SQL Consumer

Performs a SQL query in a DB2 database.

It uses Flowagent subengine for execution, and you need to connect it to a Flowagent-based producer operator (for example, [Flowagent CSV Producer \[page 140\]](#)).

Configuration Parameters

Parameter	Type	Description
DB2 Connection	object	The object containing the connection parameters to a DB2 database.
Native SQL	string	SQL statement that will be used to read from the source. The syntax depends on the underlying source.
Additional Session Parameters	string	Set session parameters that will affect the connection with the source.
Partition Type	string	Partitioning is used to enable data reading in parallel. In this case, the producer must be inside a group with multiplicity equal to the number of partitions (only the producer, for example, Flowagent File Producer). Default: None

Parameter	Type	Description
Logical Partition Specification	object	For more information about partitioning, see Partition.
Fetch Size	integer	Indicates the number of rows that will be fetched from the source in each request. For wide tables, smaller values are better. For tables with fewer columns, larger values are better. Default: 100

Input

Input	Type	Description
inSQL	string	SQL statement that will be used to read from the source. The syntax depends on the underlying source.

i Note

This is optional. It can be overridden by Native SQL, so this port can also be used as a trigger.

Output

Output	Type	Description
outConfig	string.flowagent.config	Flowagent type port, which only Flowagent producers can consume (for example, Flowagent File Producer). To get the data as string, Flowagent CSV Producer can be used as producer.
outError	string	If mapped, the operator errors are sent to this port and the graph continues running. If not mapped, the graph terminates with the operator error.

2.3.4 DB2 Table Consumer

Reads from a DB2 table or view.

It uses Flowagent subengine for execution, and you need to connect it to a Flowagent-based producer operator (for example, [Flowagent CSV Producer \[page 140\]](#)).

Configuration Parameters

Parameter	Type	Description
DB2 Connection	object	The object containing the connection parameters to a DB2 database.
Source Table	string	Table from which the content will be read. It accepts two formats: adapted dataset of the target table, which can be selected via the table browsing action, or qualified name in <Schema>.<TableName> format.
Partition Type	string	Partitioning is used to enable data reading in parallel. In this case, the producer must be inside a group with multiplicity equal to the number of partitions (only the producer, for example, Flowagent File Producer). Default: None
Logical Partition Specification	object	For more information about partitioning, see Partition.
Additional Session Parameters	string	Set session parameters that will affect the connection with the source.
Fetch Size	integer	Indicates the number of rows that will be fetched from the source in each request. For wide tables smaller values are better, while for table with few columns larger values are better. Default: 1000

Input

Input	Type	Description
inTableName	string	SQL Server table name from which the data will be read, usually in <Schema>.<TableName> format. i Note This is optional. It can be overridden by Source Table, so this port can also be used as a trigger.

Output

Output	Type	Description
outConfig	string.flowagent.config	Flowagent type port, which only Flowagent producers can consume (for example, Flowagent File Producer). To get the data as string, Flowagent CSV Producer can be used as producer.
outError	string	If mapped, the operator errors are sent to this port and the graph continues running. If not mapped, the graph terminates with the operator error.

2.3.5 Flowagent CSV Producer

Reads any Flowagent-based consumer operator and outputs in a CSV format conforming to the RFC-4180 specification. The row delimiter is CRLF, the escape character and text delimiter are double quotes, and supported column delimiters include: , ; | : and \t.

Configuration Parameters

Parameter	Type	Description
Batch Size	integer	Indicates the number of rows that will be presented in each output batch. Default: 100
Format	string	Output file format. Default: "CSV"
Column Delimiter	char	The field separator. Only the following characters are supported: , ; : and TAB. Default: ,
Row Delimiter (user defined)	string	Indicates a user-defined row delimiter.
Null Value Replacement	string	Value that will be used to replace null fields.
Text Delimiter	char	Indicates which character will be used to delimit columns in a row. Default: "
Text Delimiter Style	string	Indicates how the text delimiter should be applied. If Always is selected, all columns will have a text delimiter; if Minimal, the text delimiter will be added only when necessary. Default: Minimal
Escape Character	char	Indicates which escape character will be used to escape the text delimiter. Default: "

Parameter	Type	Description
Header Included	bool	Indicates whether or not each CSV output will have a header as first line. Default: false

Input

Input	Type	Description
<code>inConfig</code>	<code>string.flowagent.config</code>	To be connected to flowagent based consumer operator.

Output

Output	Type	Description
<code>outContent</code>	<code>string</code>	CSV formatted data. If source has no rows, it will contain an empty string.

Output	Type	Description
outMessage	message	Outputs a message object with the following definition: • <code>Body</code> (type string): CSV-formatted data. If source has no rows, it will contain an empty string. • <code>Attributes</code>: ◦ <code>header</code> (type string): header of the CSV, if header included is true. ◦ <code>message.lastBatch</code> (type boolean): boolean that indicates if the current batch is the last one. ◦ <code>message.batchIndex</code> (type integer): number that indicates which batch is being processed, starting from 0. • <code>Encoding</code>: string See the graph <code>com.sap.dh.ingestion.csvproducer</code> to check how this port can be used along with a JS Message operator.
outError	string	If mapped, the operator errors are sent to this port and the graph continues running. If not mapped, the graph terminates with the operator error.

2.3.6 Flowagent File Producer

Reads from any Flowagent-based consumer operator and produces a file. The location of the file depends on specified storage type (local and cloud). The format of the file will be CSV, where the CSV properties can be specified in the CSV Properties.

Configuration Parameters

Name	ID	Type	Description
Storage Type	service	string	Optional. The storage in which the file will be written. Additional properties depend on the selected storage.
ADL Connection	adlConnection	object	Supported cloud storage. The object containing the connection parameters to a ADL remote.
GCS Connection	gcsConnection	object	Supported cloud storage. The object containing the connection parameters to a GCS remote.
HDFS Connection	hdfsConnection	object	Supported cloud storage. The object containing the connection parameters to a HDFS remote.
S3 Connection	s3Connection	object	Supported cloud storage. The object containing the connection parameters to a S3 remote.
WASB Connection	wasbConnection	object	Supported cloud storage. The object containing the connection parameters to a WASB remote.
SDL Connection	sdlConnection	object	Supported cloud storage. Connection to SAP-provided data lake.

Name	ID	Type	Description
Local File Name	path	string	A formatted string describing the output path and file name. If the root directory of the file is not provided, <code>"/vrep/"</code> will be used as default root directory of the file. Default: <code>"/vrep/<sourceTableName>_<timestamp>.csv"</code> - only applies when storage type is local.
Cloud Storage File Name		string	Only applies when Storage type is cloud storages. File path to be saved on cloud storage.
Remote File Name (Deprecated)	targetObjectName	string	Only applies when Storage type is cloud storages. File path to be saved on cloud storage. ! Restriction DEPRECATED: Use Cloud storage file name instead. Default: <code>"<sourceTableName>_<timestamp>.csv"</code>
Mode	mode	string	Defines the write mode, where overwrite will overwrite an existing file and append will append to an existing file. In both cases, if the file does not exist yet, it will be created during the write. i Note Important: GCS, S3, and, WASB do not allow append. Default: <code>"overwrite"</code>

Name	ID	Type	Description
Format	format	string	Output file format: CSV, PARQUET, and ORC are supported for the cloud storages. The CSV that is produced conforms to the RFC-4180 specification, where the row delimiter is always CRLF and the escape character is always double quotes. Currently, only CSV is supported for local storage. Default: "CSV"
CSV Properties	additionalProperties_csv	object	Properties of the CSV output.
Column Delimiter	columnDelimiter	string	The field separator. Only the following characters are supported: , ; : and TAB. Default: ,
Header Included	csvHeaderIncluded	boolean	Indicates whether or not each CSV output will have a header as first line. Default: false
Additional Properties	additionalProperties_common	object	Additional properties related to upload options to cloud storage. It is applicable only when a cloud storage service is selected.
Compression Type	compressionType	string	Compression type used when uploading file into cloud service. Default: "None"
Batch Size	batchSize	integer	Indicates the number of rows that will be inserted in a single commit. Default: 1000

Placeholders

For file names, we support three placeholders:

- `<sourceTableName>`: It will be replaced by the table name of the source, if this operator is connected to a Flowagent Table Consumer (for example, Oracle Table Consumer). For SQL Consumers, it will be replaced by `SQL_Table`, instead.
- `<timestamp>`: It will be replaced by the timestamp at the moment of the writing, in the format `yyyyMMdd_HH:mm:ss`.
- `<partitionName>`: It will be replaced by the partition number if the consumer has partitions (for example, 0, 1, ...).

Note

If not specified and the consumer has partitions, the partition number will always be appended to the file name to avoid concurrency issues during the writing.

Remote File Name Conventions

Each cloud provider has its own naming convention for defining bucket and object names. These conventions should be followed accordingly to avoid any source-related issue.

Even though Flowagent-based operators accept most special characters, we recommend that only alphanumeric characters are used. This will avoid any undesired issue because all sources work well with such characters. Furthermore, some special characters such as `"`, `+`, and `,` are not allowed by our Flowagent File Producer operator and should not be used.

Input

Input	Type	Description
<code>inConfig</code>	<code>string.flowagent.config</code>	The source connection information provided by Flowagent-based consumer operator from which data will be read.

Output

Output	Type	Description
<code>outFileName</code>	<code>string</code>	The path to the file to which content is written.

Output	Type	Description
Body	string	(type: string) The fully qualified name of the target table.
Attributes	According to parameter	- producer.totalRows (type: string): Number of rows written to the target. - producer.type (type: string): Type of the producer that generated the message, which can be one of the following depending on which operator generated the message: CSV for Flowagent CSV Producer, File for Flowagent File Producer, or Table for Flowagent Table Producer. - producer.subType (type: string): Subtype of the object that was generated in the target, which can be one of the following: TABLE, CSV, PARQUET, or ORC depending on the producer settings. - producer.partition.count (type: integer): Number of partitions from the source. - producer.partition.index (type: integer): Current partition index that was processed. This ranges from 0 to partition.count - 1.
Encoding	string	
outMessage	message	Outputs a message object with the definitions to follow.
outError	string	If mapped, the operator errors are sent to this port and the graph continues running. If not mapped, the graph terminates with the operator error.

2.3.7 Flowagent SQL Executor

Performs an SQL Statement to any supported database.

It uses Flowagent subengine to connect to the database. The specific SQL syntax depends on the specified database.

It provides a way to run SQL commands such as `CREATE`, `INSERT`, `DELETE`, and `UPDATE` in the database source. Multiple commands can be provided, where they must be separated by a specified separator (default is `;`). The output will be success if the commands were successfully run in the source. If a command fails, it will throw an error, which the `outError` port will catch if it is mapped.

Configuration Parameters

Name	ID	Type	Description
Database Type	service	string	Mandatory. Database on which the SQL statement(s) will be executed. Supported databases are displayed below.
ODBC Connection	odbcConnection	object	The object containing the connection parameters to an ODBC database. Available options are: • Data Source Name (ID <code>dsn</code>, type <code>string</code>, default <code>""</code>): Mandatory. The data source name of your ODBC connection, which needs to be configured in System Management under <code>flowagent/odbc.ini</code> file. • User (ID <code>user</code>, type <code>string</code>, default <code>""</code>): Mandatory. ODBC connection user. • Password (ID <code>password</code>, type <code>string</code>, default <code>""</code>): Mandatory. ODBC connection password.
Oracle Connection	oracleConnection	object	The object containing the connection parameters to an Oracle database.
MSSQL Connection	microsoftsqlserverConnection	object	The object containing the connection parameters to a MS SQL database.

Name	ID	Type	Description
Azure SQL Database Connection	azuresqlldbConnection	object	The object containing the connection parameters to an Azure SQL.
DB2 Connection	db2Connection	object	The object containing the connection parameters to a DB2 database.
HANA Connection	hanaConnection	object	The object containing the connection parameters to a HANA database.
MySQL Connection	mysqlConnection	object	The object containing the connection parameters to a MySQL database.
SAP IQ Connection	hanaConnection	object	The object containing the connection parameters to an SAP IQ database.
SQL Statement(s)	sqlStatements	string	SQL statements that are standard to the source. Statements must be separated by a specified separator. Default: ""
Separator	separator	string	Separator that will be used to split the input into SQL statements. Default: ","
Additional Session Parameters	additionalSessionParameters	string	Set session parameters that will affect the connection with the source. Default: ""

Input

Input	Type	Description
sql	string	SQL statements that are standard to the source. Statements must be separated by a defined separator.

Output

Output	Type	Description
result	string	Output <code>success</code> if the commands were successfully executed.
outError	string	If mapped, the operator errors are sent to this port and the graph continues running. If not mapped, the graph terminates with the operator error.

2.3.8 Flowagent Table Producer

Reads from any Flowagent-based consumer operator and writes in a table of any supported database.

Supported databases are:

- HANA
- Vora

Configuration Parameters

Parameter	Type	Description
Mode	string	Write mode. • Append: append to and existing table. • Overwrite: drop the table and create another one according to the source schema. • Truncate: truncate the table before writing any data. Default: "overwrite"
Database type	string	The target database to operate. Additional parameters may depend on the selected service. Default: "Vora"
HANA Wire port	string	Used when Database type is Vora. HANA Wire port of the transaction coordinator, which is determined as 3XX15, where XX is the instance number of the Vora cluster as configured in the Kubernetes cluster manager. For example, when the instance number of the Vora cluster is configured as 01, the HANA Wire port is 30115. A system administrator with the kubeconfig for the cluster can obtain the HANA Wire port by using the following command: <pre>kubectl describe deployment vora-tx-coordinator -n=YourNamespace grep tc_hana_wire_port</pre> You need to replace YourNamespace with your namespace, which you can get by running the following command: <pre>kubectl get namespaces</pre>

Parameter	Type	Description
Target Table	string	The table where the data will be written. This field accepts two input formats: adapted dataset of the table, which can be selected via the browsing action, or qualified name in <Schema>.<TableName> format.
Column Mapping	object	Defines how the source schema will be mapped to the target schema. It is an array where each element defines a mapping between a column in the source schema to a column in the target schema. If the array is empty, then the default mapping will be equivalent to mirroring the source schema in the target schema. Mapping use case: You can use mapping to change the order of source columns, or rename them. Example: <pre> "mapping": [{ "source": "LOC", "target": "LOC" }, { "source": "DNAME", "target": "TGT_DNAME" }, { "source": "DEPTNO", "target": "SomeNumberThatWeCareAbout" }] </pre>
Source Column Name	string	Column name in the source table.
Target Column Name	string	Column name in the target table.

Parameter	Type	Description
Vora Connection	object	The object containing the connection parameters to a Vora database. In a Vora instance exposed in the cluster, it should be used the external Vora connection, which has vora-tx-coordinator-ext as its hostname, and works with TLS enabled. The internal Vora connection, which has vora-tx-coordinator as its hostname, is not compatible with this operator. If you are using a Vora connection that is external to the cluster, be aware that the TLS refers to the HANA Wire port.
HANA Connection	object	The object containing the connection parameters to a HANA database.
Batch size	integer	Indicates the number of rows that will be inserted in a single commit. Default: 100

Delta Processing

`Change Data Applier` mode is applicable only with operator that supports DELTA extraction, like ABAP ODP Consumer, Cloud Data Integration Consumer, and so on.

Parameter	Type	Description
Change Data Applier Mode	string	The delta record provides a row type like: Insert (I), Update (U), Delete (D), Before Image (B), Unknown (X), and so on. User can either decide to keep a history of changed records (Track Change History), or decide to apply the changes directly to the target (Apply Change Data). Currently, only HANA database supports Apply Change Data mode. Default: "Track Change History"

Input

Input	Type	Description
<code>inConfig</code>	<code>string.flowagent.config</code>	The source connection information provided by Flowagent-based consumer operator from which data will be read.

Output

Output	Type	Description
<code>outTableName</code>	<code>string</code>	The fully qualified name of the target table.
<code>outError</code>	<code>string</code>	If mapped, the operator errors are sent to this port and the graph continues running. If not mapped, the graph terminates with the operator error.

2.3.9 Google BigQuery Producer

Writes to a Google BigQuery table. It uses the Flowagent subengine to connect to Google BigQuery. Users can either specify the full path of the source Google Cloud Storage file in the operator's configuration or connect any file producer that outputs the full path of Google Cloud Storage file to the input of the operator.

Configuration Parameters

Parameter	Type	Description
Google BigQuery Connection	<code>object</code>	The object containing the connection parameters to a Google BigQuery database.
Column delimiter	<code>string</code>	Delimiter used for columns in the source file.
Mode	<code>enum</code>	Write mode. Default: overwrite

Parameter	Type	Description
Source file	string	Full path name of the source Google Cloud Storage file.
Source FILE type	enum	File type of the source Google Cloud Storage file. Default: CSV
Target table name	string	The fully qualified name of target table to which content is written in <owner>.<table> format. The schema of the target table must match the schema of the source Google Cloud Storage file.

Input

Input	Type	Description
inSourceName	string	Full path name of Google Cloud Storage file, it can be overwritten by the Source name property, so the port is used as a trigger of the operator.

Output

Output	Type	Description
outTableName	string	The fully qualified name of target table to which content is written
outError	string	Operator error. If mapped, the operator error is sent to the operator this port is mapped to and the graph remains running. If not mapped, the graph terminates with the operator error.

2.3.10 Google BigQuery SQL Consumer

Performs a SQL query to a Google BigQuery table.

It uses the Flowagent subengine for execution, and you need to connect it to a Flowagent-based producer operator (for example, [Flowagent CSV Producer \[page 140\]](#)).

Configuration Parameters

Parameter	Type	Description
Google BigQuery Connection	object	The object containing the connection parameters to a Google BigQuery database.
BigQuery SQL	string	SQL statement that will be used to read from the source. Only syntax of BigQuery SQL (legacy SQL) is supported.

Input

Input	Type	Description
inSQL	string	SQL statement that is used to read from the source. The syntax depends on the underlying source.

i Note
This is optional. If BigQuery SQL is presented, this port is ignored.

Output

Output	Type	Description
outConfig	string.flowagent.config	Flowagent type port, which only Flowagent producers can consume (for example, Flowagent File Producer).

Output	Type	Description
<code>outError</code>	string	Operator error. If mapped, the operator error is sent to the operator this port is mapped to and the graph remains running. If not mapped, the graph terminates with the operator error.

2.3.11 HANA Table Consumer

The HANA Table Consumer reads from a HANA table or view.

It uses the Flowagent subengine for execution, and you need to connect it to a Flowagent-based producer operator (for example, [Flowagent CSV Producer \[page 140\]](#)).

Configuration Parameters

Parameter	Type	Description
HANA Connection	object	The object containing the connection parameters to a HANA database.
Source Table	string	HANA table name
Partition Type	string	Partitioning is used to enable data reading in parallel. In this case, the producer must be inside a group with multiplicity equal to the number of partitions (only the producer, for example, Flowagent File Producer). Default: None
Logical Partition Specification	object	For information about partitioning, see Partition.
Additional Session Parameters	string	Set session parameters that will affect the connection with the source.
Fetch Size	integer	Indicates the number of rows that will be fetched from the source in each request. For wide tables, smaller values are better; for tables with few columns, larger values are better. Default: 100

Input

Input	Type	Description
inTableName	string	Table name from which the data will be read, usually in <Schema>.<TableName> format.

i Note

This is optional. It can be overridden by Source Table, so this port can also be used as a trigger.

Output

Output	Type	Description
outConfig	string.flowagent.config	Flowagent type port, which only Flowagent producers can consume (for example, Flowagent File Producer). To get the data as string, Flowagent CSV Producer can be used as producer.
outError	string	If mapped, the operator errors are sent to this port and the graph continues running. If not mapped, the graph terminates with the operator error.

2.3.12 MySQL SQL Consumer

Performs a SQL query in a MySQL database.

It uses Flowagent subengine for execution, and you need to connect it to a Flowagent-based producer operator (for example, [Flowagent CSV Producer \[page 140\]](#)).

Configuration Parameters

Parameter	Type	Description
MySQL Connection	object	The object containing the connection parameters to a MySQL database.
Native SQL	string	SQL statement that is used to read from the source. The syntax depends on the underlying source.
Partition Type	string	Partitioning is used to enable data reading in parallel. In this case, the producer must be inside a group with multiplicity equal to the number of partitions (only the producer, for example, Flowagent File Producer). Default: None
Logical Partition Specification	object	For information about partitioning, see Partition.
Additional Session Parameters	string	Set session parameters that will affect the connection with the source.
Fetch Size	integer	Indicates the number of rows that will be fetched from the source in each request. For wide tables, smaller values are better. For tables with fewer columns, larger values are better. Default: 100

Input

Input	Type	Description
inSQL	string	MySQL table name from which the data will be read, usually in <code><Schema>.<TableName></code> format. i Note This is optional. It can be overridden by Native SQL, so this port can also be used as a trigger.

Output

Output	Type	Description
outConfig	string.flowagent.config	Flowagent type port, which only Flowagent producers can consume (for example, Flowagent File Producer). To get the data as string, Flowagent CSV Producer can be used as producer.
outError	string	Operator error. If mapped, the operator error is sent to the operator this port is mapped to and the graph remains running. If not mapped, the graph terminates with the operator error.

2.3.13 MySQL Table Consumer

Reads from a MySQL table or view.

It uses Flowagent subengine for execution, and you need to connect it to a Flowagent-based producer operator (for example, [Flowagent CSV Producer \[page 140\]](#)).

Configuration Parameters

Parameter	Type	Description
MySQL Connection	object	The object containing the connection parameters to a MySQL database.
Source Table	string	Table from which the content is read. It accepts two formats: adapted dataset of the target table, which can be selected via the table browsing action, or qualified name in <code><Schema>.<TableName></code> format.
Partition Type	string	Partitioning is used to enable data reading in parallel. In this case, the producer must be inside a group with multiplicity equal to the number of partitions (only the producer, for example, Flowagent File Producer). Default: None

Parameter	Type	Description
Logical Partition Specification	object	For information about partitioning, see Partition.
Additional Session Parameters	string	Set session parameters that affect the connection with the source.
Fetch Size	integer	Indicates the number of rows that will be fetched from the source in each request. For wide tables, smaller values are better. For tables with fewer columns, larger values are better. Default: 100

Input

Input	Type	Description
inTableName	string	MySQL table name from which the data will be read, usually in <code><Schema>.<TableName></code> .

Note

This is optional. If Source Table is presented, this port is ignored.

Output

Output	Type	Description
outConfig	string.flowagent.config	Flowagent type port, which only Flowagent producers can consume (for example, Flowagent File Producer). To get the data as string, Flowagent CSV Producer can be used as producer.
outError	string	If mapped, the operator errors are sent to this port and the graph continues running. If not mapped, the graph terminates with the operator error.

2.3.14 OData Query Consumer

The OData Query Consumer reads from an OData RESTful API.

It uses Flowagent subengine for execution, and you need to connect it to a Flowagent-based producer operator (for example, [Flowagent CSV Producer \[page 140\]](#)).

Configuration Parameters

Parameter	Type	Description
OData Connection	object	The object containing the connection parameters to OData.
Default (N) VARCHAR Size	integer	The default size of a <code>varchar</code> . Default: 1024
Batch Query	boolean	Whether the query is performed in batches or not. Default: false
Query	string	OData Query Uniform Resource Indicator (URI) that is standard to this OData source.
Fetch Size	integer	Indicates the number of rows that will be fetched from the source in each request. For wide tables, smaller values are better. For tables with fewer columns, larger values are better. Default: 100

Input

Input	Type	Description
<code>inSQL</code>	string	OData Query URI that is valid for this OData Source.

Note
This is optional. If Query is presented, this port is ignored.

Output

Output	Type	Description
<code>outConfig</code>	string.flowagent.config	FlowAgent type port, which only Flowagent producers can consume (for example, Flowagent File Producer). To get the data as string, Flowagent CSV Producer can be used as producer.
<code>outError</code>	string	If mapped, the operator errors are sent to this port and the graph continues running. If not mapped, the graph terminates with the operator error.

2.3.15 ODBC SQL Consumer

Performs an SQL query to any database that provides an Open Database Connectivity (ODBC) connector.

It uses the Flowagent subengine to connect to the database. To establish an ODBC connection, a Data Source Name (DSN) must be provided, specifying the DSN properties. The DSN must be defined in a file, following the UNIX standard. To effectively read from it, it is necessary to connect this operator to a Flowagent-based producer operator (for example, Flowagent CSV Producer).

For more information on Data Source Name (DSN), see section **Configure Data Source Name (DSN)**.

Configuration Parameters

Name	ID	Type	Description
ODBC Connection	odbcConnection	object	Mandatory. The object containing the connection parameters to an ODBC database.
Data Source Name	dsn	string	Mandatory. The data source name of your ODBC connection, which needs to be configured in System Management. Default: ""
User	user	string	Mandatory. ODBC connection user. Default: ""
Password	password	string	Mandatory. ODBC connection password. Default: ""
Native SQL	native_sql_statement	string	SQL statement that is used to read from the source. The syntax depends on the underlying source. Default: ""
Partition Type	partitionType	string	Partitioning is used to enable data reading in parallel. In this case, the producer must be inside a group with multiplicity equal to the number of partitions (only the producer, for example, Flowagent File Producer). Default: "None"
Logical Partition Specification	partitionSpecification	object	For information about partitioning, see Partition.

Name	ID	Type	Description
Additional Session Parameters	additionalSessionParameters	string	Set session parameters that affect the connection with the source. Default: ""
Fetch Size	fetchSize	integer	Indicates the number of rows that will be fetched from the source in each request. For wide tables, smaller values are better. For tables with fewer columns, larger values are better. Default: 1000

Input

Input	Type	Description
inSQL	string	MySQL table name from which the data will be read, usually in <Schema>.<TableName> format. i Note This is optional. It can be overridden by Native SQL, so this port can also be used as a trigger.

Output

Output	Type	Description
outConfig	string.flowagent.config	Flowagent type port, which only Flowagent producers can consume (for example, Flowagent File Producer). To get the data as string, Flowagent CSV Producer can be used as producer.

Output	Type	Description
<code>outError</code>	string	If mapped, the operator errors are sent to this port and the graph continues running. If not mapped, the graph terminates with the operator error.

Configure Data Source Name (DSN)

ODBC uses DSN to connect to the sources, and each database has its own configuration parameters in the DSN definition.

To configure a DSN, you must:

1. Start the System Management application from the Launchpad.
2. Choose the *Files* tab.
3. In the *User Workspace* or *Tenant Workspace*, create a folder named `flowagent` under the folder `files`.
4. Right-click the `flowagent` and create a file called `odbc.ini`.
5. In the `odbc.ini`, you must add the DSN configuration in a UNIX style.

i Note

In the DSN definition, `/vrep` is equivalent to the `files` folder in the top level of the System Management files.

2.3.16 ODBC Table Consumer

The ODBC Table Consumer reads a table from any database that provides an Open Database Connectivity (ODBC) connector.

It uses the Flowagent subengine to connect to the database. To establish an ODBC connection, a Data Source Name (DSN) must be provided, specifying the DSN properties. The DSN must be defined in a file, following the UNIX standard.

For more information on Data Source Name (DSN), see section **Configure Data Source Name (DSN)**.

Configuration Parameters

Name	ID	Type	Description
ODBC Connection	odbcConnection	object	Mandatory. The object containing the connection parameters to an ODBC database.
Data Source Name	dsn	string	Mandatory. The data source name of your ODBC connection, which needs to be configured in System Management. Default: ""
User	user	string	Mandatory. ODBC connection user. Default: ""
Password	password	string	Mandatory. ODBC connection password. Default: ""
Table Name	table_name	string	Mandatory. Source table name usually in <Schema>.<Tablename> format. Default: ""
Partition Type	partitionType	string	Partitioning is used to enable data reading in parallel. In this case, the producer must be inside a group with multiplicity equal to the number of partitions (only the producer, for example, Flowagent File Producer). Default: "None"
Additional Session Parameters	additionalSessionParameters	string	Set session parameters that affect the connection with the source. Default: ""

Name	ID	Type	Description
Fetch Size	fetchSize	integer	Indicates the number of rows that will be fetched from the source in each request. For wide tables, smaller values are better. For tables with fewer columns, larger values are better. Default: 1000

Input

Input	Type	Description
inTableName	string	SQL Server table name from which the data is read, usually in <Schema>.<TableName> format. i Note Defining this port is optional. If Source Table is defined, then this port is ignored. Therefore, the input port can be used as a trigger.

Output

Output	Type	Description
outConfig	string.flowagent.config	Flowagent type port, which only Flowagent producers can consume (for example, Flowagent File Producer). To get the data as string, Flowagent CSV Producer can be used as producer.
outError	string	If mapped, the operator errors are sent to this port and the graph continues running. If not mapped, the graph terminates with the operator error.

Configure Data Source Name (DSN)

ODBC uses DSN to connect to the sources, and each database has its own configuration parameters in the DSN definition.

To configure a DSN, you must:

1. Start the System Management application from the Launchpad.
2. Choose the [Files](#) tab.
3. In the [User Workspace](#) or [Tenant Workspace](#), create a folder named `flowagent` under the folder `files`.
4. Right-click the `flowagent` and create a file called `odbc.ini`.
5. In the `odbc.ini`, you must add the DSN configuration in a UNIX style.

Note

In the DSN definition, `/vrep` is equivalent to the `files` folder in the top level of the System Management files.

2.3.17 Open Connectors SQL Consumer

The Open Connectors SQL Consumer gets and filters data by the given SQL statement from SAP Cloud Platform Open Connectors.

It uses the Flowagent subengine for execution, and you need to connect it to a Flowagent-based producer operator (for example, [Flowagent CSV Producer \[page 140\]](#)).

Configuration Parameters

Parameter	Type	Description
Open Connectors Connection	object	Connection parameters or connection ID of an Open Connectors connection in Connection Management.
Access Method	string	Open Connectors API used for retrieving data. In GET mode, Open Connectors' data query API will be called directly once per page to retrieve data. In BULK method, an asynchronous query job will be created at Open Connectors and then start streaming data. Default: GET

Parameter	Type	Description
Native SQL	string	SQL statement that is used to read and filter data from the source.

Input

Input	Type	Description
inSQL	string	SQL statement that is used to read and filter data from the source. Supported SQL syntax is limited as follows: <pre>SELECT <field-list> FROM <tableName> [WHERE <where-clause>] [ORDER BY <order-by-clause>]</pre> - field-list = * field1, field2, ... - where-clause = WHERE expression AND/OR expression AND/OR ... - order-by-clause = field1 asc/desc (only allow one field) - expression = field operator value

Valid Operators

Operator	Description
=	Equal to
!=	Not equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to
LIKE	TRUE if the operand matches a pattern
IN	TRUE if the operand is equal to one of a list of expressions

Operator	Description
IS NULL	TRUE if a NULL value is found

Data Type Mapping

Data Type Mapping

Open Connection Type	Vendor Native Type	SAP Data Intelligence Type
string		NVARCHAR(5000)
numeric	int	INTEGER
numeric		FLOATING(8)
boolean		INTEGER
date	date	DATE
date	datetime	DATETIME
object		NVARCHAR(5000)

Output

Output	Type	Description
outConfig	string.flowagent.config	Flowagent type port, which only Flowagent producers can consume (for example, <code>Flowagent File Producer</code>). To get the data as string, <code>Flowagent CSV Producer</code> can be used as producer.
outError	string	If mapped, the operator errors are sent to this port and the graph continues running. If not mapped, the graph terminates with the operator error.

2.3.18 Open Connectors Table Consumer

Open Connectors Table Consumer gets all data from the given table name from SAP Cloud Platform Open Connectors.

It uses the Flowagent subengine for execution (for example, [Flowagent CSV Producer \[page 140\]](#)).

Configuration Parameters

Parameter	Type	Description
Open Connectors Connection	object	Connection parameters or connection ID of an Open Connectors connection in Connection Management.
Source Table	string	Source table name in an SAP Cloud Platform Open Connectors instance. It's also called <code>object name</code> in Open Connectors.
Access Method	string	Open Connectors API used for retrieving data. In GET mode, Open Connectors' data query API will be called directly once per page to retrieve data. In BULK method, an asynchronous query job will be created at Open Connectors and then start streaming data. Default: GET

Input

Input	Type	Description
inTableName	string	Source table name in an SAP Cloud Platform Open Connectors instance. It's also called <code>object name</code> in Open Connectors.

Data Type Mapping		
Open Connection Type	Vendor Native Type	SAP Data Intelligence Type
string		NVARCHAR(5000)
numeric	int	INTEGER
numeric		FLOATING(8)
boolean		INTEGER
date	date	DATE
date	datetime	DATETIME

Open Connection Type	Vendor Native Type	SAP Data Intelligence Type
object		NVARCHAR(5000)

Output

Output	Type	Description
outConfig	string.flowagent.config	Flowagent type port, which only Flowagent producers can consume (for example, Flowagent File Producer). To get the data as string, Flowagent CSV Producer can be used as producer.
outError	string	If mapped, the operator errors are sent to this port and the graph continues running. If not mapped, the graph terminates with the operator error.

2.3.19 Oracle Table Consumer

The Oracle Table Consumer reads from an Oracle table or view.

It uses the Flowagent subengine for execution, and you need to connect it to a Flowagent-based producer operator (for example, [Flowagent CSV Producer \[page 140\]](#)).

Configuration Parameters

Parameter	Type	Description
Oracle Connection	object	The object containing the connection parameters to an Oracle database.
Source Table	string	Table from which the content will be read. It accepts two formats: adapted dataset of the target table, which can be selected via the table browsing action, or qualified name in <Schema>.<TableName> format.

Parameter	Type	Description
Partition Type	string	It can be either Logical (User Defined), Physical (Source Based), or Row ID. Partitioning is used to enable data reading in parallel. In this case, the producer must be inside a group with multiplicity equal to the number of partitions (only the producer, for example, Flowagent File Producer). Default: None
Logical Partition Specification	object	For information about partitioning, see Partition.
Number of partitions	number	If Partition Type is set to Row ID Partitioning, this field will be visible and it will define the number of partitions which will be generated based on the row ids of the table.
Fetch Size	integer	Indicates the number of rows that will be fetched from the source in each request. For wide tables smaller values are better. For tables with few columns larger values are better. Default: 100

Input

Input	Type	Description
inTableName	string	SQL Server table name from which the data is read, usually in <Schema>.<Tablename> format. i Note Defining this port is optional. If <code>Source Table</code> is defined, then this port is ignored. Therefore, the input port can be used as a trigger.

Output

Output	Type	Description
outConfig	string.flowagent.config	Flowagent type port, which only Flowagent producers can consume (for example, Flowagent File Producer). To get the data as string, Flowagent CSV Producer can be used as producer.
outError	string	If mapped, the operator errors are sent to this port and the graph continues running. If not mapped, the graph terminates with the operator error.

2.3.20 Oracle SQL Consumer

The Oracle SQL Consumer performs a SQL query in an Oracle database.

It uses the Flowagent subengine for execution, and you need to connect it to a Flowagent-based producer operator (for example, [Flowagent CSV Producer \[page 140\]](#)).

Configuration Parameters

Parameter	Type	Description
Oracle Connection	object	The object containing the connection parameters to an Oracle database.
Native SQL	string	SQL statement that is used to read from the source. The syntax depends on the underlying source. Use the SQL editor to enter the SQL statement.
Partition Type	string	Partitioning is used to enable data reading in parallel. In this case, the producer must be inside a group with multiplicity equal to the number of partitions (only the producer , for example, Flowagent File Producer). Default: None

Parameter	Type	Description
Logical Partition Specification	object	For information about partitioning, see Partition.
Fetch Size	integer	Indicates the number of rows that will be fetched from the source in each request. For wide tables, smaller values are better. For tables with fewer columns, larger values are better. Default: 100

Input

Input	Type	Description
inSQL	string	SQL statement used to read from the source. The syntax depends on the underlying source.

i Note

This is optional. It can be overridden by Native SQL, so this port can also be used as a trigger.

Output

Output	Type	Description
outConfig	string.flowagent.config	Flowagent type port, which only Flowagent producers can consume (for example, Flowagent File Producer). To get the data as string, Flowagent CSV Producer can be used as producer.
outError	string	If mapped, the operator errors are sent to this port and the graph continues running. If not mapped, the graph terminates with the operator error.

2.3.21 SAP IQ Table Consumer

The SAP IQ Table Consumer reads from an SAP IQ table or view.

It uses the Flowagent subengine for execution, and you need to connect it to a Flowagent-based producer operator (for example, [Flowagent CSV Producer \[page 140\]](#)).

Configuration Parameters

Name	ID	Type	Description
SAP IQ Connection	sapiqConnection	object	Mandatory. The object containing the connection parameters to an SAP IQ database.
Source Table	adapted_dataset	object	Mandatory. Table from which the content will be read. It accepts two formats: adapted dataset of the target table, which can be selected via the table browsing action, or qualified name in <Schema>.<TableName> format.
Partition Type	partitionType	string	Partitioning is used to enable data reading in parallel. In this case, the producer must be inside a group with multiplicity equal to the number of partitions (only the producer, for example, Flowagent File Producer). Default: "None"
Logical Partition Specification	partitionSpecification	object	For information about partitioning, see Partition.
Additional Session Parameters	additionalSessionParameters	string	Set session parameters, which will affect the connection with the source. Default: ""

Name	ID	Type	Description
Fetch Size	fetchSize	integer	Indicates the number of rows that will be fetched from the source in each request. For wide tables, smaller values are better. For tables with fewer columns, larger values are better. Default: 1000

Input

Input	Type	Description
inTableName	string	SQL Server table name from which the data is read, usually in <Schema>.<Tablename> format. Note Defining this port is optional. If <code>Source Table</code> is defined, then this port is ignored. Therefore, the input port can be used as a trigger.

Output

Output	Type	Description
outConfig	string.flowagent.config	Flowagent type port, which only Flowagent producers can consume (for example, Flowagent File Producer). To get the data as string, Flowagent CSV Producer can be used as producer.
outError	string	If mapped, the operator errors are sent to this port and the graph continues running. If not mapped, the graph terminates with the operator error.

2.3.22 SAP IQ SQL Consumer

Performs an SQL query to an SAP IQ database.

It uses the Flowagent subengine for execution, and you need to connect it to a Flowagent-based producer operator (for example, [Flowagent CSV Producer \[page 140\]](#)).

Configuration Parameters

Name	ID	Type	Description
SAP IQ Connection	sapiqConnection	object	Mandatory. The object containing the connection parameters to an SAP IQ database.
Native SQL	native_sql_statement	string	SQL statement used to read from the source. The syntax depends on the underlying source. Default: ""
Partition Type	partitionType	string	Partitioning is used to enable data reading in parallel. In this case, the producer must be inside a group with multiplicity equal to the number of partitions (only the producer, for example, Flowagent File Producer). Default: "None"
Logical Partition Specification	partitionSpecification	object	For information about partitioning, see Partition.
Additional Session Parameters	additionalSessionParameters	string	Set session parameters that will affect the connection with the source. Default: ""

Name	ID	Type	Description
Fetch Size	fetchSize	integer	Indicates the number of rows that will be fetched from the source in each request. For wide tables, smaller values are better. For tables with fewer columns, larger values are better. Default: 1000

Input

Input	Type	Description
inSQL	string	SQL statement used to read from the source. The syntax depends on the underlying source. i Note This is optional. It can be overridden by Native SQL, so this port can also be used as a trigger.

Output

Output	Type	Description
outConfig	string.flowagent.config	Flowagent type port, which only Flowagent producers can consume (for example, Flowagent File Producer). To get the data as string, Flowagent CSV Producer can be used as producer.
outError	string	If mapped, the operator errors are sent to this port and the graph continues running. If not mapped, the graph terminates with the operator error.

2.3.23 SQL Server SQL Consumer

The SQL Server SQL Consumer performs an SQL query in an SQL Server database.

It uses the Flowagent subengine for execution, and you need to connect it to a Flowagent-based producer operator (for example, [Flowagent CSV Producer \[page 140\]](#)).

Configuration Parameters

Name	ID	Type	Description
SQL Server Connection	microsoftsqlserverConnec-tion	object	The object containing the connection parameters to a MySQL database.
Native SQL	native_sql_statement	string	SQL statement used to read from the source. The syntax depends on the underlying source.
Partition Type	partitionType	string	Partitioning used to enable data reading in parallel. In this case, the producer must be inside a group with multiplicity equal to the number of partitions (only the producer, for example, Flowagent File Producer). Default: None
Logical Partition Specifica-tion	partitionSpecification	object	For information about partitioning, see Partition.
Additional Session Paramete-rs	additionalSessionParameters	string	Session parameters that affect the connection with the source.
Fetch Size	fetchSize	integer	The number of rows that will be fetched from the source in each request. For wide tables, smaller values are better. For tables with fewer columns, larger values are better. Default: 100

Input

Input	Type	Description
inSQL	string	SQL statement used to read from the source. The syntax depends on the underlying source.

i Note

This is optional. It can be overridden by Native SQL, so this port can also be used as a trigger.

Output

Output	Type	Description
outConfig	string.flowagent.config	Flowagent type port, which only Flowagent producers can consume (for example, Flowagent File Producer). To get the data as string, Flowagent CSV Producer can be used as producer.
outError	string	Operator error. If mapped, the operator error is sent to the operator this port is mapped to and the graph remains running. If not mapped, the graph terminates with the operator error.

2.3.24 SQL Server Table Consumer

The SQL Server Table Consumer reads from an SQL Server table or view.

It uses the Flowagent subengine for execution, and you need to connect it to a Flowagent-based producer operator (for example, [Flowagent CSV Producer \[page 140\]](#)).

Configuration Parameters

Name	ID	Type	Description
SQL Server Connection	microsoftsqlserverConnec- tion	object	The object containing the connection parameters to a MySQL database.
Source Table	adapted_dataset	string	SQL Server table name from which the data is read, usually in <code><Schema>.<TableName></code> format.
Partition Type	partitionType	string	Partitioning used to enable data reading in parallel. In this case, the producer must be inside a group with multiplicity equal to the number of partitions (only the producer, for example, Flowagent File Producer). Default: None
Logical Partition Specification	partitionSpecification	object	For information about partitioning, see Partition.
Additional Session Parameters	additionalSessionParameters	string	Session parameters that will affect the connection with the source.
Fetch Size	fetchSize	integer	The number of rows that will be fetched from the source in each request. For wide tables, smaller values are better. For tables with fewer columns, larger values are better. Default: 100

Input

Input	Type	Description
<code>inTableName</code>	string	SQL Server table name from which the data is read, usually in <code><Schema>.<TableName></code> format.

i Note

This is optional. It can be overridden by Source Table, so this port can also be used as a trigger.

Output

Output	Type	Description
<code>outConfig</code>	string.flowagent.config	Flowagent type port, which only Flowagent producers can consume (for example, Flowagent File Producer). To get the data as string, Flowagent CSV Producer can be used as producer.
<code>outError</code>	string	Operator error. If mapped, the operator error is sent to the operator this port is mapped to and the graph remains running. If not mapped, the graph terminates with the operator error.

2.3.25 Table Replicator

Change Data Capture (CDC) is a Delta Capture Technique that uses triggers for Insert, Update, and Delete to track the change history for a specific table.

In addition to capturing the delta, this operator also handles applying the changes to a target table, allowing replication of tables in an efficient and secure way through an in-built recovery mechanism.

Depending on the selected Execution Mode, this operator performs one of the following operations:

- Delta: At first, it creates all necessary triggers in the source, and then performs an initial load. For consecutive executions, it only extracts delta records created since the last run.
- Clean-up: It removes all previously created triggers, procedures, and tables in the source

Supported Sources

- Azure SQL Database
- DB2
- HANA
- MySQL
- Microsoft SQL Server
- Oracle

i Note

The following source-specific SQL Privileges are required. Ensure that the connection user has the following privileges:

- CREATE TABLE
- CREATE VIEW
- CREATE SEQUENCE
- CREATE PROCEDURE
- CREATE TRIGGER

Supported Targets

- Databases (HANA, Vora)
- Cloud Storage (ADL, WASB, S3, OSS, GCS, HDFS)
- Local File system (Local)

Modes

For HANA as target we support 2 modes:

- Track Change History
- Apply Change Data

When using Apply Change Data, you can either copy primary keys only or all columns. When you select primary keys only, the created triggers are faster and create less impact on the source, but the replication is slower. When all columns are selected, the created triggers are slower but the replication is faster.

For other targets, only **Track Change History** is supported.

Limitations

- Schema Changes (for example, changes through DDLs) are not supported

- Truncate Table operation is not supported

Migration

If triggers and procedures have already been created for a given table by the CDC Graph Generator (`com.sap.cdc.graphGenerator`) operator, then it's necessary to either stick to the generated Delta Graph to do the Delta processing or execute this operator in **Clean Up** execution mode to remove all triggers and procedures for a given source table and then re-execute it again in **Delta** execution mode.

⚠ Caution

This action drops all nonapplied shadow records.

Operator Usage

CDC performance depends on the initial table size and how rapidly change data is being generated in the source table. To optimize and process different scenarios, here are some helpful points on using this operator.

- Initial Load Volume
 - If the source table is large, you can partition the source data using the **Partition Specification** property. After the partitions are defined based on the resources available, you can set the **Max Number of Loaders** property to an appropriate value (default is 8).
- Delta Change Rate
 - For rapidly changing tables, we recommended setting the **Delta Graph Run Mode** field under the **Delta Load Properties** property to **Polling Interval**, with a short **Max Polling Interval**. This way, the operator runs continuously, fetching the data in short intervals.
 - For slowly changing tables, one can either set the **Delta Graph Run Mode** field to **Manual** and schedule the graph to run periodically (every N hours), or set the *Delta Graph Run Mode* to **Polling Interval** with high **Max Polling Interval**. When **Polling Interval** is used, the graph runs continuously, while in Manual mode the graph fetches the Delta records once and then stops processing. You can then terminate the graph connecting the output of this operator to a graph terminator.

Tracking Multiple Tables

This operator handles a single table at a time. If it is necessary to track multiple tables, create an instance of this operator for each source-target pair. However, if each operator runs in a separate graph, it uses a lot of storage, because each graph has at least one additional pod. Here are some strategies to reduce resource usage. Consider the example below:

❖ Example

You have 10 source tables. Two of them are rapidly changing while the other eight are slowly changing.

For rapidly changing tables (see the Operator Usage section above), you should create one graph per table. This graph should be run forever in **Polling Interval** mode with short **Max Polling Interval**.

For the slowly changing tables, you can chain this operator multiple times, one after another in the same graph. While chaining this operator, set the *Delta Graph Run Mode* to **Manual**. This setting means that when the first operator completes its data movement, the second one is started, and so on. While chaining, you can group every N operators (recommended: 5), so that each group runs in their own pod. To save further resource usage, schedule this graph based on the Delta change rate on the source (see the Operator Usage section above). This way, you are not consuming resources for slowly changing tables or reducing the number of used SQL connections to the source.

Note

- The recommended group resource is 0.5 CPU and 1500 m memory. More than 2 CPUs are not helpful for processing. However, more RAM (max is 4 GB) is better when the operators are chained.
- RAM requirements are based on the defined *Source Fetch Size* and *Target Batch Size*.
 - For wide tables (over 100 columns), a smaller fetch size is better (<1000) and higher RAM value.
 - For narrow tables, a larger fetch size gives better throughput.
- During Delta processing, a maximum of 2 SQL connections are required for each source table. However, during the Initial Load, we use 1 + *Max Number of Loaders* connections.

Delta Extraction Logic

Several methods exist to capture the changes of a source table. Timestamp-based, native CDC, log-based reading, and so on. Native CDC methods with transactional consistency are difficult to scale due to their atomicity nature. For Data Intelligence purposes where the data volume can be significantly higher for certain tables, we recommend using triggers. Note that trigger is the only method that is supported for all databases.

To capture the changes of a source table, a few objects are created in the source schema.

Object	Type	Description
srcTable_ICL	PROCEDURE	Procedure to drop all auxiliary objects created for the Delta extraction logic
srcTable_SEQ	SEQUENCE	A sequence used to generate an operation ID for each Delta record
srcTable_SHA	TABLE	Shadow table used to store the Delta records
srcTable_SHV	VIEW	View that displays the current batch of Delta records in the shadow table
srcTable_CHK	TABLE	Checkpoint table used to flag some values necessary for the replication process, such as flagging the timestamp of when the replication began, or the number delta records that have been applied so far
srcTable_INS	TRIGGER	Insert Trigger
srcTable_DEL	TRIGGER	Delete Trigger
srcTable_UPD	TRIGGER	Update Trigger

Object	Type	Description
srcTable_ISS	PROCEDURE	Procedure to set some flags in the srcTable_CHECKPOINT table
srcTable_SCB	PROCEDURE	Procedure to mark shadow table records to be extracted
srcTable_SCE	PROCEDURE	Procedure to mark shadow table records to be deleted
srcTable_SCL	PROCEDURE	Procedure to remove records from shadow table

Given those objects, the Delta extraction logic happens as follows:

1. The triggers capture all changes in the source table and copy them to the `srcTable_SHADOW` table.
2. A graph containing this operator is started.
3. This operator calls the `srcTable_INITIAL_SHADOW_SCAN` procedure to set some flags in the `srcTable_CHECKPOINT` table.
4. This operator calls the `srcTable_SHADOW_CHECKPOINT_BEGIN` procedure to mark a batch of records in the shadow table to be extracted.
5. An ingestion job is started to apply all records in the `srcTable_SHADOW_VIEW` to the target.
6. This operator calls the `srcTable_SHADOW_CHECKPOINT_END` procedure to mark all records in `srcTable_SHADOW` table previously marked to be extracted now to be deleted.
7. This operator calls the `srcTable_SHADOW_CLEANUP` to remove all records in the `srcTable_SHADOW` table that are marked for deletion.

Note

The SQL statements for each database are located under `/vrep/vflow/subengines/com/sap/dh/flowagent/operators/com/sap/dh/cdc/initialize/sqlFiles/V2`, which can be inspected and modified before running the operator.

Configuration Parameters

Name	ID	Type	Description
Execution Mode	execution-Mode	string	Mandatory. Choose an operation. • Delta: Apply all Delta records from source to target since the last execution. • Clean Up: Remove all temporary objects and triggers created in the source. Default: ""
Source	source	string	Mandatory. Database of the source table. Supported databases are displayed below. Default: ""
Source Azure SQL DB Connection	azuresqldb-Connection	object	The object containing the connection parameters to an Azure SQL DB database.

Name	ID	Type	Description
Source DB2 Connection	db2Connection	object	The object containing the connection parameters to a DB2 database.
Source HANA Connection	hanaConnection	object	The object containing the connection parameters to a HANA database.
Source MSSQL Connection	mssqlConnection	object	The object containing the connection parameters to an MSSQL database.
Source MySQL Connection	mysqlConnection	object	The object containing the connection parameters to a MySQL database.
Source Oracle Connection	oracleConnection	object	The object containing the connection parameters to an Oracle database.
Source Azure SQL DB Table	azuresqldbAdaptedDataset	object	Mandatory. Source Azure SQL DB table from which the Delta data is read.
Source DB2 Table	db2AdaptedDataset	object	Source DB2 table from which the Delta data is read.
Source HANA Table	hanaAdaptedDataset	object	Source HANA table from which the Delta data is read.
Source MSSQL Table	mssqlAdaptedDataset	object	Source MSSQL table from which the Delta data is read.
Source MySQL Table	mysqlAdaptedDataset	object	Source MySQL table from which the Delta data is read.
Source Oracle Table	oracleAdaptedDataset	object	Source Oracle table from which the Delta data is read.
Azure SQL DB Partition Type	azuresqldbPartitionType	string	To load the data in parallel, partition the source data during the initial load. Default: "None":
Azure SQL DB Logical Partition Specification	azuresqldbPartitionSpecification	object	For more information about partitioning, check the Working with Flowagent Subengine to Connect to Databases documentation.
Azure SQL DB Max Number of Loaders	azuresqldbMultiplicity	integer	Defines the maximum number of parallel loaders during the initial load. Each parallel loader is equivalent to a pod. The final number of loaders is equal to the minimum (maximum number of loaders, number of partitions). Default: 8
DB2 Partition Type	db2PartitionType	string	To load the data in parallel, partition the source data during the initial load. Default: "None"
DB2 Logical Partition Specification	db2PartitionSpecification	object	For more information about partitioning, check the Working with Flowagent Subengine to Connect to Databases documentation.
DB2 Max Number of Loaders	db2Multiplicity	integer	Defines the maximum number of parallel loaders during the initial load. Each parallel loader is equivalent to a pod. The final number of loaders is equal to the minimum (maximum number of loaders, number of partitions). Default: 8

Name	ID	Type	Description
HANA Partition Type	hanaPartitionType	string	To load the data in parallel, partition the source data during the initial load.
HANA Logical Partition Specification	hanaPartitionSpecification	object	For more information about partitioning, check the Working with Flowagent Subengine to Connect to Databases documentation.
HANA Max Number of Loaders	hanaMultiplicity	integer	Defines the maximum number of parallel loaders during the initial load. Each parallel loader is equivalent to a pod. The final number of loaders is equal to the minimum (maximum number of loaders, number of partitions). Default: 8
MySQL Partition Type	mysqlPartitionType	string	To load the data in parallel, partition the source data during the initial load. Default: "None"
MySQL Logical Partition Specification	mysqlPartitionSpecification	object	For more information about partitioning, check the Working with Flowagent Subengine to Connect to Databases documentation.
MySQL Max Number of Loaders	mysqlMultiplicity	integer	Defines the maximum number of parallel loaders during the initial load. Each parallel loader is equivalent to a pod. The final number of loaders is equal to the minimum (maximum number of loaders, number of partitions). Default: 8
MSSQL Partition Type	mssqlPartitionType	string	To load the data in parallel, partition the source data during the initial load.
MSSQL Logical Partition Specification	mssqlPartitionSpecification	object	For more information about partitioning, check the Working with Flowagent Subengine to Connect to Databases documentation.
MSSQL Max Number of Loaders	mssqlMultiplicity	integer	Defines the maximum number of parallel loaders during the initial load. Each parallel loader is equivalent to a pod. The final number of loaders is equal to the minimum (maximum number of loaders, number of partitions). Default: 8
Oracle Partition Type	oraclePartitionType	string	To load the data in parallel, partition the source data during the initial load. Default: "None"
Oracle Logical Partition Specification	oraclePartitionSpecification	object	For more information about partitioning, check the documentation.
Oracle Number of partitions	oracleNumberOfPartitions	integer	When partition type is set to Row ID , specify the number of ranges in the table for partitioning.
Oracle Max Number of Loaders	oracleMultiplicity	integer	Defines the maximum number of parallel loaders during the initial load. Each parallel loader is equivalent to a pod. The final number of loaders is equal to the minimum (maximum number of loaders, number of partitions).
Source Tablespace	tablespace	string	Tablespace where auxiliary tables are created Default: ""

Name	ID	Type	Description
Source Fetch Size	fetchSize	integer	Indicates the number of rows fetched from the source in each request. For wide tables, smaller values are better. For tables with fewer columns, larger values are better. Default: 1000
Target	target	string	System of the target table or file. Supported systems are displayed below.
Target HANA Connection	hanaTarget-Connection	object	The object containing the connection parameters to a HANA database.
Target Vora Connection	voraTarget-Connection	object	The object containing the connection parameters to a Vora database.
Target HANA Wire port	hanaWired-Port	string	HANA Wire port of the transaction coordinator, which is determined as 3XX15, where XX is the instance number of the Vora cluster as configured in the Kubernetes cluster manager. For example, when the instance number of the Vora cluster is configured as 01, the HANA Wire port is 30115. A system admin with the kubeconfig for the cluster can obtain the HANA Wire port by using the following command: <pre>kubectl describe deployment vora-tx-coordinator -n=YourNamespace grep tc_hana_wire_port</pre> You need to replace YourNamespace with your namespace, which you can get by running the following command: <pre>kubectl get namespaces</pre> Default: "30115"
Target ADL Connection	adlTargetConnection	object	The object containing the connection parameters to an ADL storage.
Target SDL Connection	sdlTargetConnection	object	The object containing the connection parameters to an SDL storage.
Target HDFS Connection	hdfsTarget-Connection	object	The object containing the connection parameters to a HDFS storage.
Target GCS Connection	gcsTargetConnection	object	The object containing the connection parameters to a GCS storage.
Target S3 Connection	s3TargetConnection	object	The object containing the connection parameters to a S3 storage.
Target OSS Connection	ossTargetConnection	object	The object containing the connection parameters to an OSS storage.
Target WASB Connection	wasbTarget-Connection	object	The object containing the connection parameters to a WASB storage.
Target HANA Table	hanaTargetAdaptedDataSet	object	Target HANA table where the Delta data is written.
Target VORA Table	voraTargetAdaptedDataSet	object	Target VORA table where the Delta data is written.

Name	ID	Type	Description
Target ADL File	adlTargetObjectName	string	Target ADL file where the Delta data is written. It is required to use <timestamp> as part of the file name to ensure that each Delta batch has a unique file name.
Target SDL File	sdlTargetObjectName	string	Target ADL file where the Delta data is written. It is required to use <timestamp> as part of the file name to ensure that each Delta batch has a unique file name.
Target HDFS File	hdfsTargetObjectName	string	Target HDFS file where the Delta data is written. It is required to use <timestamp> as part of the file name to ensure that each Delta batch has a unique file name.
Target GCS File	gcsTargetObjectName	string	Target GCS file where the Delta data is written. It is required to use <timestamp> as part of the file name to ensure that each Delta batch has a unique file name.
Target S3 File	s3TargetObjectName	string	Target S3 where the Delta data is written. It is required to use <timestamp> as part of the file name to ensure that each Delta batch has a unique file name.
Target OSS File	ossTargetObjectName	string	Target OSS file where the Delta data is written. It is required to use <timestamp> as part of the file name to ensure that each Delta batch has a unique file name.
Target WASB File	wasbTargetObjectName	string	Target WASB file where the Delta data is written. It is required to use <timestamp> as part of the file name to ensure that each Delta batch has a unique file name.
Target Local File	path	string	A formatted string describing the output path and file name. If the root directory of the file is not provided, "/vrep/" is used as default root directory of the file. It is required to use <timestamp> as part of the file name to ensure that each Delta batch has a unique file name. Default: /vrep/<sourceTableName>_<timestamp>.csv
Target Write Mode	mode	string	The write mode for the target table during initial load. If set to <i>overwrite</i>, the table is dropped and recreated. Otherwise, the data from the source is appended. Default: "overwrite"
Target Format	format	string	Output file format: CSV, PARQUET, and ORC are supported for the cloud storages. The CSV that is produced conforms to the RFC-4180 specification, where the row delimiter is always CRLF and the escape character is always double quotes. Currently, only CSV is supported for local storage. Default: "CSV"
Target CSV Properties	additionalProperties_csv	object	Properties of the CSV output: - Column delimiter (ID columnDelimiter; type string): The field separator. Only the following characters are supported: ";", ":", " ", ":", and "TAB". Default: ";", - Header Included (ID csvHeaderIncluded; type boolean): Indicates whether each CSV output has a header as first line. Default: false

Name	ID	Type	Description
Delta Load Properties	deltaProperties	object	Additional properties for Delta processing: - Change Data Applier Mode (ID <code>cdcApplierType</code>; type string): See the Delta Processing section in the Flowagent Table Producer operator documentation. Default: "Track Change History" - Delta Graph Run (ID <code>deltaGraphMode</code>; type string): How Delta Processing is run. In <i>manual</i> mode, it counts the number of Delta rows in the source at the beginning of the job, and it stops after all rows have been fetched. In <i>polling interval</i>, it runs indefinitely. Default: "Polling Interval" - Capture Only Primary Keys (ID <code>captureOnlyPrimaryKeys</code>; type boolean): This option only affects the initial load. When enabled, the created triggers copy the primary keys only to this shadow table, which saves space but results in a slower replication. Default: false - Capture Only Primary Keys on Delete (ID <code>captureOnlyPrimaryKeysOnDelete</code>; type boolean): During a delete operation, the created triggers copy the primary keys only to the shadow table to save space and perform a quicker replication. - Capture Only New Values on Update (ID: <code>captureOnlyNewValueOnUpdate</code>; type boolean): During an update operation, the created triggers copy the updated value of the modified entry to save space and perform a quicker replication. - Max Polling Interval (ID <code>maxPollingInterval</code>; type number): When Delta Graph Run Mode is set to <i>polling interval</i>, this option defines the max interval on which the polling is performed in seconds. Default: 3600 - Delta Batch Size (ID <code>deltaBatchSize</code>; type number): The number of Delta rows fetched at a time. Default: 100000
Source table name alias	sourceTableNameAlias	string	The source table name alias, which can be used when there are long identifier issues in the database. When specified, it should be shorter than 28 characters. When not specified, the source table name is used instead. In this process, a helper table called <code>DI_CDC_ALIASES</code> is created in the database to store specified suffixes.

Input

Input	Type	Description
inTrigger	message	Port to trigger the operator.

Output

Input	Type	Description
result	message	The fully qualified name of the target object. It is sent to each consumed Delta batch.
Body	string	The fully qualified name of the target object.
Attribute: message.lastBatch	boolean	Boolean option that indicates the current batch is the last one.
Attribute: message.batchIndex	int	Indicates which batch is being processed, starting from 0.
Attribute: message.executionMode	string	Execution mode that was performed.
encoding	string	
error	message.error	Operator error. If mapped, the operator error is sent to this port and the graph remains running. If not mapped, the graph terminates with the operator error.

2.4 Converter

Related Information

[StreamToString Converter \[page 195\]](#)

[StringToStream Converter \[page 196\]](#)

[ToBlob Converter \[page 196\]](#)

[ToMessage Converter \[page 197\]](#)

[ToNumber Converter \[page 198\]](#)

[ToString Converter \[page 199\]](#)

[ToStringType Converter \[page 200\]](#)

2.4.1 StreamToString Converter

This operator parses a contiguous stream of strings and breaks them at platform-dependent line delimiters into separate output strings.

Configuration Parameters

Name	ID	Type	Description
Unescape	unescape	bool	If set to "True", the previously escaped "\" and "\n" have the escaping character removed (see StringToStream Converter [page 196]) Default: false

Input

Input	Type	Description
in	stream.*	The input stream to be parsed.

Output

Output	Type	Description
out	string.*	The strings parsed from the incoming stream.

2.4.2 StringToStream Converter

This operator converts incoming strings into a contiguous stream. A platform-dependent line delimiter is appended after each string written to the output stream.

Configuration Parameters

Name	ID	Type	Description
Escape	escape	bool	If set to True, the sequences \ (backslash) and \n (new line) are escaped. (See StringToStream Converter [page 196] .) Default: false

Input

Input	Type	Description
in	string.*	The string inputs.

Output

Output	Type	Description
out	stream.*	The contiguous stream of input parameters delimited by a platform-dependent line delimiter.

2.4.3 ToBlob Converter

The ToBlob Converter operator converts the input into a byte array. When data is received at inInterface, the following conversion is performed:

- primitive number types: a big-endian byte array representation.

- string: a standard conversion.
- vora message: the body of the message.

Configuration Parameters

None

Input

Input	Type	Description
<code>inInterface</code>	any	The input interface.

Output

Output	Type	Description
<code>outByteArray</code>	blob	The output byte array.

2.4.4 ToMessage Converter

The ToMessage Converter operator converts the input to a Modeler message. The input can be either a string representation of the entire message or a data of any type that will be placed in the body of a new message.

Configuration Parameters

None

Input

Input	Type	Description
inbody	any.*	When any data arrives in this port, a new message is created with the received data placed in its body. The attributes and encoding fields are left empty.
instring	string	The input string that represents the entire message. For example, the string: <code>{"message.attributes.encoding": "jpeg", "otherField": 3}</code> this part is the body' will be converted to a message where the body has the string "this part is the body", the attributes has the map {"otherField": 3}, and the bodyEncoding has the string "jpeg". The body will always become a string when this port is used.

Output

Output	Type	Description
out	message	The resulting Modeler message derived from the input.

2.4.5 ToNumber Converter

The ToNumber Converter operator converts types int64, uint64, float64, and string into other numeric types. When converting from float64, you can also choose the rounding behavior. If converting from a string, the content of the string must be numerical.

i Note

The current version of this operator does not handle representation overflow when converting, for example, from uint64 to int64 and will raise an error upon encountering this scenario.

Configuration Parameters

Name	ID	Type	Description
Rounding Behavior	roundingBehavior	string	How rounding will behave when converting from a float64 type. Default: "up"

Input

Input	Type	Description
input	any*	The input value, which can be of types int64, uint64, float64, and string.

Output

Output	Type	Description
outFloat	float64	The input value converted to type float64.
outInt	int64	The input value converted to type int64.
outUInt	uint64	The input value converted to type uint64.

2.4.6 ToString Converter

The ToString Converter operator converts the input to a string. When data is received at inInterface, the following conversion is performed:

- int: a standard conversion to string.
- float64: a floating point conversion to string.
- byte array: a standard conversion.
- vora message: the body of the message.

For an SAP Vora message received at inMessage, a JSON string that represents the headers is followed by the body in its string representation.

Configuration Parameters

None

Input

Input	Type	Description
<code>inInterface</code>	<code>any.*</code>	The input interface.
<code>inMessage</code>	<code>message</code>	The input SAP Vora message.

Output

Output	Type	Description
<code>outString</code>	<code>string</code>	The output string.

2.4.7 ToStringType Converter

This operator converts regular types to the specialized string types used for Scenes. In other words, it converts from string or message to the types `string.boolean`, `string.array` and `string.object`.

You can use multiple conversions at once, as the ports work independently.

Note that the operator doesn't actually check the validity of the content. If you want to convert to the type `string.array`, it's expected that the input will be a JSON stringified array, and the output type will work normally. This operator exists solely to allow connections from an unmatched type to the Scene types, as long as the content is already correctly formatted.

Configuration Parameters

Parameter	Type	Description
messageProp	string	The name of the property to be extracted from the message's body. If empty, will be ignored. Otherwise, the output the operator will be the value of the specified property, if the message is an object with properties. Default: ""

Input

Input	Type	Description
input	string	The input value. Expects a valid JSON-formatted string to perform a "type cast" to Scene specialized string types.
inMessage	message	The input message. Converts the entire body of the message to string.

Output

Output	Type	Description
outBoolean	string.boolean	The resulting string. Does not check the content, but supposed to contain a stringified boolean value.
outArray	string.array	The resulting string. Does not check the content, but supposed to contain a stringified array.
outObject	string.object	The resulting string. Does not check the content, but supposed to contain a stringified object.

2.5 Data Quality

Use the data quality operators to improve your data so you have the best information available to make business decisions and plans.

Related Information

[Anonymization \[page 202\]](#)

[Data Mask \[page 215\]](#)

[DQMm Address Cleanse \[page 236\]](#)

[DQMm Client \[page 245\]](#)

[DQMm Reverse Geo \[page 248\]](#)

[Person and Firm Cleanse \[page 251\]](#)

[Validation Rule \[page 259\]](#)

[Data Quality SQL Reference \[page 264\]](#)

2.5.1 Anonymization

Anonymization helps to gain statistically valid insights from your data while protecting the privacy of individuals.

When analyzing data, you must ensure privacy of personal or sensitive information. By removing information that directly identifies an individual, such as a Social Security number or a credit card number, you ensure a certain amount of privacy, but it could still lead to an individual's reidentification.

The Anonymization operator helps to create anonymized groups, where you set the minimum number of records within the group. You have several options for anonymizing data:

- **I-Diversity:** Specify the minimum number of distinct sensitive values that must be present in the equivalency class.
- **t-Closeness:** Specify how closely the equivalency class distributes values relative to the source table.
- **Mask or generalization:** Choose a column to anonymize. For example, if you group individuals into age brackets of 20-29, 30-39, and 40-49, then it's more difficult to reidentify an individual. When you mask or generalize multiple columns, then the ability to reidentifiy a person or business decreases.

As you use the Anonymization operator, you see the following terms:

- **Sensitive:** data that most individuals don't want known about them, for example, salary information or an illness.
- **Nonsensitive:** data that most individuals may not mind sharing, for example, the country they live in.
- **Identifier:** data that directly identifies individuals such as their name or Social Security number.
- **Quasi-identifier:** data that indirectly identifies and individual, especially when combined with other quasi-identifiers, such age, gender, and postcode.

i Note

Each business identifies how to categorize the data. What is categorized as nonsensitive for one business may be a quasi-identifier for another business. Any examples shown may be categorized differently based on your needs.

Configuration Parameters

Parameter	Type	Description
Label	String	Mandatory. Enter the name of the data mask operator.
Minimum rows in anonymized group	Integer	Mandatory. Specifies the minimum rows in a group. Groups with fewer records than the number specified are not output. When the group reaches the specified value, then the group is output. For example, when the operator receives more data, those groups that previously did not have enough records to output may reach the minimum value, and then are output. Any records meeting the criteria of other output groups are immediately added to those groups. Enter a value from 2 to 100. The larger the number you enter, the less likely the data can be reidentified, and a smaller number of records are output. The lower the number entered, the more likely the data can be reidentified and a larger number of records are output.
End of Data Flag	String	Optional. Specifies the name of the header attribute that indicates whether this input message is the end of the input dataset. When this value is set to <code>False</code> in the input message, the data accumulates until the application reads a header attribute value of <code>True</code> . After reading the <code>True</code> value, the application runs anonymization on all accumulated rows. If you remove the default value and leave the option blank, then anonymization is performed on the first input message, but subsequent input messages generate an error. If you enter an end of data flag that is absent or different from what is in the input message, then anonymization isn't run.
Date Format	String	Mandatory. Specifies the order in which month, day, and year elements appear in the input string. This value is used only when the day, month, or year in the input string is ambiguous.
Month Format	String	Mandatory. Specifies the format in which the randomized month is output when the software can't determine the output month format based on the input alone. For example, the options would output the following for the month of January: Short: Jan Full: January Numeric: 1

Parameter	Type	Description
Language	String	Mandatory. Specifies the language that the software uses when determining the output of an ambiguous input month string.
Century Threshold	Integer	Optional. Indicates whether a two-digit date is considered part of the 20th or 21st century. Enter a value from 0 to 99. For example, when set to 25, the dates with a 2-digit value from 00 to 25 result in the years 2000-2025. Dates with a 2-digit value of 26 to 99 result in the years 1926-1999.
Default Column Behavior	String	Mandatory. Define whether to output any columns that are not defined in the Column Definitions option.

Parameter	Type	Description
Column Definitions	Array	You can define the Anonymization operation on one or more columns. Each column has its own definition. Click + Add and complete the following options: - Column ID (string): Mandatory. This string uniquely identifies the column. It should match the ID or name of the column coming into the operator. - Column Designation (string): Mandatory. Specifies the categorization and any masking or generalization of this column: Sensitive: Data is output without modification, for example, height. You can choose to make the equivalency groups more secure by setting one or both of these options: <i>I-Diversity</i> and <i>t-Closeness</i>. - For <i>I-Diversity</i>, choose <i>none</i> to ignore the <i>I-Diversity</i> option. Choose <i>Distinct I-diversity</i> to enable this option and to configure the <i>I-Value</i>. Set the <i>I-Value</i> to a number from 2 through 100. This number specifies the minimum number of distinct sensitive values that must be present in the equivalency class to output the group. For example, if your equivalency class has four distinct values for height (127 cm, 132 cm, 160 cm, and 193 cm), and you set the I-Value to 5, then this equivalency class is not output. The equivalency class is output when the option is set to 4. - For <i>t-Closeness</i>, choose <i>none</i> to ignore the <i>t-Closeness</i> option. Choose <i>Equal distance t-closeness</i> to enable this option and to configure the <i>t-Value</i>. Set the <i>t-Value</i> to a number from 0.000001 to 1.0. Entering a number near 0.000001 means that the equivalency class closely distributes the value to the number of times the sensitive value appears in the source table. Entering a value closer to 1.0 means that the distribution is not as reflective to the occurrences in the source table. Nonsensitive: Data is output without modification, for example, hair color. Quasi-identifier: Data is output and used to form equivalence classes, for example, age. You can further mask or generalized the quasi-identifier columns. Mask: Mask all or a portion of the data with another character. For example, a Social Security number might output as ***-**-1234. Date Generalization: Output date ranges into groups. For example, divide subscribers into groups based on their birth dates, and label the era (such as Millennials, GenX, Baby Boomers) rather than using the actual birth date. Numeric Generalization: Output number ranges into groups. For example, output the records in a SALARY column that have values between

Parameter	Type	Description
		\$42,000 and \$125,000 into a group called Middle Class. ◦ <i>Do Not Modify</i>: Output data without masking or generalization. ◦ <i>Identifier</i>: Data can positively identify an individual and is not output, for example Social Security number.

Mask Options

Mask all or a portion of the data with another character. For example, a credit card number might output as
 ****-****-****-1234.

Parameter	Type	Description
Starting Position	String	Mandatory. Specifies whether masking starts at the beginning or end of the value.
Unmasked Length	Integer	Mandatory. Specifies the number of characters at the beginning or end of the value that should not be masked.
Masking Character	String	Mandatory. The character or number that replaces the characters in the input data, for example, "#" or "*".
Maintain Formatting	Boolean	Mandatory. • <i>True</i>: retains any special characters such as dashes, slashes, or periods, spaces between characters, and formatting in the output. For example, if you have a phone number that uses dashes, then the dashes are output. • <i>False</i>: replaces special characters and spaces with the designated masking character.

Date Generalization Options

Output date ranges into groups.

Parameter	Type	Description
Auto Range Scale	String	Mandatory. Defines the scale on which to base the auto range. - Not in Use: Indicates that you are not using auto range for the specified input column. This setting is appropriate when you complete the Range Definition options for the input column, or when you don't use this feature. To define the option, click + Add item. - Calendar Year: Groups records based on the calendar year. The software defines a calendar year as 1/1/yyyy to 12/31/yyyy. - Calendar Month: Groups records based on the calendar month. The software defines a calendar month as mm/01/yyyy to mm/eom/yyyy, where "eom" is end of month.
Minimum Date	String	Mandatory. Enter the lowest acceptable date in the range. i Note Enter a minimum or maximum value, or both. You can't leave both values empty.
Minimum Date Inclusive	Boolean	Mandatory. Select True when you want to include the minimum date. Select False when you don't want to include the minimum date in the results. For example, if you set the minimum value to 12/31/2020, then 12/31/2020 is included in the results when True is selected.
Maximum Date	String	Mandatory. Enter the highest acceptable date in the range. i Note Enter a minimum or maximum value, or both. You can't leave both values empty.
Maximum Date Inclusive	Boolean	Mandatory. Select True when you want to include the maximum date. Select False when you don't want to include the maximum date in the results. For example, if you set the maximum date to 06/30/2020, then dates through 06/29/2020 are included in the results when False is selected.
Replacement Value	String	Mandatory. To describe the group, enter a value.
Default Replacement Value	String	Optional. Value to output when the input value doesn't fall into any of the defined ranges.

Parameter	Type	Description
Auto Range Duration	Integer	Mandatory. Number of years or months to include in the range. i Note This option is visible only when Auto Range Scale is set to Calendar Month or Calendar Year.
Auto Range Start Date	String	Mandatory. Starting date in auto range. i Note This option is visible only when Auto Range Scale is set to Calendar Month or Calendar Year.
Auto Range End Date	String	Mandatory. Ending date in auto range. i Note This option is visible only when Auto Range Scale is set to Calendar Month or Calendar Year.
Auto Range Output Format	String	Mandatory. Determines the format of the output Auto Range Replacement Value. i Note This option is visible only when Auto Range Scale is set to Calendar Month or Calendar Year.
Auto Range Year Format	String	Mandatory. Specifies the number of digits to use for the year. Full Year outputs a four-digit number, for example, 2018. Short Year outputs a two-digit number, for example, 18. i Note This option is visible only when Auto Range Scale is set to Calendar Month or Calendar Year.
Auto Range Month Format	String	Mandatory. Determines the month format to use in the Auto Range Replacement Value. Full Text outputs the month name, for example, January. Short Text outputs the abbreviated month name, for example, Jan. Numeric outputs the number of the month, for example, 1 for January. i Note This option is visible only when Auto Range Scale is set to Calendar Month or Calendar Year.

Parameter	Type	Description
Auto Range Date Delimiter	String	Mandatory. Determines the delimiter to use in the Auto Range Replacement Value. i Note This option is visible only when Auto Range Scale is set to Calendar Month or Calendar Year.
Auto Range Numeric Format	String	Mandatory. Determines the numeric format to use in the Auto Range Replacement Value. i Note This option is visible only when Auto Range Scale is set to Calendar Month or Calendar Year.
Auto Range Enable Zero Pad	Boolean	Optional. Pads a one-digit number with zero when the format includes the month and day. For example, 1/5/2018 changes to 01/05/2018 when set to True. i Note This option is visible only when Auto Range Scale is set to Calendar Month or Calendar Year.
Auto Range Output Language	String	Optional. Determines the language to use in the Auto Range Replacement Value. This setting is applicable when the Month Format is set to Short Text or Full Text. i Note This option is visible only when Auto Range Scale is set to Calendar Month or Calendar Year.

Numeric Generalization Options

Output number ranges into groups. For example, output the records in an `AGE` column that have values between 13–19 into a group called `Teenager`. Specify the ranges to use for numeric variance. In the Numeric Generalization option, select [+ Add item](#).

i Note

You can add multiple items.

Parameter	Type	Description
Minimum Value	Decimal	Mandatory. Enter the lowest acceptable value in the range. Note Enter a minimum or maximum value, or both. You can't leave both values empty.
Minimum Value Inclusive	Boolean	Mandatory. Select True when you want to include the minimum value. Select False when you don't want to include the minimum value in the results. For example, if you set the minimum value to 30, then 30 is included in the results when True is selected.
Maximum Value	Decimal	Mandatory. Enter the highest acceptable value in the range. Note Enter a minimum or maximum value, or both. You can't leave both values empty.
Maximum Value Inclusive	Boolean	Required. Select True when you want to include the maximum value. Select False when you don't want to include the maximum value in the results. For example, if you set the maximum value to 50, then numbers through 49 are included in the results when False is selected.
Replacement Value	String	Optional. To describe the group, enter a value.
Default Replacement Value	String	Optional. Value to output when the input value doesn't fall into any of the defined ranges. For example, you might want to label those records as Exceptions.

Numeric Generalization Example

Let's say that you want to assign employees to one of three geographic areas based on their employee number. You would add three items and complete the options as follows.

Sample Code

```
Item 0
Minimum Value: 12000
Minimum Value Inclusive: True
Maximum Value: 17000
Maximum Value Inclusive: False
Replacement Value: Asia_Pac
Item 1
Minimum Value: 17000
Minimum Value Inclusive: True
Maximum Value: 23000
Maximum Value Inclusive: False
Replacement Value: Europe_Africa
Item 2
Minimum Value: 23000
Minimum Value Inclusive: True
Maximum Value: 28000
Maximum Value Inclusive: True
Replacement Value: Americas
```

Related Information

[Anonymized Groups Example \[page 211\]](#)

2.5.1.1 Anonymized Groups Example

Use anonymization to place masked data into match groups so that you can publish data without risking reidentification of sensitive data.

Before masking, the data might be unique. After masking several columns of identifying data with [Mask](#), [Date Generalization](#), or [Numeric Generalization](#), there will be duplicate records. In the Anonymization settings, you can choose the minimum number of records you want to output in a group. The groups that contain fewer records than the number you specify are not output. The larger the number you enter (from 2 through 100), the less likely the data can be reidentified and a smaller number of records are output. The lower the number entered, the more likely the data can be reidentified and a larger number of records are output.

Let's say that you work for an insurance company. You want to publish quarterly information about the cause of emergency room visits without compromising the patients' privacy.

Row ID	Patient ID	Name	Age	Postcode	Date of Visit	Issue
1	L1234-0987	James Smith	1	54601	05/13/2017	Ear infection
2	R5678-6543	Allison Zhou	26	54650	02/06/2017	Influenza
3	J2345-9876	Mia Vang	53	55190	12/18/2016	Heart attack
4	J6789-5432	Ben McCleary	68	55118	09/25/2016	Stroke
5	P7789-1212	Franz Gullikson	2	54603	04/15/2017	Ear infection
6	R0606-1223	Joseph Kas-wizki	4	54551	01/21/2017	Influenza
7	B7212-7306	Avijit Farooq	3	54601	02/02/2017	Influenza
8	R8675-3099	Alejandro Rodriguez	18	54650	11/27/2016	Concussion
9	J0673-1272	Aleksandra Kaminski	2	54603	04/17/2017	Ear infection
10	W1720-0825	Amanda Barns	4	54601	02/04/2017	Influenza

You can mask the data in many different ways. This table shows how we've defined each of the columns.

Column Name	Column Designation
Row ID	Nonsensitive

Column Name	Column Designation
Patient ID	Identifier
Name	Identifier
Age	Quasi-identifier Operation: Numeric Generalization • Minimum Value: 13 • Minimum Value Inclusive: True • Maximum Value: 19 • Maximum Value Inclusive: True • Replacement Value: Teenager • Default Replacement Value: other i Note Create more groups, defining ranges such as 0-5, 6-12, 20-29, 30-39, and so on.
Postcode	Quasi-identifier Operation: Mask • Starting Position: Start • Unmasked Length: 2 • Masking Character: * • Maintain Formatting: False
Date of Visit	Quasi-identifier Operation: Date Generalization Auto Range Scale: Not_In_Use Date Generalization Range Definition: • First range: ◦ Minimum Date: 01/01/2017 ◦ Minimum Date Inclusive: True ◦ Maximum Date: 03/31/2017 ◦ Maximum Date Inclusive: True ◦ Replacement Value: Q1 2017 • Second range: ◦ Minimum Date: 04/01/2017 ◦ Minimum Date Inclusive: True ◦ Maximum Date: 06/30/2017 ◦ Maximum Date Inclusive: True ◦ Replacement Value: Q2 2017 • Third range: ◦ Minimum Date: 07/01/2017

Column Name	Column Designation
	- Minimum Date Inclusive: True - Maximum Date: 09/30/2017 - Maximum Date Inclusive: True - Replacement Value: Q3 2017 - Fourth range: - Minimum Date: 10/01/2017 - Minimum Date Inclusive: True - Maximum Date: 12/31/2017 - Maximum Date Inclusive: True - Replacement Value: Q4 2017
	Default Replacement Value: Other
Issue	Sensitive

In this example, we removed the patient name and ID. We masked the last three digits of the postcode. We generalized the age and generalized the date of service from a specific day into a generalized quarter. We defined the Issue column as Sensitive. The resulting data set shows how the formerly unique records are more generic while still providing the necessary data.

Row ID	Age	Postcode	Date of Visit	Issue
1	Toddler	54***	Q2 2017	Ear infection
2	[20-29]	54***	Q1 2017	Influenza
3	[50-59]	55***	Q4 2016	Heart attack
4	[60-69]	55***	Q3 2016	Stroke
5	Toddler	54***	Q2 2017	Ear infection
6	Toddler	54***	Q1 2017	Influenza
7	[0-5]	54***	Q1 2017	Influenza
8	Teenager	54***	Q4 2016	Concussion
9	Toddler	54***	Q2 2017	Ear infection
10	Toddler	54***	Q1 2017	Influenza

Now that the data is masked, you can see how the data is placed into anonymized groups.

Age	Postcode	Date of Visit	Issue	Anonymization Group Size
Toddler	54***	Q1 2017	Influenza	3
[20-29]	54***	Q1 2017	Influenza	1
Toddler	54***	Q2 2017	Ear Infection	3
[50-59]	55***	Q4 2016	Heart attack	1
[60-69]	55***	Q3 2016	Stroke	1

Age	Postcode	Date of Visit	Issue	Anonymization Group Size
Teenager	54***	Q4 2016	Concussion	1

If you set the *Minimum rows in anonymized group* option to 3, then you could publish six of 10 records.

Row ID	Age	Postcode	Date of Visit	Issue
1	Toddler	54***	Q2 2017	Ear infection
5	Toddler	54***	Q2 2017	Ear infection
6	Toddler	54***	Q1 2017	Influenza
7	Toddler	54***	Q1 2017	Influenza
9	Toddler	54***	Q2 2017	Ear infection
10	Toddler	54***	Q1 2017	Influenza

Now, let's say that the issue in the first row is a concussion rather than an ear infection. Would the record still be published? Yes, because the Issue column isn't anonymized. The Age, Postcode, and Date of Visit columns are the only anonymized columns. Therefore, only the data in those columns are used in forming anonymized groups, not the data in the Issue column.

When the operator receives more data, those groups that previously did not have enough records to output may reach the minimum value, and then are output. Any records meeting the criteria of other output groups are immediately added to those groups.

If you would like to use the data in the Issue column for securing your data in addition to the masking and generalizations, you can set the *I-Diversity* or *t-Closeness* options.

Using the I-Diversity Option

To make equivalency classes more secure, you can set the *Distinct I-Diversity* option. This option defines a minimum number of distinct values that must be present, or the equivalency class is not output.

The *I-Diversity* option works with columns that have the *Column Designation* set to *Sensitive*. After selecting the *Distinct I-Diversity* option, set the *I-Value* to a positive value from 2 through 100. Only those equivalency classes that meet this minimum value are output.

Building on the previous example, the Issue column is the only column defined as *Sensitive*. There are two distinct values in the equivalency class: Ear Infection and Influenza. If you set the *I-Value* to 2, then this equivalency class is output. If you set this option to 3 or more, then it is not output because there aren't enough distinct values in the equivalency class.

Using the t-Closeness Option

You can use the *t-Closeness* option with the *I-Diversity* option. The *I-Diversity* option ensures that there are a certain number of similar records in an equivalency class. The *t-Closeness* option defines how closely the sensitive values in the equivalency class match the distribution of sensitive values in the source table.

Let's say that your equivalency class has 100 records in it. Let's also say that 60% of the records in the source table have an influenza issue. Setting the *t-Value* close to 0.000001 places approximately 60 influenza records in the equivalency class. If you set the *t-Value* close to 1.0, then there's a less equal distribution in the equivalency class relative to the source table.

2.5.2 Data Mask

Protect the personally identifiable or sensitive information by covering all or a portion of the data.

Some examples of personal and sensitive data include credit card numbers, birth dates, tax identification numbers, salary information, medical identification numbers, bank account numbers, and so on. Use data masking to support security and privacy policies, and to protect your customer or employee data from possible theft or exploitation.

Place the Data Mask operator toward the end of your graph to ensure that all columns that are to be masked have undergone processing by upstream operators. If you place Data Mask before other operators, the downstream operators may not process the actual data but rather the masked data. In some cases, the operator isn't able to process the columns at all if Data Mask replaced input data with blanks or a masking character such as "#".

Configuration Parameters

Parameter	Type	Description
Label	String	Mandatory. Enter the name of the data mask operator.
Seed	String	Optional. An alpha or numeric string. Set this option to mask the data in a way that ensures consistent output values each time the data is output. One seed value maintains referential integrity for the following variance types set up in the Data Mask transform: Number Variance, Date Variance, and Pattern Variance.
Date Format	String	Required. Specifies the order in which month, day, and year elements appear in the input string. This value is used only when the day, month, or year in the input string is ambiguous.
Month Format	String	Required. Specifies the format in which the randomized month is output when the software cannot determine the output month format based on the input alone.
Language	String	Required. Specifies the language that the software uses when determining the output of an ambiguous input month string.

Parameter	Type	Description
Century Threshold	String	Optional. Indicates whether a two-digit date is considered part of the 20th or 21st century. Enter a value from 0-99. For example, when set to 25, the dates with a 2-digit value from 00-25 result in the years 2000-2025. Dates with a 2-digit value of 26-99 result in the years 1926-1999.
Column Definitions		You can define the mask operation on one or more columns. Each column has its own definition. Click + Add and complete the following options: - Column ID (string): Required. This string uniquely identifies the column. It should match the ID or name of the column coming into the operator. - Operation (string): Required. Specifies the type of masking operation for this column: Mask, Pattern Variance, Numeric Variance, Numeric Generalization, Date Variance, or Date Generalization.

Mask Options

Mask all or a portion of the data with another character. For example, a credit card number might output as
 ****-****-****-1234.

Parameter	Type	Description
Starting Position	String	Required. Specifies whether masking starts at the beginning or end of the value.
Unmasked Length	String	Required. Specifies the number of characters at the beginning or end of the value that are not masked.
Masking Character	String	Required. The character or number that replaces the characters in the input data, for example, "#" or "*".
Maintain Formatting	String	Required. - True: retains any special characters such as dashes, slashes or periods, spaces between characters, and formatting in the output. For example, if you have a phone number that uses dashes, then the dashes are output. - False: replaces special characters and spaces with the designated masking character.

Pattern Variance Options

Mask an input substring with a specific pattern. For example, using the part number ABC123GHI, mask the first three characters with ZYW, mask the next three characters with 999, and preserve the final three characters as input. The result would be ZYW999GHI. In the Pattern Variance Definition option, select **+** [Add item](#).

Note

You can add multiple items.

Parameter	Type	Description
Variance Type	String	Required. Choose one of the options. • Default: Masks each applicable character with the like characters for alpha and numeric content. Retains spaces and special characters. • Preserve: Outputs the defined substring the same as it is input. • Character: Masks the defined substring by randomly placing each of the characters in the defined substring with the values that you specify in the Value setting. Retains spaces and special characters. • String: Masks the defined substring by randomly replacing the entire substring with values that you specify in the Value option. Does not retain spaces or special characters.
Starting Position	Integer	Required. A positive integer that indicates the character number where masking starts. Alpha, numeric, space, and other printable characters are included in the position count.
Length	Integer	Required. A positive integer that indicates the number of positions (characters) to mask.
Value	String	Required. When parsing the string for a value, it trims leading and trailing spaces from the value. You can maintain the spaces by surrounding the value in framing characters such as double quotes. For example, STRING, "John,Smith". If the value has double quotes, you can escape the double quotes using a backslash. For example, "\"Slim\"". If the value with framing characters has a backslash in it, the backslash can be escaped with an additional backslash. For example, "\\path".

Note

The Value option is visible only when Variance Type is set to Character or String.

Pattern Variance Regular Expression

Mask an input substring with a regular expression. Use a regular expression to break a piece of data into groups, and then mask each group using a different variance type. For example, you could have email addresses with variable lengths like `jon.gill@ai.com` and `stephanie.milkowski@business.com`. You can create a regular expression breaking the value into 4 groups: username, @ symbol, domain, and domain extension. Define each group and assign a variance type to each group.

Parameter	Type	Description
Regular Expression	String	Required. Enter a regular expression. If you have defined multiple groups using the parenthesis (), define each group by clicking the link next to Pattern Variance Regex Definition.
Pattern Variance Regex Definition	Array	Define how you want each regular expression group to be masked. Click + Add Item to create groups. i Note You can create multiple groups.
Group Number	String	Required. Enter the number of a group that was defined in the regular expression option.
Variance Type	String	Required. Choose one of the options. • Default: Masks each applicable character with the like characters for alpha and numeric content. Retains spaces and special characters. • Preserve: Outputs the defined substring the same as it is input. • Character: Masks the defined substring by randomly placing each of the characters in the defined substring with the values that you specify in the Value setting. Retains spaces and special characters. • String: Masks the defined substring by randomly replacing the entire substring with values that you specify in the Value option. Does not retain spaces or special characters.
Value	String	Required. When parsing the string for a value, it trims leading and trailing spaces from the value. You can maintain the spaces by surrounding the value in framing characters such as double quotes. For example, STRING, "John,Smith". If the value has double quotes, you can escape the double quotes using a backslash. For example, "\"Slim\"". If the value with framing characters has a backslash in it, the backslash can be escaped with an additional backslash. For example, "\\path". i Note The Value option is visible only when Variance Type is set to Character or String.

Numeric Variance Options

Output randomized numbers. For example, change the input salary of 50,000 to a random number between 45,000-55,000.

Parameter	Type	Description
Numeric Variance Type	String	Required. Define how you want to vary a number. - Percentage: Varies the data by a percentage that is within a calculated minimum and maximum range. - Fixed Number: Varies the data by a fixed number that is within a calculated minimum and maximum range. - Range: Varies the data that is greater than or equal to the user-defined minimum value and less than or equal to the user-defined maximum values that you set. You must set the minimum and maximum values.
Numeric Variance	Number	Required. Determines the number by which to randomize the input. Enter a value greater than zero.
Minimum Value	Number	Required. Determines the number by which to randomize the input. Enter the lowest value that can be output as a whole number or decimal. Negative decimal numbers are supported. For best results, set a realistic maximum value. i Note The Minimum Value is required only when the Numeric Variance Type is set to Range.
Maximum Value	Number	Required. Enter the highest value that can be output as a whole number or decimal. Negative numbers are supported. For best results, set a realistic maximum value. i Note The Minimum Value is required only when the Numeric Variance Type is set to Range.

Numeric Generalization Options

Output numbers ranges into groups. For example, output the records in an `AGE` column that have values from 13 to 19 into a group called `Teenager`. Specify the ranges to use for numeric variance. In the Numeric Generalization option, select [+ Add item](#).

i Note

You can add multiple items.

Parameter	Type	Description
Minimum Value	Integer	Enter the lowest acceptable value in the range. i Note Enter a minimum or maximum value, or both. You cannot leave both values empty.
Minimum Value Inclusive	String	Required. Select True when you want to include the minimum value. Select False when you do not want to include the minimum value in the results. For example, if you set the minimum value to 30, then 30 is included in the results when True is selected.
Maximum Value	Integer	Enter the highest acceptable value in the range. i Note Enter a minimum or maximum value, or both. You cannot leave both values empty.
Maximum Value Inclusive	String	Required. Select True when you want to include the maximum value. Select False when you do not want to include the minimum value in the results. For example, if you set the maximum value to 50, then numbers through 49 are included in the results when False is selected.
Replacement Value	String	Optional. Enter a value to describe the group.
Default Replacement Value	String	Optional. Value to output when the input value does not fall into any of the defined ranges. For example, you could label those records as Exceptions.

Numeric Generalization Example

Let's say that you want to assign employees to one of three geographic areas based on their employee number. You would add three items and complete the options as follows.

Sample Code

```

Item 0
Minimum Value: 12000
Minimum Value Inclusive: True
Maximum Value: 17000
Maximum Value Inclusive: False
Replacement Value: Asia_Pac
Item 1
Minimum Value: 17000
Minimum Value Inclusive: True
Maximum Value: 23000
Maximum Value Inclusive: False
Replacement Value: Europe_Africa
Item 2
Minimum Value: 23000
Minimum Value Inclusive: True
Maximum Value: 28000
Maximum Value Inclusive: True
Replacement Value: Americas
Default Replacement Value: Exceptions

```

Date Variance Options

Output randomized dates. For example, change the input date of 01/15/2017 to a random date between 01/01/2017 and 01/31/2017.

Parameter	Type	Description
Date Variance Type	String	Specifies how you want to vary a date. • Range: Varies the date within the user-defined minimum and maximum dates that you set. You must set the Minimum and Maximum date values. • Fixed Days: Varies the date by a fixed number of days that occur before or after the input date. • Fixed Months: Varies the date by a fixed number of months that occur before or after the input month. • Fixed Years: Varies the date by a fixed number of years that occur before or after the input year.
Date Variance	Number	Required. Determines the number of days, months, or years by which to randomize the input. The value must be greater than zero.
Minimum Date	String	Required for Range; optional for other types. Specify the minimum date allowed on output.
Maximum Date	String	Required for Range; optional for other types. Specify the maximum date allowed on output.

Date Generalization Options

Output date ranges into groups.

Parameter	Type	Description
Auto Range Scale	String	Required. Defines the scale on which to base the auto range. • Not in Use: Indicates that you are not using auto range for the specified input column. This setting is appropriate when you complete the Range Definition options for the input column, or when you do not use this feature. Click + Add item to further define the option. • Calendar Year: Group records based on the calendar year. The software defines a calendar year as 1/1/yyyy to 12/31/yyyy. • Calendar Month: Group records based on the calendar month. The software defines a calendar month as mm/01/yyyy to mm/eom/yyyy, where "eom" is end of month.

Parameter	Type	Description
Minimum Date	String	Enter the lowest acceptable date in the range. i Note Enter a minimum or maximum value, or both. You cannot leave both values empty.
Minimum Date Inclusive	String	Required. Select True when you want to include the minimum date. Select False when you do not want to include the minimum date in the results. For example, if you set the minimum value to 12/31/2020, then 12/31/2020 is included in the results when True is selected.
Maximum Date	String	Enter the highest acceptable date in the range. i Note Enter a minimum or maximum value, or both. You cannot leave both values empty.
Maximum Date Inclusive	String	Required. Select True when you want to include the maximum date. Select False when you do not want to include the minimum date in the results. For example, if you set the maximum date to 06/30/2020, then dates through 06/29/2020 are included in the results when False is selected.
Replacement Value	String	Required. Enter a value to describe the group.
Default Replacement Value	String	Optional. Value to output when the input value does not fall into any of the defined ranges.
Auto Range Duration	Integer	Required. Number of years or months to include in the range. i Note This option is visible only when Auto Range Scale is set to Calendar Month or Calendar Year.
Auto Range Start Date	String	Required. Starting date in auto range. i Note This option is visible only when Auto Range Scale is set to Calendar Month or Calendar Year.
Auto Range End Date	String	Required. Ending date in auto range. i Note This option is only visible Auto Range Scale is set to Calendar Month or Calendar Year.

Parameter	Type	Description
Auto Range Output Format	String	Required. Determines the format of the output Auto Range Replacement Value. i Note This option is visible only when Auto Range Scale is set to Calendar Month or Calendar Year.
Auto Range Year Format	String	Required. Specifies the number of digits to use for the year. Full Year outputs a four-digit number, for example, 2018. Short Year outputs a two-digit number, for example, 18. i Note This option is visible only when Auto Range Scale is set to Calendar Month or Calendar Year.
Auto Range Month Format	String	Required. Determines the month format to use in the Auto Range Replacement Value. Full Text outputs the month name, for example, January. Short Text outputs the abbreviated month name, for example, Jan. Numeric outputs the number of the month, for example, 1 for January. i Note This option is visible only when Auto Range Scale is set to Calendar Month or Calendar Year.
Auto Range Date Delimiter	String	Required. Determines the delimiter to use in the Auto Range Replacement Value. i Note This option is visible only when Auto Range Scale is set to Calendar Month or Calendar Year.
Auto Range Numeric Format	String	Optional. Determines the numeric format to use in the Auto Range Replacement Value. i Note This option is visible only when Auto Range Scale is set to Calendar Month or Calendar Year.
Auto Range Enable Zero Pad	String	Optional. Pad a one-digit number with zero when the format includes the month and day. For example, 1/5/2018 changes to 01/05/2018 when set to True. i Note This option is visible only when Auto Range Scale is set to Calendar Month or Calendar Year.

Parameter	Type	Description
Auto Range Output Language	String	Optional. Determines the language to use in the Auto Range Replacement Value. This setting is applicable when the Month Format is set to Short Text or Full Text.

i Note
 This option is visible only when Auto Range Scale is set to Calendar Month or Calendar Year.

Input

i Note

Only string and number are valid data types for input columns.

Input	Type	Description
Input	Message	The input is expected to be in JSON format.

Output

Output	Type	Description
Output	BLOB	The output is in JSON format.

Related Information

[Data Mask Examples \[page 225\]](#)

2.5.2.1 Data Mask Examples

Examples of Mask, Date Generalization, Date Variance, Numeric Generalization, Numeric Variance, and Pattern Variance masking options show how various settings affect the data.

Mask

Masking a portion of data:

- If you have a column for User_ID, the value is `Smith7887`, and you set the option to *Everything except the first 4 characters*, your output would be `Smitxxxx`.
- If you set the same option to *Everything except the last 2 characters*, your output would be `xxxxxxxx87`. If your masking character is left blank, your output would be `87`.

Using the *Maintain format* option:

- If you have a column for Phone1, the value is `800-555-1234`, and then select *Maintain format*, your output would be `xxx-xxx-xxxx`. Not selecting this option would output `xxxxxxxxxxxx`.
- If you have a column for Email1, the value is `john.smith@abc.com`, and you enable *Maintain format* and *Mask all characters in email name*, then your output would be `xxxxxxxxxx@xxx.xxx`.
- If you enable *Maintain format* and disable *Mask all characters in email name*, then your output would be `xxxx.xxxxx@xxx.xxx`.
- If you disable both *Maintain format* and *Mask all characters in email name*, then your output would be `xxxxxxxxxxxxxxxxxx`.

Date Generalization

Let's say that you want to divide subscribers into groups based on their birth dates, and that you want to label the era rather than using the actual birth date. Because the age groups are not an equal number of years, it would be best to manually define each range one at a time rather than using the auto range. The default masked value is defined as `Out of Range`, so that any records that do not belong in the defined ranges will output `Out of Range` as the masked value.

Minimum	Column Name	Maximum	Masked Value
1940.01.01<=	BIRTHDATE	<=1964.12.31	Baby Boomer
1965.01.01<=	BIRTHDATE	<=1976.12.31	Gen X
1977.01.01<=	BIRTHDATE	<=1995.12.31	Millennial

Date Variance

The following table shows several examples for one date value in a database: May 27, 1995. The examples help to illustrate how the date is calculated internally, and how it is output when the user-defined minimum and maximum values are used together.

Variance Type	Internally calculated date	Output value within this range	Notes
Fixed Days Variance: 10 Minimum date: <not set> Maximum date: <not set>	Min date: May 17, 1995 Max date: Jun 6, 1995	Min date: May 17, 1995 Max date: Jun 6, 1995	Max date: Jun 6, 1995
Fixed Days Variance: 100 Minimum date: Mar 1, 1995 Maximum date: Aug 31, 1995	Min date: Feb 9, 1995 Max date: Sep 4, 1995	Min date: Mar 1, 1995 Max date: Aug 31, 1995	The user-defined minimum and maximum dates are within the calculated minimum and maximum dates. Therefore, the user-defined dates are used.
Fixed Months Variance: 6 Min date: Jan 1, 1995 Max date: Dec 31, 1995	Min date: Nov 27, 1994 Max date: Nov 27, 1995	Min date: Jan 1, 1995 Max date: Nov 27, 1995	The user-defined minimum date is within the calculated variance, and is the value used for output. The maximum user-defined date is outside of the calculated variance. Therefore, the maximum calculated date is used.
Fixed Years Variance: 15 Min date: Jan 1, 1965 Max date: Dec 31, 2015	Min date: May 27, 1980 Max date: May 27, 2010	Min date: May 27, 1980 Max date: May 27, 2010	Both of the user-defined minimum and maximum dates are outside of the calculated variance. Therefore the calculated date is used.
Range Min date: Jan 1, 1965 Max date: Dec 31, 2015	n/a	Min date: Jan 1, 1965 Max date: Dec 31, 2015	Because there is no variance for range, only the user-defined minimum and maximum values are used.

Numeric Generalization

Let's say that you want to categorize your employee numbers by location. Rather than outputting the employee number, the defined masked values replace the value in the EMPNO column. Any numbers that do not fall into the range have the default masked value, `Outliers`.

Minimum	Column Name	Maximum	Masked Value
12000<=	EMPNO	<=16999	Asia_Pac
17000<=	EMPNO	<=29999	Europe_Africa
23000<	EMPNO	<=28999	Americas

Numeric Variance

The following table shows several examples for one salary value in a database: \$50,000. The examples help to illustrate how the value is calculated internally, and how it is output when the user-defined minimum and maximum values are used together.

Variance Type	Internally calculated value	Output within this range	Notes
Percentage	Min value: \$37,500	Min value: \$37,500	The output uses the internally calculated values, because the user-defined values are not specified.
Variance: 25	Max value: \$62,500	Max value: \$62,500	
Minimum value: <not set>			
Maximum value: <not set>			
Percentage	Min value: \$37,500	Min value: \$45,00	The user-defined minimum value is within the calculated variance, and is the value used for output.
Variance: 25	Max value: \$62,500	Max value: \$62,500	
Minimum value: 45,000			The maximum user-defined value is outside of the calculated variance. Therefore, the maximum calculated value is used.
Maximum value: 85,000			
Fixed number	Min value: \$47,500	Max value: \$52,500	The output uses the internally calculated values, because the user-defined values are not specified.
Variance: 2500	Max value: \$52,500	Max value: \$52,500	
Minimum value: <not set>			
Maximum value: <not set>			

Variance Type	Internally calculated value	Output within this range	Notes
Fixed number	Min value: \$47,500	Min value: \$47,500	Both the user-defined minimum and maximum values are outside of the calculated variance. Therefore, the calculated minimum and maximum values are used.
Variance: 2500	Max value: \$52,500	Max value: \$52,500	
Minimum value: 45,000			
Maximum value: 85,000			
Range	n/a	Minimum value: 55,000	Because there is no variance for range, only the user-defined minimum and maximum values are used.
Minimum value: 55,000		Maximum value: 95,000	
Maximum value: 95,000			

Pattern Variance

- **Default variance type:** The following table describes how the default pattern variance masks input characters with like characters.

Input character type	Mask values
Alphabetic	Masks lowercase alpha character with random lowercase alpha character. Masks uppercase alpha character with random uppercase alpha character.
Numeric	Masks each numeric character with a random numeric character from 0 up to and including 9.
Special characters or spaces	Does not mask special characters or spaces, but outputs them as they are input, unmasked. For example, when the input substring contains a dash (-), the default pattern variance keeps the dash in the output. When the input substring contains a space, the default pattern variance keeps the space in the output.

The following table shows the best practice use of default pattern variance.

Description	Best Practice	Example
Mask an entire input field using the default pattern variance.	Add a substring definition with a Default variance type.	Starting position: 1 Ending position: <slider all the way to the right>

Description	Best Practice	Example
Automatically apply the default pattern variance to substrings of a mapped input column that are not defined.	Define input field substrings using one or more of the other pattern variance types, and leave portions of the input value undefined.	Definition 1: Definition type: Preserve Starting position: 1 Ending position: 3 Definition 2: Definition type: String Starting position: 4 Ending position: 5 Value range min-max: 20-25 Undefined range: 6 through the end of the value Results Definition 1: Preserves the characters in positions 1-3 Definition 2: Masks the entire substring (position 4-5) with a number that is included in the minimum and maximum range (in this case 20-25) Undefined: Masks position six through the end of the field with the default pattern variance.

- **Preserve variance type:** The application outputs the defined substring as it was input, with no masking.

i Note

The default pattern variance is applied to any undefined portions of the input column. Undefined portions are the sections of the input column that have not been defined with preserve, character, or string pattern variance.

The following table contains an example of the preserve pattern variance.

Strategy	Settings	Example input and output	Notes
Preserve the unit identification number in each record. Mask the rest of the value with the default pattern variance.	Undefined: Character in position 1 Definition: Definition type: Preserve Starting position: 2 Ending position: 3 Value: <blank> Undefined: Characters 4 through the end of the value	Input value: B12 : G350 Possible output values, preserved characters in bold. A12 : N799 F12 : M127	Undefined: Masks the first position with a like character using the default pattern variance. Definition: Preserves position two and three with the preserve pattern variance (in this case the numbers 12). Undefined: masks the fourth position to the end of the string using the default pattern variance. The colon in the input field (character four) is included in the undefined portion. The software outputs the colon as it was input based on the default pattern variance definition.

- Character variance type:** The application masks each character in the defined substring with a character from the Value option.
The Value option can include individual uppercase or lowercase alpha characters, numeric characters from 0 to 9, ranges of alpha characters, or ranges of numeric characters (using numbers from 0 to 9), spaces, special characters (such as @, #, _, &), or any combination of these.

i Note

Each alpha and numeric value must be one character in length. Also, you can use only one character for minimum and maximum values when using a value range.

Special characters are output as they are input, without masking them, when they are present in a defined substring for character pattern variance.

The following table contains an example of the character pattern variance.

Strategy	Settings	Example input and output	Notes
Mask an identification code with specific alpha or numeric values, and apply the default pattern variance to the remaining portion of the value.	Undefined: character in position 1 Definition: Definition type: Character Starting position: 2 Ending position: 3 Value: J-I B W-Y 2 Undefined: characters 4 through the end of the value	Input value: 123a Possible output values, masked characters in bold. 8 K Bx 32 W t	Undefined: Masks the first position with a like character using the default pattern variance. Definition: Masks position two and three using the character pattern variance and randomly chooses a character specified in the Value field to mask each position. Undefined: Masks the fourth position to the end of the string using the default pattern variance.

- **String variance type:** The application masks the entire defined substring with a random character or string from the [Value](#) option.

The [Value](#) option can include one or more alpha or numeric characters (such as MILK or 2458), spaces, special characters (such as @, #, _, &), numeric ranges, or any combination of these in a list in the [Value](#) option.

The application counts all alphanumeric characters, spaces, and special characters when it determines the substring length. However, the application does not retain the special characters or spaces in the output when they are present in a defined substring for string pattern variance.

The following table contains an example of the string pattern variance.

Strategy	Settings	Example input and output	Notes
Preserve the product code, but mask the type of milk (white, chocolate, soy, and so on) with the general term MILK.	Definition 1: Definition type: Preserve Starting position: 1 Ending position: 5 Value: <blank> Definition 2: Definition type: String Starting position: 6 Ending position: <end of column> Value: MILK	Input value: 5428-WTMLK 5429-SOYMLK Possible output values, string characters in bold. 5428- MILK 5429- MILK	Definition 1: Preserves the first through the fifth positions, including the dash, as part of preserve pattern variance. The dash is output as it was input because it is included in the preserve pattern variance. Definition 2: Masks position six to the end of the field with the value MILK.

Strategy	Settings	Example input and output	Notes
Include a zero to the left of any number in a range (to the left of the lower or higher number or to both numbers) so the mask value is left-padded with zeros for the length of the substring.	Definition 1: Definition type: String Starting position: 1 Ending position: 5 Value: 01–8 999 Undefined: position 6 through the end of the column.	Input value: 04–a1099 Possible output values, string characters in bold. 00003502 999832	Definition 1: Outputs the first through the fifth characters with a number from 1 up to and including 8, or the number 999. When the application chooses a number from the range as a mask value, it zero-pads the number to the left so the masked value is the length of the defined substring (5 characters). The application does not zero-pad the number 999 because it does not contain a leading zero in the Value option. The application includes the dash in the input field in the position count for the substring. However, the application does not output the dash as part of the string pattern variance definition. Undefined: The application applies the default pattern variance to the undefined portion of the input field, character six to the end of the field, and replaces each numeric value with a random value from 0 to 9.

i Note

Zero-padding numbers in a range are applicable for string pattern variance only.

- Examples of various pattern variance types by showing example definition option settings, input values, and possible output values.

Strategy	Settings	Example Input and Output	Notes
Mask the weight and the unit of measure from a product code, but preserve the product type.	Definition 1: Definition type: Preserve Starting position: 1 Ending position: 3 Value: <blank> Undefined: Position 4 and 5. Definition 2: Definition type: String Starting position: 6 Ending position: 9 Value: GAL qt pt oz CUP	Input value: MLK12CUP Possible output values: MLK63GAL MLK18pt MLK04oz	Definition 1: Preserves the first three positions using the preserve pattern variance. Undefined: Masks the fourth and fifth positions using the default pattern variance, which replaces each numeric character with a value from 0-9. Definition 2: Masks the sixth through the eighth positions with one of the values listed using the string pattern variance. In some cases, the software replaces a 3-character substring with a 2-character value.
Mask the product type and the weight from a product code, but preserve the unit of measure.	Definition 1: Definition type: String Starting position: 1 Ending position: 3 Value: ALMLK SOYMLK RCEMLK WTMLK CHMLK Definition 2: Definition type: String Starting position: 4 Ending position: 5 Value: 01-12 32 16 Definition 3: Definition type: Preserve Starting position: 6 Ending position: <end of column> Value: <blank>	Input value: MLK12CUP Possible output values: WTMLK32CUP RCEMLK16CUP ALMLK08CUP	Definition 1: Masks the first three positions using one of the values specified for string pattern variance. The application masks the 3-character substring with values that may be longer than 3 characters. Definition 2: Masks the fourth and fifth positions using the string pattern variance. The first value listed in the Value option for Definition 2 is a range beginning with a zero-padded number. This ensures that the mask value is the length of the defined substring, 2 characters. Definition 3: Preserves the sixth through the eighth positions.

Strategy	Settings	Example Input and Output	Notes
Mask the number of paper sheets per package, and the type of packaging from the product description column.	Definition 1: Definition type: Character Starting position: 1 Ending position: 4 Value: 0-9 Undefined: Position 5. Definition 2: Definition type: String Starting position: 6 Ending position: <end of column> Value: Ream Case Pack Box	Input value: 1500/Ream Possible output values: 0950/Case 8945/Box 2639/Pack	Definition 1: Masks the first through the fourth positions with a number from 0 to 9. The user could leave the first through the fifth position undefined so the application masks the substring using the default pattern variance to get similar output values. The forward slash would be output as part of the substring in this case. Undefined: Outputs the forward slash (/) character in the fifth position using the default pattern variance (maintains special characters on output). Definition 2: Masks the sixth position to the end of the column with one of the character strings listed in the Value option.

Strategy	Settings	Example Input and Output	Notes
Mask the school district, the state, and the enrollment number. Preserve the type of school.	Definition 1:	Input value:	Definition 1: Masks position one to three with the string DST.
	Definition type: String	INDNE7321MID	
	Starting position: 1	ANMA7321HIGH	
	Ending position: 3	SNBCA7321ELEM	Definition 2: Masks the fourth and the fifth position with the string ST.
	Value: DST	Possible output values:	
	Definition 2:	DSTST3829MID	Undefined: Masks position six through nine with the default pattern variance.
	Definition type: String	DSTST5784HIGH	
	Starting position: 4	DSTST0789ELEM	Definition 3: Preserves the tenth position to the end of the column.
	Ending position: 5		
	Value: ST		
	Undefined: Position 6, 7, 8, and 9.		
	Definition 3:		
	Definition type: Preserve		
	Starting position: 10		
	Ending position: <end of column>		
	Value: <blank>		

Note

The mask out variance could also mask the fields in this example. However, with pattern variance, you can distinguish between parts of the whole string and have more control over the mask values.

Pattern Variance Regular Expression

Using a regular expression is useful when the data values are not a specific length. You can use regular expressions on any type of data. In this example, we'll use email addresses. Let's say that you have these email addresses: jon.gill@ai.com and stephanie.milkowski@business.com. You can create a regular expression breaking the value into 4 parts: username, @ symbol, domain, and domain extension. Define each group and assign a variance type to each group. Here is an example of the regular expression that you can use: `^([^\@]*) (\@) ([^\.]*) (.*)$`. First, let's break down the regular expression.

Part of Regular Expression	Description
<code>^</code>	The first caret <code>^</code> identifies the beginning of the string.
<code>([^\@] *)</code>	This is the first group of the expression identified by the parentheses <code>()</code> . The brackets, caret and @ sign, <code>[^\@]</code> , identifies everything up to the @ sign. The asterisk, <code>*</code> , identifies all instances of the preceding token. In this case, the @ sign.
<code>(\@)</code>	The second group identifies the @ symbol

Part of Regular Expression	Description
([^ .] *)	The third group identifies the period between the domain and the domain extension. The [^ .] identifies everything up to the period. The asterisk identifies all instances of the preceding token. In this case, the period.
(.*)	The fourth group identifies the the content after the period.
\$	The dollar sign \$ identifies the end of the string.

Now, let's see how defining each group with a different variance type looks. The variance types work the same as the Pattern Variance options for Default, Character, Preserve, and String. The input address, `stephanie.milkowski@business.com`, becomes `linqqltco.tcsdxoldc@ABC.jmz` with the these settings.

Group Number	Variance Type	Value	Input and Output data
1	Character	a-z	Input: stephanie.milkowski Output: linqqltco.tcsdxoldc
2	Preserve	n/a	@
3	String	ABC 123	ABC
4	Default	n/a	jmz

Here is another example that changes `jon.gill@ai.com` to `fev.hesx@123.net` with these settings.

Group Number	Variance Type	Value	Input and Output data
1	Character	a-z	Input: jon.gill Output: fev.hesx
2	Preserve	n/a	@
3	String	ABC 123	123
4	String	com gov net org	net

With Pattern Variance Regular Expression, you don't have to know the starting position or length of a piece of data. For those times when the data is of the same length and structure, you can use the Pattern Variance option.

2.5.3 DQMm Address Cleanse

With the DQMm Address Cleanse operator, you can prepare address cleanse requests to be sent to the SAP Data Quality Management, microservices for location data.

SAP Data Quality Management, microservices for location data offers cloud-based microservices for address cleansing, geocoding, and reverse geocoding. You can embed address cleansing and enrichment services within any business process or application so that you can quickly reap the value of complete and accurate address data.

This operator will create requests from your data that can then be passed to the DQMm Client operator.

Configuration Parameters

Parameter	Type	Description
configurationName	string	The name of the configuration to use if configSource is specified as DQM Microservices.
outputFields	string	The list of result field names separated by commas that specify the fields from the service that will be output. Default: "std_addr_address_delivery, std_addr_locality, std_addr_region, std_addr_postcode_full, std_addr_country_2char" The original input fields will be included in the output. If an output field name has the same name as an input field, then the input field will be overwritten on output. If you wish to retain the original input, you should map the output field to a different name.

Parameter	Type	Description
processingMode	string	- empty: Default value. Use the default service value or the value present in the configuration if one is specified. - both: The service cleanses address data and appends geo-location coordinates, provided the request includes output fields from both processes. - addressOnly: The service performs only cleansing address data, not appending geo-location coordinates. If the request includes output fields for the geo-location process, the values for those fields will be empty in the response. In this mode, a transaction is counted for the address, but not for the geo-location. - geoOnly: The service performs only appending geo-location coordinates, not cleansing address data. If the request includes output fields for the cleansed address, the values for those fields will be empty in the response. In this mode, a transaction is counted for the geo-location, but not for the address. This mode should be used only when the address data sent in the request has previously gone through address cleansing, or when you have high confidence in the quality of the address data. Default: "both"
casing	string	(empty) : Default value. Use the default service value or the value present in the configuration if one is specified. - mixed : Use mixed case. Example: 100 Main St - upper : Use all uppercase. Example: 100 MAIN ST Default: "mixed"

Parameter	Type	Description
diacritics	string	• (empty) : Default value. Use the default service value or the value present in the configuration if one is specified. • include: Include diacritical characters. Example: Münchner Str 100 • remove: Remove diacritical characters. Example: Muenchner Str 100 Default: "include"

Parameter	Type	Description
streetFormat	string	• (empty): Default value. Use the default service value or the value present in the configuration if one is specified. • countryCommonStyle: Abbreviate or use the full form of street address components based on what is most common for each country. See countryCommonStyle. • abbreviateNoPunctuation: Abbreviate street type, street prefix or suffix, and secondary designator. No punctuation is inserted. Example: 100 N Main St Ste 300 • abbreviateWithPunctuation: Abbreviate street type, street prefix or suffix, and secondary designator. Punctuation is inserted for each abbreviated word. Example: 100 N. Main St. Ste. 300 • expand: Use the full form of street type, street prefix or suffix, and secondary designator. Example: 100 North Main Street Suite 300 • expandPrimaryAbbreviateSecondaryNoPunctuation: Use the full form of street type and street prefix or suffix; abbreviate secondary designator. No punctuation is inserted. Example: 100 North Main Street Ste 300 • expandPrimaryAbbreviateSecondaryWithPunctuation : Use the full form of street type and street prefix or suffix; abbreviate secondary designator. Punctuation is inserted for each abbreviated word. Example: 100 North Main Street Ste. 300 Default: "countryCommonStyle"

Parameter	Type	Description
postalFormat	string	(empty): Default value. Use the default service value or the value present in the configuration if one is specified. - countryCommonStyle: Abbreviate or use the full form of the postbox designator based on what is most common for each country. See countryCommonStyle. - abbreviateNoPunctuation: Abbreviate the postbox designator. No punctuation is inserted. Example: PO Box 100 - abbreviateWithPunctuation: Abbreviate the postbox designator. Punctuation is inserted for each abbreviated word. Example: P.O. Box 100 - expand: Use the full form of the postbox designator. Example: Post Office Box 100 Default: "countryCommonStyle"
regionFormat	string	(empty): Default value. Use the default service value or the value present in the configuration if one is specified. - countryCommonStyle: Abbreviate or use the full form of the region based on what is most common for each country. See countryCommonStyle. - abbreviate: Abbreviate the region. Example: CA - expand: Use the full region name. Example: California Default: "countryCommonStyle"

Parameter	Type	Description
scriptConversion	string	• (empty): Default value. Use the default service value or the value present in the configuration if one is specified. • none: Preserve the script. Do not convert. • convertToLatin: Convert the script to Latin. Supported only for the following countries and regions: ◦ China ◦ Taiwan ◦ Republic of Korea ◦ Russian Federation Default: "preserve"

Parameter	Type	Description
minAssignmentLevel	string	• (empty): Default value. Use the default service value or the value present in the configuration if one is specified. • none: The service charges for the transaction and returns cleansed address results provided the address meets the billable transaction criteria, regardless of the address assignment level. • city: The service charges for the transaction and returns cleansed address results when the address assignment level is city (L1) or better. When the address assignment level is lower than city, the service does not charge for the transaction, and it returns address results that are formatted but not corrected. • street: The service charges for the transaction and returns cleansed address results when the address assignment level is street (PN) or better. When the address assignment level is lower than street, the service does not charge for the transaction, and it returns address results that are formatted but not corrected. • houseNumber: The service charges for the transaction and returns cleansed address results when the address assignment level is house number (PR) or better. When the address assignment level is lower than house number, the service does not charge for the transaction, and it returns address results that are formatted but not corrected. Default: "none"

Parameter	Type	Description
geoAssign	string	(empty): Default value. Use the default service value or the value present in the configuration if one is specified. - best: Returns geo-location coordinates that represent points of the actual address. When this fine level of assignment is not possible, then return points that represent the centroid point of the geometric shape represented by part or the entire postcode. When that is not possible, then return points that represent the centroid point of the geometric shape represented by the city. - houseNumberOnly: Returns geo-location coordinates that represent points of the actual address. For each address that this fine level of assignment is not possible, then do not return geo-location coordinates at all. Default: "best"
suggestionSuppressLevel	string	(empty): Default value. Use the default service value or the value present in the configuration if one is specified. - none: The service returns suggestions when there is ambiguity at any level when matching the input address with reference data. - secondary: The service returns suggestions when there is ambiguity when matching the input postcode, city, region, or primary portion of the street address with reference data. However, the suggestions feature is suppressed when there is ambiguity when matching the secondary portion of the street address, such as apartment, suite, unit, or floor. Default: "none"

Parameter	Type	Description
generateSuggestionList	string	Indicates whether it is possible for the microservice to generated suggestion lists. Default: "no"
suggestionReplyField	string	The field within the data to be used as a suggestion reply when performing address cleansing with suggestion lists enabled.
passThroughField	string	The field within the data to be returned unchanged in the response.

Input

Input	Type	Description
input	message	The input expected to be in JSON format.

Output

Output	Type	Description
output	message	The output is in JSON format.

2.5.4 DQMm Client

The DQMm Client operator is a convenient client for sending requests to the Data Quality Management, microservices for location data (DQM microservices). DQM microservices is a service hosted on the SAP Cloud Platform (SCP). For more information on DQM microservices, view *Data Quality Management, Microservices for Location Data*.

Data Quality Management microservices supports two types of authentication schemes. These include basic and OAuth authentication schemes.

Configuration Parameters

Parameter	Type	Description
host	string	The service's host optionally including the port.
includeResponseHeaders	string	A list of response header names that should be included in the output message. Overridden by header <code>openapi.include_response_headers</code>. Default: ""
authScheme	string	The security scheme. The possible values are "basic" and "oauth2". Overridden by header <code>openapi.auth_scheme</code>. Default: "oauth2"
user	string	The user name used by "basic" security scheme. Overridden by header <code>openapi.user</code>. Default: ""
password	string	The password used by "basic" security scheme. Overridden by header <code>openapi.user</code>. Default: ""
oauth2Flow	string	The flow used by "oauth2" security scheme. The only possible value is "application". Default: "application"
oauth2TokenUrl	string	The token URL used by "oauth2" security scheme. Default: ""
oauth2ClientId	string	The client ID used by "oauth2" security scheme. Default: ""
oauth2ClientSecret	string	The client secret used by "oauth2" security scheme. Default: ""

Parameter	Type	Description
oauth2AdditionalHeaders	string	Optional headers added in a token request, given as <code>\{"header1": "value", ... \}</code> . Default: ""
useCsrfToken	bool	If set to true, the client automatically retrieves a CSRF-token at its first request and uses it in its subsequent requests. Default: false
tlsCertificate	string	The file path to the TLS certificate file. Default: ""
tlsKey	string	The file path to the TLS private key file. Default: ""
tlsCa	string	The file path to the TLS certificate authority file. Default: ""
tlsSkipVerify	bool	If set to true, the client does not verify the server's certificate chain nor the host name. Default: false

Input

Input	Type	Description
in	message	Input message, where parameters are set in the headers and the optional content in body.

Output

Output	Type	Description
out	message	Output message, where parameters are set in the headers and the optional content in body.

Related Information

[Data Quality Management, Microservices for Location Data](#)

2.5.5 DQMm Reverse Geo

With the DQMm Reverse Geo operator, you can prepare address cleanse requests to be sent to the SAP Data Quality Management, microservices for location data.

SAP Data Quality Management, microservices for location data offers cloud-based microservices for address cleansing, geocoding, and reverse geocoding. You can embed address cleansing and enrichment services within any business process or application so that you can quickly reap the value of complete and accurate address data.

This operator will create requests from your data that can then be passed to the DQMm Client operator.

Configuration Parameters

Parameter	Type	Description
outputFields	string	The list of result field names separated by commas that specify the fields from the service that will be output. Default: default: "geo_search_results, geo_search_results_count, geo_info_code, geo_info_code_msg" The original input fields will be included in the output. If an output field name has the same name as an input field, then the input field will be overwritten on output. If you wish to retain the original input, you should map the output field to a different name.
radius	numberA	One way to reduce the number of addresses returned is by including a maximum distance, which represents the radius of a circle with center at the geo-location coordinates sent in the request. All returned addresses are within the limits of this circle. The value for radius must be a positive number. If a radius is between 0 and 1, prefix the number with a zero (for example, 0.4). Default: 1.0

Parameter	Type	Description
distanceUnit	string	Either kilometers or miles. Default: "kilometers" For example, to obtain a list of addresses within 10 km from the geo-location coordinates, set: <pre>radius = 10.0 distanceUnit = kilometers</pre> To obtain a list of addresses within a half mile, set: <pre>radius = 0.5 distanceUnit = miles</pre> When omitted from the request, the default distance is 1 kilometer. The maximum distance supported by the service is 111 kilometers or 68.97 miles, which represents one degree of latitude. If the request is sent with a distance greater than the maximum, then the service's maximum value is used.
maxRecords	number	A second way to reduce the number of addresses returned is by including a maximum number of records. The value for maxRecords may be between 1 and 100, sent as a data type of integer. When omitted from the request, the default maximum number of records is 100. Default: 100
passThroughField	string	The field within the data to be returned unchanged in the response.

Input

Input	Type	Description
input	message	The input expected to be in JSON format.

Output

Output	Type	Description
output	message	The output is in JSON format.

2.5.6 Person and Firm Cleanse

The Person and Firm Cleanse operator identifies the names of people and firms even when both types of names are in the same column.

Sometimes person and firm (business organization) data is listed in one column, such as a Customer column. Let's say that you own a bakery. You deliver your baked goods to businesses such as grocery stores as well as to people in their homes. In your data, you have a Customer column that contains both the names of businesses and of individual people.

In some instances, businesses have names that resemble peoples' names. For example, a person can be named Misty Green, but that can also be the name of a plant or an agricultural watering company.

Whether the person or firm data is in a single column or in separate person and firm columns, the Person and Firm Cleanse operator can help differentiate these types of data by comparing the name to a list of firm name data.

Configuration Parameters

Name	ID	Type	Description
Cleanse	cleanse	String	Mandatory. When a country field is input to the Person and Firm Cleanse operator, then the person, title, firm, and person-or-firm data is cleansed according to regional standards of the input country. Use this setting to select which language or region domain you want to use by default when cleansing data for records that have a blank country or for all records when a country field is not available. The Global domain is a special content domain that contains all variations and their associated properties. If a variation is not associated with domain-specific information, the Global domain serves as the default domain. Default: "EN_US - English (United States & Canada)"

Name	ID	Type	Description										
Output		String	Mandatory. certain output fields and formatting options are automatically set according to regional standards, based on the selected output format. When a country field is input to the Person or Firm Cleanse operator, then the person, title, firm, and person-or-firm data is output according to regional standards of the input country. Use this setting to select the cultural domain you want to use by default when cleansing data for records that have a blank country, or for all records when a country field isn't available. For example, when selecting one of the English domains, if you output person name data to discrete fields, the first name is output to First Name, the middle name to Middle Name, and the full last name to Last Name (nothing is output to Last Name 2), and if you output to the composite Person field, the name is ordered as first name - middle name - last name - maturity postname - honorary postname with a space between each word. <table><thead><tr><th>Input</th><th>Output</th></tr></thead><tbody><tr><td>Juan Carlos Sánchez Cruz Jr. PhD</td><td>Discrete field output: • First Name: Juan • Middle Name: Carlos • Last Name: Sánchez Cruz • Last Name2: n/a • Maturity Postname: Jr. • Honorary Postname: PhD </td></tr><tr><td></td><td>Composite field output: Person: Juan Carlos Sánchez Cruz Jr. PhD</td></tr></tbody></table> When selecting one of the Spanish domains to output to discrete fields, the system outputs the paternal last name to Last Name and the maternal last name to Last Name 2. <table><thead><tr><th>Input</th><th>Output</th></tr></thead><tbody><tr><td>Juan Carlos Sánchez Cruz Jr. PhD</td><td>Discrete field output: • First Name: Juan • Middle Name: Carlos • Last Name: Sánchez • Last Name2: Cruz • Maturity Postname: Jr. </td></tr></tbody></table>	Input	Output	Juan Carlos Sánchez Cruz Jr. PhD	Discrete field output: • First Name: Juan • Middle Name: Carlos • Last Name: Sánchez Cruz • Last Name2: n/a • Maturity Postname: Jr. • Honorary Postname: PhD		Composite field output: Person: Juan Carlos Sánchez Cruz Jr. PhD	Input	Output	Juan Carlos Sánchez Cruz Jr. PhD	Discrete field output: • First Name: Juan • Middle Name: Carlos • Last Name: Sánchez • Last Name2: Cruz • Maturity Postname: Jr.
Input	Output												
Juan Carlos Sánchez Cruz Jr. PhD	Discrete field output: • First Name: Juan • Middle Name: Carlos • Last Name: Sánchez Cruz • Last Name2: n/a • Maturity Postname: Jr. • Honorary Postname: PhD												
	Composite field output: Person: Juan Carlos Sánchez Cruz Jr. PhD												
Input	Output												
Juan Carlos Sánchez Cruz Jr. PhD	Discrete field output: • First Name: Juan • Middle Name: Carlos • Last Name: Sánchez • Last Name2: Cruz • Maturity Postname: Jr.												

Name	ID	Type	Description								
			<table><tr><th>Input</th><th>Output</th></tr><tr><td></td><td> - Honorary Postname: PhD </td></tr></table> When selecting the Chinese domain, if you output to discrete fields, the system outputs the given name to First Name and the family name to Last Name. Nothing is output to Middle Name or Last Name 2. If you output to the composite Person field, the name is ordered as last name - first name without any spaces between the words. <table><tr><th>Input</th><th>Output</th></tr><tr><td>Mie Ping Chen MD</td><td>Discrete field output: - First Name: Mie Ping - Middle Name: n/a - Last Name: Chen - Last Name2: n/a - Maturity Postname: n/a - Honorary Postname: MD Composite field output: Person: Chen Mie Ping MD</td></tr></table> The valid values are the same as Cleanse Domain, but you can select one domain only, and Global is not an option. Default: "EN_US - English (United States & Canada)"	Input	Output		Honorary Postname: PhD	Input	Output	Mie Ping Chen MD	Discrete field output: - First Name: Mie Ping - Middle Name: n/a - Last Name: Chen - Last Name2: n/a - Maturity Postname: n/a - Honorary Postname: MD Composite field output: Person: Chen Mie Ping MD
Input	Output										
	Honorary Postname: PhD										
Input	Output										
Mie Ping Chen MD	Discrete field output: - First Name: Mie Ping - Middle Name: n/a - Last Name: Chen - Last Name2: n/a - Maturity Postname: n/a - Honorary Postname: MD Composite field output: Person: Chen Mie Ping MD										
Casing		String	Mandatory: Specifies the casing format. - Mixed Case: Converts data to initial capitals. For example, if the input data is JOHN MCKAY, then the data is output as John McKay. i Note The application intelligently capitalizes common names with mixed casing like McHenry, and MacKenzie. Also note that the mixed case option does not apply to email addresses. - Upper Case: Converts the casing to upper case. For example, JOHN MCKAY. Default: "mixed"								

Name	ID	Type	Description
Diacritics	diacritics	String	Mandatory. Specifies whether to retain diacritical characters in the output: • Include: Retains the standardized diacritical characters. For example, Sánchez. • Remove: Replaces diacritical characters such as accent marks, umlauts, and so on, with the ASCII-equivalent letters. For example, Sanchez. Default: "include"
Data Type	dataType	String	Mandatory. Used to specify the type of parsing. • Firm: Contains only firm data • Person: Contains only person data • Person or Firm: Contains a mix of person and firm data Default: ""
Passthrough	passthrough	Boolean	Mandatory. Setting this option to True passes the input data unchanged to the output data. You cannot rename the input fields. If an input field has the same name as an output field defined in the operator, then the output field overwrites the input field's value. false
[Firm Person Person or Firm] Input		Array	Click + Add to define one or more input columns where you specify the column name and mapping for each column.
Column ID		String	Mandatory. This string uniquely identifies the column and should match the ID or name of the column coming into the operator.
Mapping		String	Mandatory. Specifies the mapping for this column.
[Firm Person Person or Firm] Input		Array	Mandatory. Click + Add to map from the internal processing columns to one or more output columns.
Column ID		String	Mandatory. Name to give to the output column.
Mapping		String	Mandatory. Specifies the mapping to the output column.

Related Information

[Person and Firm Cleanse Input Columns \[page 255\]](#)

[Person and Firm Cleanse Output Columns \[page 257\]](#)

2.5.6.1 Person and Firm Cleanse Input Columns

Map the input columns in the Person and Firm Cleanse node.

Person Cleansing Best Practice

When you set the *Data Type* option to *Person* and the name is in a single input column, map it to *Person*. Otherwise, map the name input columns to the discrete input columns such as *First Name*, *Middle Name*, and *Last Name*. An error occurs if you attempt to map the input data to *Person* and other discrete input fields at the same time.

Input Columns

The columns are listed alphabetically. Some columns are available only in one of the following data types: Firm, Person, or Person and Firm. The Data Type column shows the data types where the column is available.

Input Column	Description	Data Type
Country	Use these columns to control how the data is processed and output on a record-by-record basis. You must prepare the content yourself before inputting to the Cleanse operator. To use this feature, Country is required, and Language and Region are optional. <i>Country</i> : Prepare a column that contains the appropriate 2-character ISO country code. Country is the primary column that determines how the data is processed and output. <i>Language</i> : This column is optional and when mapped it is used only when the country is Belgium (BE) or Switzerland (CH). For Belgium records, include FR for records that use the French domain and include NL for records that use the Dutch domain. For Switzerland records, include DE for records that use the German domain, include FR for records that use the French domain, and include IT for records that use the Italian domain. If nothing is mapped to Language, then the French domain is used for all Belgium records and the German domain is used for all Switzerland records. <i>Region</i> : This column is optional and when mapped it is used only when the country is Canada (CA). For Canada records, include QC (Quebec) for records that use the French domain, and for records that use the English domain you can include a blank, null or any other 2-character province abbreviation. If nothing is mapped to Region, then the English (EN_US) domain is used for all Canada records.	Firm, Person, Person or Firm
Language		
Region		

Input Column	Description	Data Type
Firm	Map a discrete column that contains organization name information to this column. The contents of this column can include names of companies, organized groups, educational institutions, and so on. Operator can parse only a single firm at a time. Additional firms can be parsed by adding a second operator to the workflow.	Firm
First Name	Map a discrete column that contains first name information to this column. The contents of this column can contain a combination of first name, middle name, compound names, or prenames.	Person
Honorary Post-name	Map a discrete column that contains honorific name suffix information to this column. For example, Ph.D.	Person
Last Name	Map a discrete column that contains last name information to this column. The contents of this column can contain a single last name, compound last names, or name suffix information.	Person
Last Name 2	Map a discrete column that contains a second last name to this column. Map to this column only if the input data contains two last name columns. For example, map a paternal last name column to Last Name and map a maternal last name column to Last Name 2, or vice versa depending on cultural norms.	Person
Maturity Post-name	Map a discrete column that contains maturity name suffix information to this column. For example, Jr., Sr., or III.	Person
Middle Name	Map a discrete column that contains middle name information to this column. Map to this column only if the input data contains two given name columns. For example, map a first name column to First Name and map a middle name column to Middle Name. If the input data contains only one column that contains the combination of first name and middle name, map the column to First Name; do not map any column to Middle Name.	Person
Person	Use this input field when the person information is in a single field. For example Mary Angela Smith.	Person
Person or Firm	Map a discrete column to this column that contains a person name in some records and an organization name in other records. For example, if there is a customer name column in which some customers are individuals and other customers are organizations.	Person or Firm
Prenome	Map a discrete column that contains name prefix information to this column. For example, Mr., Mrs., Dr., or Lt. Col.	Person
Title	Map a discrete column that contains occupational title information to this column.	Person

2.5.6.2 Person and Firm Cleanse Output Columns

Map the output columns in the Person and Firm Cleanse node.

Person Cleansing Best Practices

When you set the *Data Type* option to *Person* and the *Cleanse* option to anything except *Global*, map the columns in a way that is appropriate for that region or country. When setting the *Cleanse* option to *Global*, follow these best practices:

- When outputting to a single person column, map to the *Person* column.
- When outputting to two columns (first and last names), map to *First Name and Middle Name* and *Last Name 1-2* columns.
- When outputting to three columns (first, middle, and last names), map to *First Name*, *Middle Name*, and *Last Name 1-2* columns.
- When outputting to four columns (first, middle, paternal last, and maternal last names), select *First Name*, *Middle Name*, *Last Name*, and *Last Name 2* columns.

Output Columns

The columns are listed alphabetically. Some columns are available only in one of the following data types: Firm, Person, or Person and Firm. The Data Type column shows the data types where the column is available.

Output Column	Description	Data Type
Extra	Data that is determined to be something other than organization or person name data.	Firm, Person, Person and Firm
Firm	The cleansed form of the organization name.	Firm, Person and Firm
First Name	First given names for most cleanse domains. For example, in "Mr. John Paul Anderson Schmidt Jr., PhD" the First Name is "John". For some domains such as French, the First Name contains the first two given names when the person has a compound name. For example, in "Mr. John Paul Anderson Schmidt Jr., PhD" the First Name is "John Paul".	Person, Person and Firm
First Name and Middle Name	Combination of the First Name column and the Middle Name column. For example, in "Mr. John Paul Anderson Schmidt Jr., PhD" the First Name and Middle Name is "John Paul".	Person, Person and Firm

Output Column	Description	Data Type
Gender	Gender of the person as classified in the reference data and is based on the probability of gender within locales represented by the cleanse domain. The gender is determined primarily by the name prefix or first name and, when ambiguous, then by the middle name. MALE_STRONG and FEMALE_STRONG are generated when the gender confidence is high. MALE_WEAK and FEMALE_WEAK are generated when the gender confidence is medium. AMBIGUOUS is generated when the gender confidence is low.	Person, Person and Firm
Honorary Postname	Name suffix that represents honorific or academic affiliation. For example, in "Mr. John Paul Anderson Schmidt Jr., PhD" the Honorary Postname is "PhD".	Person, Person and Firm
Information Code	Code generated only for records that have data in the firm or person columns that appears to be suspect.	Firm, Person, Person and Firm
Last Name	The full last name for most cleanse domains, even when the last name consists of multiple last names. For example, in "Mr. John Paul Anderson Schmidt Jr., PhD" the Last Name is "Anderson Schmidt". For some domains such as Spanish and Portuguese, the Last Name contains only the first of the last names when the person has a compound last name. For example, in "Mr. John Paul Anderson Schmidt Jr., PhD" the Last Name is "Anderson".	Person, Person and Firm
Last Name 1-2	Combination of the Last Name column and the Last Name 2 column. For example, in "Mr. John Paul Anderson Schmidt Jr., PhD" the Last Name 1-2 is "Anderson Schmidt".	Person, Person and Firm
Last Name 2	This column is empty for most cleanse domains because it's the Last Name column that contains the full last name when a last name consists of multiple names. For some domains such as Spanish and Portuguese, Last Name 2 contains the second of the last names when the person has a compound last name. For example, in "Mr. John Paul Anderson Schmidt Jr., PhD" the Last Name 2 is "Schmidt".	Person, Person and Firm
Maturity Postname	Name suffix that represents generational level. For example, in "Mr. John Paul Anderson Schmidt Jr., PhD" the Maturity Postname is "Jr.".	Person, Person and Firm
Middle Name	Second given name for most cleanse domains. For example, in "Mr. John Paul Anderson Schmidt Jr., PhD" the Middle Name is "Paul". For some domains such as French, when the person has a compound name the First Name contains the first two given names and the Middle Name is empty unless there is a third name. For example, in "Mr. John Paul Anderson Schmidt Jr., PhD" the First Name is "John Paul" and the Middle Name is empty.	Person, Person and Firm
Name Designator	Designator that can be input with the person name. For example, in "Attn: John Anderson" the Name Designator is "Attn:".	Person, Person and Firm
Person	Full person name with the name suffix, but without the name prefix, name designator, or occupational title. For example, in "Mr. John Paul Anderson Schmidt Jr., PhD" the Person is "John Paul Anderson Schmidt Jr., PhD".	Person, Person and Firm
Person 2	Name of the second person when data is input with two person names. For example, in "John and Mary Anderson" the Person is "John Anderson" and the Person 2 is "Mary Anderson", and in "Gurvinder Singh s/o Sh. Tejveer Singh" the Person is "Gurvinder Singh" and the Person 2 is "Tejveer Singh".	Person, Person and Firm

Output Column	Description	Data Type
Person or Firm	A person name in some records and an organization name in other records. This column contains data only when a column is input mapped to Person or Firm, and is intended to be used when a single column may contain either a person name or an organization name. For example, a customer name column in which the customer may be an individual or a corporation.	Person and Firm
Prename	Name prefix that can represent either a personal title such as "Mr." or "Ms." or a professional title such as "Dr." or "Prof.". For example, in "Mr. John Paul Anderson Schmidt Jr., Ph. D." the Prename is "Mr".	Person, Person and Firm
Title	The job or occupational title of a person. For example, Manager or Vice President of Marketing.	Person, Person and Firm

2.5.7 Validation Rule

Create rules and route records that pass through the Pass output port. Route failed records through the Fail output port.

You may want to create a validation rule to output a list of users who are the age of 18 or older, for example. You can create a rule that specifies the AGE column is $\geq$ (greater than or equal to) 18. Any values that are 18 or more are output through the Pass port, while those records that are less than 18 or have a null or invalid value are output through the Fail port.

You can view defined examples of this operator by clicking the Graphs tab and searching for "Basic Validation" and "IoT Validation".

Configuration Parameters

Parameter	Type	Description
Label	string	Required. Enter the name of the validation rule operator.

Parameter	Type	Description
Rules	array	Required: Enter one or more rules. Click the Open Editor icon, and then click +Add item and complete the following options. • Column (type string): Required. Enter the column name. For example, AGE. • Condition (type string): Required. Enter the condition. For example, >=. The following conditions are available: ○ IS NULL ○ IS NOT NULL ○ < ○ <= ○ = ○ <> In the expression Title <> 'CEO', the values CEO and null fail the rule. ○ = ○ LIKE When using LIKE, remember these items: ○ STRING matches any STRING matching expression, which must be in single quotes. For example 'Be%' matches Ben, Bench, Begin. ○ INTEGER matches any INTEGER specified without single quotes, or a single quoted expression such as '15%', which matches values of 15, 150, 151, and so on. ○ NUMBER matches any INTEGER or NUMBER. ○ Use these wildcards with LIKE; ○ %: A string of zero or more characters. ○ _ (underscore): A single character. ○ []: A single character with a specific range or set. ○ [^]: A single character not within the specified range or set. ○ MATCH PATTERN: Enter a pattern based on the match_pattern function. ○ BETWEEN: Specify a range of values. When using BEWEEN, remember these items: ○ The values set are inclusive. For example, [20, 22] includes 4 values: 20, 21, and 22. ○ Enclose string values in single quotes. ○ STRING ['A', 'D'] matches any STRING alphabetically, inclusively between A and D. ○ INTEGER [20, 24] matches any INTEGER, NUMBER, or STRING, inclusively between 20 and 24.

Parameter	Type	Description
		○ NUMBER [20.15, 400.78] matches any INTEGER or NUMBER, inclusively between 20.15 and 400.78. ○ IN SET ○ CUSTOM: Select to call a more complex function that is defined in the Validation Functions option. ● Value (type string): Required. See Value list. ● Value List (type string): Required. Depending on the condition you selected, you may need to enter a value or a values list. For example, enter a value of 18. A values list is available when you select BETWEEN or IN SET. Click +Add item to set the minimum value, for example, 9. Click +Add item again to set the maximum value, for example, 17. If the substitution value is a hardcoded string you must enclose the string in single quotes. If single quotes are present within the string then those must be escaped with a backslash () character. ● Fail Action (type string): Enter the Fail Action for those records that do not pass the rule. ○ Fail: Sends the record to the Fail port. ○ Pass: Sends the record to the Pass port. ○ Both: Sends the record to the Pass and Fail ports.
Substitutions	array	If you choose Pass or Both for a rule's Fail Action, you can enter a substitution value for the failed values that go to the Pass port. For example, if you have a rule that the column AGE Is Not Null, you can create a substitution variable that assigns the AGE column with a value of 999. Therefore, the failed records are output to both the Fail and Pass ports. The failed records included in the Pass port have an age of 999, whereas the same records in the failed port continue to have a null value. If the substitution value is a hardcoded string, you must enclose the string in single quotes. If single quotes are present within the string, then those quotes must be escaped with a backslash (\\) character.

Parameter	Type	Description
Validation Functions	array	If you want to create a more complex function, click the Open Editor icon, and then click +Add item and complete the following options: 1. Enter the name of the function. For example, EighteenAndOlder. 2. Enter the function in the Body option. For example, <code>if(\$arg >= 18) return 1; else return 0;</code>. This function reads: if the argument passed is greater than or equal to 18, return a 1 (true). If the value is less than 18, then return 0 (false). 3. Click +Add item to create the arguments. 4. Enter the name of the argument. For example, \$arg. 5. Define the data type of the argument: String, Number, or Integer. 6. If you chose a String data type, then enter the Length. For example, 256. 7. Click Save. 8. In the Rules option, set the Condition to CUSTOM. In the Value option, call the name of the validation function. For example, EighteenAndOlder (AGE). Note that the 'AGE' in parentheses is the name of the input column name from your defined schema. Make sure that the input column's data type in the schema matches the data type defined in this argument. 9. Click Save.
Input Schema	array	Required. Define the column names and data types for the input source used in the Validation Rule operator. Click the Open Editor icon, and then click +Add item for each input column. If you have FIRST_NAME, LAST_NAME, and AGE, you would create three values. 1. Enter the name of the first column. 2. Select the data type. 3. If you chose a String data type, then enter the Length. For example, 256. 4. Repeat for the additional columns, and then click Save.
Format	string	Required. Choose the format of the input.

Parameter	Type	Description
CSV Properties	array	Optional. Click the  pencil icon, and then set these options. - Column Delimiter (type string): The field separator. Only the following characters are supported: , ; : and TAB. Default: , - Text Delimiter Style (type string): Indicates how the text delimiter should be applied. If Always is selected, all columns will have a text delimiter; if Minimal, the text delimiter will be added only when required. The text delimiter style is used to correctly format the output data. - Header Included (type boolean): Indicates that the output will have header information. The input schema is used to set the first row of the output as header. If the input file contains header information, preprocess the data by removing the header and sending it to the Validation Rule.
Batch Size	integer	Required. Enter the number of rows you want included in each batch that is output from this operator. If you have 1000 rows of data and set this option to 250, then four batches are output. If you want all the records output in a single batch, set the size larger than the greatest input size. In the previous example, you would set this option to 1001 or higher.

Input

Input	Type	Description
in	string	The input port to the Validation Rule operator expects the data in the format specified by the Format option.

Output

Output	Type	Description
pass	string	Contains the list of passed records. It shows the contents of each column. If a substitution variable was used on the failed records, then that value is included in the column that was defined.
fail	string	Contains the list of records that failed the rule or rules. Each record has the columns, the Failed Action indicator (B=Both, F=Fail), and which rules the record failed.
failInformation	string	Contains information about why the record failed. It contains a unique ID for each record, the name of the rule that failed, and the column name.

Output	Type	Description
outError	string	When mapped, the operator error is sent to next operator in line, and the graph is run. When not mapped, the graph terminates with the operator error.

2.5.8 Data Quality SQL Reference

Learn about the built-in SQL functions available for use in the Data Quality operators.

This table lists the names, categories, and descriptions of the Data Quality functions.

Function	Category	Description
abs [page 265]	Math	Returns the absolute value of an input number.
ceil [page 266]	Math	Returns the smallest integer value greater than or equal to an input number.
double_metaphone [page 267]	String	Encodes the input string using the Double Metaphone algorithm and returns a string.
floor [page 268]	Math	Returns the largest integer value less than or equal to an input number.
index [page 269]	String	Returns the index of a given word in a string.
length [page 270]	String	Returns the number of characters in a given string.
lower [page 271]	String	Changes the characters in a string to lowercase.
ltrim [page 272]	String	Removes specified characters from the start of a string.
ltrim_blanks_ext [page 273]	String	Removes blank and control characters from the start of a string.
rtrim [page 274]	String	Removes given characters from the end of a string.
rtrim_blanks [page 275]	String	Removes blank characters from the end of a string.
substr [page 276]	String	Returns a specific portion of a string starting at a given point in the string.
trunc [page 278]	Math	Truncates a given number to the specified precision.
upper [page 279]	String	Changes the characters in a string to uppercase.

Related Information

[abs \[page 265\]](#)
[ceil \[page 266\]](#)
[double_metaphone \[page 267\]](#)
[floor \[page 268\]](#)
[index \[page 269\]](#)
[length \[page 270\]](#)

[lower](#) [page 271]
[ltrim](#) [page 272]
[ltrim_blanks_ext](#) [page 273]
[rtrim](#) [page 274]
[rtrim_blanks](#) [page 275]
[substr](#) [page 276]
[trunc](#) [page 278]
[upper](#) [page 279]

2.5.8.1 abs

Use the abs function to return the absolute value of a number. The absolute value (sometimes known as the modulus) of a number is the value of a number without regard to its sign – it can also be thought of as the distance of a number from zero.

This topic applies to SAP Data Intelligence.

Syntax

```
abs (<num>)
```

Return value

decimal, double, int, or real

The absolute value of the given number, <num>. The type of the return value is the same as the type of the original number.

Where

<num>

The source number.

Example

Function	Results

abs (12.12345)	12.12345

Function	Results
<code>abs (-12.12345)</code>	12.12345

2.5.8.2 ceil

Use the ceil function to return the smallest integer value greater than or equal to a number.

This topic applies to SAP Data Intelligence.

Syntax

```
ceil (<num>)
```

Return value

decimal, double, int, or real

The indicated integer, cast as the same type as the original number, `<num>`.

Where

`<num>`

The source number.

Example

Function	Results
<code>ceil (12.12345)</code>	13.00000
<code>ceil (12)</code>	12
<code>ceil (-12.223)</code>	-12.000

2.5.8.3 double_metaphone

Use the double_metaphone function to encode the input string using the Double Metaphone algorithm.

This topic applies to SAP Data Intelligence.

Syntax

```
double_metaphone(<input_str, alternate, return_if_empty>)
```

Return Value

varchar

Returns the string containing the double metaphone encoding of the input string. The length of the return string depends on the length of the input string, but it is always shorter than the input string.

Where

<code><input_str></code>	The source string to encode.
<code><alternate></code>	A flag to control how the function returns the encoded strings. • When 0, returns the primary encoding. When no primary encoding, returns NULL or the input string based on how you set <code><return_if_empty></code>. • When input is not 0, returns the alternate encoding. If no alternate encoding, returns NULL or the input string based on how you set <code><return_if_empty></code>. • When NULL or invalid, such as non numeric, defaults to 0.
<code><return_if_empty></code>	A flag to determine whether to return null or the input string when there is no encoding. When 0, return NULL. Otherwise, return the input string when there is no encoding. When the parameter is NULL or invalid, such as non numeric, defaults to 1. When input is empty, there is no primary or secondary encodings. When <code><return_if_empty> = 0</code>, then returns NULL. When <code><return_if_empty> = 1</code>, then returns the empty string.

Details

Only use this function for input strings in English. The function ignores non-English characters.

When input is NULL, the function returns NULL.

When the second or third parameter has an invalid value, defaults to 0 and 1, respectively.

❖ Example

Function	Result
<code>Print(double_metaphone('Hello',0,0);</code>	Prints the double metaphone of the word "Hello."
<code>double_metaphone(\$VAR, 1,1);</code>	If the string stored in \$VAR does not have encoding available, then returns the original string.
<code>double_metaphone(\$VAR,'a','b');</code>	Returns the primary double metaphone encoding or the variable \$VAR when the primary encoding does not exist.

2.5.8.4 floor

Use the floor function to return the largest integer value equal to or less than a number.

This topic applies to SAP Data Intelligence.

⌘ Syntax

```
floor(<num>)
```

Return value

decimal, double, int, or real

The indicated integer, cast as the same type as the original number, `<num>`.

Where

`<num>`

The source number.

Details

❖ Example

Function	Results
<code>floor(12.12345)</code>	12.00000
<code>floor(12)</code>	12
<code>floor(-12.223)</code>	-13.000

2.5.8.5 index

Use the index function to return the index of a give character sequence in a string.

This topic applies to SAP Data Intelligence.

≡ Syntax

```
index(<input_string>, <index_string>, <start>)
```

Return value

int

Specifies the first location of the indicated character sequence.

Where

<code><input_string></code>	The source string.
<code><index_string></code>	The character sequence sought in <code><input_string></code> .
<code><start></code>	The position where the function starts searching in <code><input_string></code> for the character sequence contained in <code><index_string></code> . <code><Start></code> should be a positive number between 1 and the length of <code><input_string></code> .

Details

The function searches for the `<index_string>` beginning at the `<start>` position in the `input_string`.

- If `<start>` is 0, the function resets it to 1.
- If `<start>` is greater than the number of characters in `<input_string>`, the function returns NULL.
- If `<index_string>` is not found in `<input_string>`, the function returns NULL.

Ensure that the characters in `<index_string>` exactly match the sequence of characters in `<input_string>`.

❖ Example

The search is case-sensitive.

Function	Results
<code>index('Accounting Department', 'DEPARTMENT', 1)</code>	0 The <code>index_string</code> does not match the case of the <code>input_string</code> so the function returns 0.
<code>index('Accounting Department', 'Department', 1)</code>	12 The string "Department" starts at position 12 when you begin counting at position 1, which is the "A" of Accounting.

2.5.8.6 length

Use the `length` function to return the number of characters in a given string.

This topic applies to SAP Data Intelligence.

≡ Syntax

```
length(<value>)
```

Return value

integer

The number of characters in `<value>`.

Where

`<value>`

A string indicating the column name, variable, or other element whose length is calculated.

Details

❖ Example

In the [Mapping](#) box of a query, use the length function to return the number of characters in each row of a column. With the OUTPUT field selected in the target schema of a query, enter the following statement in the [Mapping](#) box:

```
length(dal_emp.ename)
```

The software produces the following results:

Source column (dal_emp.ename)	Target column (output)
jones	5
nguyen	6
tanaka	6

2.5.8.7 lower

Use the lower function to change the characters in a string to lower case.

This topic applies to SAP Data Intelligence.

≡ Syntax

```
lower(<value>, '<locale>')
```

Return value

varchar

The lowercase string. The return type is the same as `<value>`. The function leaves any characters that are not letters unchanged.

Where

<code><value></code>	The string to be modified.
<code><locale></code>	Optional. Specify a locale to convert the string to the locale.

Note
The function supports the ISO 639 language code and the ISO 3166 country code formats.

Details

❖ Example

Function	Results
<code>lower('Accounting101')</code>	'accounting101'
<code>upper((LastName,1,1)) lower(substr(LastName,2,LENGTH(LastName)))</code>	The value in column <code>LastName</code> with the first letter uppercase and the rest of the value lowercase. Note that this example does not account for two-word last names.
<code>lower(LastName,'tr')</code>	The value in column <code>LastName</code> is converted to all lowercase. It is also converted to the Turkish locale, using the ISO 639 language code.

2.5.8.8 ltrim

Use the `ltrim` function to remove specified characters from the start of the string.

This topic applies to SAP Data Intelligence.

≡ Syntax

```
ltrim(<input_string>, <trim_string>)
```


Return value

varchar

The modified string. The return type is the same as `<input_string>`.

Where

<code><input_string></code>	The string to be modified.
<code><trim_string></code>	The characters to remove from <code><input_string></code> .

Details

The `ltrim` function is case-sensitive.

The function scans `<input_string>` left-to-right removing all characters that appear in `<trim_string>` until it reaches a character not in `<trim_string>`.

❖ Example

Function	Results
<code>ltrim('Marilyn', ' ')</code>	'Marilyn'
<code>ltrim('ABCABCD', 'ABC')</code>	'D'
<code>ltrim('ABCABCD', 'EFG')</code>	'ABCABCD'
<code>ltrim('ABCDEABCDE', 'ABC')</code>	'DEABCDE'

❖ Example

To remove all leading blanks in a string, use `ltrim` as follows:

```
ltrim(EMPLOYEE.NAME, ' ')
```

where `EMPLOYEE.NAME` specifies the `NAME` column in the `EMPLOYEE` table. You may also use the `ltrim_blanks` or `ltrim_blanks_ext` functions for this.

2.5.8.9 ltrim_blanks_ext

Use the `ltrim_blanks_ext` function to remove blank and control characters from the start of a string.

This topic applies to SAP Data Intelligence.

Syntax

```
ltrim_blanks_ext(<input_string>)
```

Return value

varchar

The modified string. The return type is the same as `<input_string>`.

Where

`<input_string>`

The string to be modified.

Details

Example

Function	Results
<code>ltrim_blanks_ext(' Marilyn')</code>	'Marilyn'
<code>ltrim_blanks_ext(last_name)</code>	The value contained in the column last_name, with all leading blanks and control characters removed.

2.5.8.10 rtrim

Use the rtrim function to remove specified characters from the end of a string.

This topic applies to SAP Data Intelligence.

Syntax

```
rtrim('<input_string>', '<trim_string>')
```

Return value

varchar

The modified string. The return type is the same as `<input_string>`.

Where

<code><input_string></code>	The string to be modified.
<code><trim_string></code>	The characters to remove from <code><input_string></code> .

Details

The function scans `<input_string>` from right to left removing all characters that appear in `<trim_string>` until it reaches a character not in `<trim_string>`.

Removes trailing blanks only if `<trim_string>` contains trailing blanks. If the length of the modified string becomes zero after trimming, the function returns "" (empty string).

To remove all trailing blanks in a string, use the `rtrim_blanks` function.

❖ Example

Function	Results
<code>rtrim('Marilyn ', ' ')</code>	'Marilyn'
<code>rtrim('ZABCABC', 'ABC')</code>	'Z'
<code>rtrim('ZABCABC', 'EFG')</code>	'ZABCABC'

You may also use the `rtrim_blanks` or `rtrim_blanks_ext` functions for this.

2.5.8.11 rtrim_blanks

Use the `rtrim_blanks` function to remove blank characters from the end of a string.

This topic applies to SAP Data Intelligence.

≡ Syntax

```
rtrim_blanks(<input_string>)
```

Return value

varchar

The modified string. The return type is the same as `<input_string>`.

Where

`<input_string>`

The string to be modified.

Details

If the length of the modified string becomes zero after trimming, the function returns "" (empty string).

❖ Example

Function	Results
<code>rtrim_blanks('Marilyn ')</code>	'Marilyn'
<code>rtrim_blanks(last_name)</code>	The value contained in the column <code>last_name</code> with trailing blanks removed.

2.5.8.12 substr

Use the `substr` function to return a specific portion of a string starting at a given point in the string.

≡ Syntax

```
substr(<input_string>, <start>, <length>)
```

Return value

varchar

The modified string. The return data type is the `<input_string>`. If the length is a constant, then it is a varchar of the given length.

Where

<code><input_string></code>	The string to be modified.
<code><start></code>	The position in the <code><input_string></code> where the function obtains the first character of the new string. The function counts characters from the beginning of <code><input_string></code>. • In normal data flows, the first character is position number 1. • If <code><start></code> is 0, the new string begins with the first character (position 1). • If <code><start></code> is negative, the function counts characters from the end of <code><input_string></code>. The new string begins with the character in that position from the end of the string. The function returns NULL or an empty string under the following circumstances: • If <code><start></code> is greater than the number of characters in <code><input_string></code>, the function returns NULL. • If <code><length></code> is less than 1, the function returns an empty string.
<code><length></code>	The number of characters in the resulting string. • If <code><length></code> is 0 or negative, the function returns an empty string. • If <code><length></code> is greater than the number of characters remaining in <code><input_string></code> after <code><start></code>, the function returns only the remaining characters. The function keeps the trailing blanks in the remaining <code><input_string></code> after <code><start></code>.

For information about how Data Services uses the substr function with HANA, see SAP Note [2808903](#).

Details

❖ Example

Function	Results
<code>substr('94025-3373', 1, 5)</code>	'94025'
<code>substr('94025-3373', 7, 4)</code>	'3373'
<code>substr('94025', 7, 4)</code>	NULL
<code>substr('Dr. Schultz', 4, 18)</code>	'Schultz'
<code>substr('San Francisco, CA', -4, 18)</code>	', CA'

2.5.8.13 trunc

Use the trunc function to truncate a given number to the specified precision without rounding the value.

This topic applies to SAP Data Intelligence.

Syntax

```
trunc(<num1>, <precision>)
```

Return value

decimal, double, int, or real

The truncated number. The return type is the same as the original number, `<num1>`.

Where

<code><num1></code>	The source number.
<code><precision></code>	An integer indicating the number of decimals in the result. If <code><precision></code> is negative, digits to the left of the decimal point are truncated and the value is padded with zeros.

Details

Example

Function	Results
<code>trunc(120.12345, 2)</code>	120.12
<code>trunc(120.12999, 2)</code>	120.12
<code>trunc(180, -2)</code>	100
<code>trunc(120.123, 5)</code>	120.12300

2.5.8.14 upper

Use the upper function to change the characters in a string to uppercase.

This topic applies to SAP Data Intelligence.

Syntax

```
upper (<value>, '<locale>')
```

Return value

varchar

The uppercase string. The return type is the same as `<value>`. The software does not change the characters that are not letters.

Where

<code><value></code>	The string to be modified.
<code><locale></code>	Optional. A locale to which the function converts the string.

i Note
The software supports ISO 639 language code and ISO 3166 country code formats.

Details

Example

Function	Results
<code>upper ('Accounting101')</code>	<code>'ACCOUNTING101'</code>
<code>upper (substr (LastName, 1, 1)) lower (substr (LastName, 2, LENGTH (LastName)))</code>	The value in column <code>LastName</code> with the first letter uppercase and the rest of the value lowercase. Note that this example does not account for last names with two words.

Function	Results
<code>upper (LastName, 'tr')</code>	The value in column <code>LastName</code> is converted to all upper-case. It is also converted to the Turkish locale, using the ISO 639 language code.

2.6 Data Workflows

The graphs modeled with the data work flow operators are referred to as Data Workflows.

Modeler categorizes certain operators as Data Workflow operators, which when executed runs for a limited period of time and finishes with the status either as *completed* or *dead*. This section describes the supported data workflow operators in SAP Data Intelligence.

Related Information

[BW Process Chain \[page 280\]](#)

[Data Transfer \[page 282\]](#)

[HANA Flowgraph \[page 284\]](#)

[Notification \[page 285\]](#)

[Pipeline \[page 287\]](#)

[SAP Data Services Job \[page 289\]](#)

[Workflow Merge \(and\) \[page 291\]](#)

[Workflow Merge \(or\) \[page 292\]](#)

[Workflow Split \[page 293\]](#)

[Workflow Terminator \[page 294\]](#)

[Workflow Trigger \[page 295\]](#)

2.6.1 BW Process Chain

The BW Process Chain operator helps execute an SAP BW process chain in an SAP BW system.

SAP BW process chain is a sequence of processes that are scheduled to wait in the background for an event. Some of these processes can trigger a separate event that can, in turn, start other processes.

This operator has one input port (`input`) and two output ports (`output` and `error`). Use this operator in a graph that uses only other data workflow operators.

Connect the input port to another data workflow operator to trigger its start. On the `error` output port, the engine writes a message in case of errors when trying to execute the process chain. On success, the engine

writes a message on the `output` port. If you want the execution of the graph to proceed, connect the respective output ports to other data workflow operators.

i Note

Any message written to an unconnected output port results in the graph failure.

Configuration Parameters (using Connection ID)

Parameter	Type	Description
SAP BW Connection	object	Connection ID that references a connection to a remote SAP Data Intelligence system.
SAP BW ProcessChain ID	string	Name of the process chain to execute
Retry Interval	string	Time interval in seconds (default 20) to wait until next status update.
Retry Attempts	string	Maximum number of attempts (default 1080) to query for status update.

Configuration Parameters (manual)

If you are manually entering the connection details of the SAP BW system in which you want to execute the process chain:

Parameter	Type	Description
Host	string	Host name of the SAP BW system. For example, <code>host.name</code> .
Port	number	Port number. For example, <code>8090</code> .
Protocol	string	"HTTPS" or "HTTP".
User	string	Name of the user for the authentication.
Password	string	Password for authentication.

Input

Input	Type	Description
<code>input</code>	string	Accepts message from a connected data workflow operator.

Output

Output	Type	Description
<code>output</code>	string	If the operator executed successfully, this port carries the success message.
<code>error</code>	string	Operator error. If the port is connected, then the graph will keep running. If it is not connected, the graph will terminate with the operator error.

For more information on how to use this operator, see [Execute an SAP BW Process Chain Operator](#).

2.6.2 Data Transfer

Data Transfer operator helps transfer data from an SAP BW system to an SAP Vora system or other major cloud storages such as S3, ADL, Alibaba Cloud OSS, GCS, WASB, or HDFS.

This operator enables you to select the source data from an SAP BW system and the target storage to which you want to transfer the data. It has one input port (`input`) and two output ports (`output` and `error`). Use this operator in a graph that uses only other data workflow operators.

Connect the input port to another data workflow operator to trigger its start. On the `error` output port, the engine writes a message in case of errors when trying to execute the data transfer. On success, the engine writes a message on the `output` port. If you want the execution of the graph to proceed, connect the respective output ports to other data workflow operators.

i Note

Any message written to an unconnected output port results in a graph failure.

Configuration Parameters

Parameter	Type	Description
source	custom	• Source SAP BW system and a data source (BW query or BW InfoProvider) within the system. • Variable values, if any. • Measures and dimensions to project to the target SAP Vora table,
target	custom	Target SAP Vora system and a table within the system or other cloud storages.

Input

Input	Type	Description
input	string	Accepts message from a connected data workflow operator.

Output

Output	Type	Description
output	string	If the operator executed successfully, this port carries the success message.
error	string	Operator error. If the port is connected, then the graph will keep running. If it is not connected, the graph will terminate with the operator error.

For more information on how to use the Data Transfer operator, see [Transfer Data from SAP BW to SAP Vora or Cloud Storage](#).

2.6.3 HANA Flowgraph

The HANA flowgraph operator helps execute an SAP HANA flowgraph in an SAP HANA system.

Executing an HANA flowgraph operator within a graph helps you to transform data from a remote source into SAP HANA either in batch or real-time mode. The flowgraphs are predefined in the server and represents a complete data flow. User must select the required SAP HANA flowgraph, and all the operations that the flowgraph must perform is defined in the server.

This operator has one input port (`input`) and two output ports (`output` and `error`). Use this operator in a graph that uses only other data workflow operators.

Connect the input port to another data workflow operator to trigger its start. On the `error` output port, the engine writes a message in case of errors when trying to execute the flowgraph. On success, the engine writes a message on the `output` port. If you want the execution of the graph to proceed, connect the respective output ports to other data workflow operators.

Note

Any message written to an unconnected output port results in the graph failure.

Configuration Parameters

Parameter	Type	Description
SAP HANA Connection	object	Connection ID that references a connection to an SAP HANA system.
SAP HANA Flowgraph Name	string	Fully qualified SAP HANA Flowgraph name including schema name. For example, <code>/FLOWGRAPH_SCHEMA/PACKAGE::NAME</code>
Variables	array	Name-value pairs for substitution parameters. The default value is no variables.
Table Variables	array	Fully quoted SAP HANA table or view names. The default value is no variables.
Retry Interval	string	Time interval in seconds (default 20) to wait until next status update.
Retry Attempts	string	Maximum number of attempts (default 10) to query for status update.

Input

Input	Type	Description
input	string	Accepts message from a connected data workflow operator.

Output

Output	Type	Description
output	string	If the operator executed successfully, this port carries the success message.
error	string	Operator error. If the port is connected, then the graph will keep running. If it is not connected, the graph will terminate with the operator error.

For more information on how to use this operator, see [Execute a HANA Flowgraph Operator](#).

2.6.4 Notification

The Notification operator helps send email notifications at certain points in the data workflow execution.

The message body contains technical information that the Notification operator receives at its input port from the previous operator in the data workflow.

Configuration Parameters

Parameter	Type	Description
Connection	object	Connection ID that references a connection to an SMTP server.
Delete attachments after sending	boolean	A flag that controls whether the files attached to the email should be deleted after sending.
Default From Value	string	The value of <code>email.from</code> , which will be used when no value is received through the incoming message.

Parameter	Type	Description
Default To Value	array	The value of <code>email.to</code> , which will be used when no value is received through the incoming message.
Default Subject Value	string	The value of <code>email.subject</code> , which will be used when no value is received through the incoming message.

Input

Input	Type	Description
<code>in</code>	any.*	Accepts a message whose attributes will be used to fill the email information and whose body will be used as the email body. If there are multiple entries for a same header, they can be separated by comma. Message attributes: • <code>email.to</code> (type string) mandatory: The recipient of the email. • <code>email.from</code> (type string) mandatory: The sender of the email. • <code>email.subject</code> (type string) mandatory: The subject of the email. • <code>email.cc</code> (type string): The CC's of the email. • <code>email.attachments</code> (type string): The attachments of the email.

Output

Output	Type	Description
<code>out</code>	string	String containing a list of all the recipients that received the email, separated by semicolon (;).

2.6.5 Pipeline

Pipeline operator helps execute an SAP Data Intelligence pipeline in an SAP Data Intelligence system.

The data pipeline, when executed, can help process the raw data from multiple sources and make it available for different use cases.

This operator has one input port, Input, and two output ports, Output and Error. Use this operator in a graph that uses only other data workflow operators.

Connect the input port to another data workflow operator to trigger its start. On the Error output port, the engine writes a message in case of errors when trying to execute the Pipeline. On success, the engine writes a message on the Output port. If you want the execution of the graph to proceed, connect the respective output ports to other data workflow operators.

i Note

Any message written to an unconnected output port results in a graph failure.

Configuration Parameters (using Connection ID)

Parameter	Type	Description
VFlow Connection	object	Connection ID that references a connection to a remote SAP Data Intelligence system.
Graph Name	string	Name of the graph to execute.

i Note
The value help that the tool provides lists only the graphs from the local instance (that is, the currently used modeler instance) even if you have selected a remote instance.

Parameter	Type	Description
Running Permanently	string	If set to true, the operator execution checks whether the data pipeline state is in a running state. Then it starts the data pipeline execution if it is not in a running state, and terminates immediately (status: completed) while the data pipeline remains in the running state. But, if the task was already running with a different task version, then that instance is stopped, and the new task version is started and immediately terminated (status: completed). If set to false, the operator executes the data pipeline once, and the operator execution terminates after the pipeline execution terminates. → Remember If you do not provide any value, the modeler uses false as the default value.
Retry Interval	string	Time interval in seconds (default 20) to wait until next status update.
Retry Attempts	string	Maximum number of attempts (default 10) to query for a status update.

Configuration Parameters (manual)

If you are manually entering the connection details of the SAP Data Intelligence system in which you want to execute the pipeline:

Parameter	Type	Description
Host	string	Host name of the Modeler system. For example, localhost .
Port	number	Port number. For example, 8090 .
Protocol	string	"HTTPS" or "HTTP"

Parameter	Type	Description
User	string	Name of the user used for the authentication. Use this syntax, tenant \username . tenant is the tenant to use and username the login user name.
Password	string	The password of the user used for the authentication.

Input

Input	Type	Description
input	string	Accepts a message from a connected data workflow operator.

Output

Output	Type	Description
output	string	If the operator executed successfully, this port transports the success message.
error	string	Operator error. If the port is connected then the graph will keep running. If it is not connected, the graph will terminate with the operator error.

For more information, see [Execute an SAP Data Intelligence Pipeline](#).

2.6.6 SAP Data Services Job

The SAP Data Services Job operator helps execute an SAP Data Services job in a remote system.

Executing an SAP Data Services job helps users to integrate, transform, and improve the data quality.

This operator has one input port, Input, and two output ports, Output and Error). Use this operator in a graph that uses only other data workflow operators.

Connect the input port to another data workflow operator to trigger its start. On the Error output port, the engine writes a message in case of errors when trying to execute the job. On success, the engine writes a message on the Output port. If you want the execution of the graph to proceed, connect the respective output ports to other data workflow operators.

i Note

Any message written to an unconnected output port results in a graph failure.

Configuration Parameters

Parameter	Type	Description
SAP Data Services Connection	object	Connection ID that references a connection to a remote SAP Data Services system.
Job	string	Name of the SAP Data Services job to execute.
Repository	string	Name of the repository that contains the job.
Job Server	string	Name of the job server to use to execute the job.
System Configuration	string	Name of the system configuration to use for running the job.
Global Variables	string	Name-value pairs for global variables. The default value is no variables.
Substitution Parameters	string	Name-value pairs for substitution parameters. The default value is no parameters.

Input

Input	Type	Description
in	string	Trigger job execution.

Output

Output	Type	Description
outSuccess	string	Success output of remote task.
outError	string	If mapped, the operator errors are sent to this port and the graph continues running. If not mapped, the graph terminates with the operator error.

For more information, see [Execute an SAP Data Intelligence Pipeline](#).

2.6.7 Workflow Merge (and)

The Workflow Merge (and) operator helps control the flow of execution in a data workflow.

This operator sends a message to the output port when it receives a message on both input ports. The process shuts down after sending the message. This operator is intended to control the flow of execution when there are parallel executions within the data workflow.

Configuration Parameters

None

Input

Input	Type	Description
input	string	Accepts message from another data workflow operator.
input2	string	Accepts message from another data workflow operator.

i Note

Connect both input ports to other data workflow operators. Leaving any of the input ports unconnected results in the data workflow execution to stop.

Output

Output	Type	Description
output	string	Message from the operator when both its input ports receive a message.

For an example on how to use the Workflow Merge (and) operator, see [Control Flow of Execution](#).

2.6.8 Workflow Merge (or)

The Workflow Merge (or) operator helps control the flow of execution in a data workflow.

This operator sends a message to the output port once it receives a message from any of the input ports. All further inputs are ignored.

Configuration Parameters

None

Input

Input	Type	Description
input	string	Accepts message from another data workflow operator.
input2	string	Accepts message from another data workflow operator.

i Note

If the input ports are connected to other data workflow operators and if the operator receives a message at any of its input ports, then leaving the output port unconnected results in a data workflow execution failure.

Output

Output	Type	Description
output	string	Message from the operator once it receives a message from any of its input ports.

For an example on how to use the Workflow Merge (or) operator, see [Control Flow of Execution](#).

2.6.9 Workflow Split

The Workflow Split operator helps duplicate an input message into two output messages.

The Workflow Split operator is intended to control the flow of execution when there are parallel executions within the graph. The operator has one input port and two output ports. The operator sends the message at the input port to both the output ports.

Configuration Parameters

None

Input

Input	Type	Description
in	any.*	Accepts a message from the connected data workflow operator.

Output

Output	Type	Description
out1	any.*	Sends the incoming message to another connected data workflow operator.

Output	Type	Description
out2	any.*	Sends the incoming message to another connected data workflow operator.

i Note

Leaving the output port unconnected results in a graph failure when the operator receives an input message.

For an example on how to use the Workflow Split (and) operator, see [Control Flow of Execution](#).

2.6.10 Workflow Terminator

The Workflow Terminator operator helps shut down the data workflow execution.

The operator has one input port. When started and when the operator receives a message, it shuts down the data workflow. The data workflow terminates with the state, *Completed*.

Use this operator with other data workflow operators only.

Configuration Parameters

None

Input

Output	Type	Description
output	string	Receives a message to shut down the execution of the data workflow.

Output

None

2.6.11 Workflow Trigger

The Workflow Trigger operator sends a start message, which you can use to start a data workflow.

This operator has one output port. When started, the operator sends a message on its output port. If the output port is connected, the next data workflow operator starts its execution. But, if the output port is unconnected, the graph execution fails.

Use this operator with other data workflow operators only.

Configuration Parameters

None

Input

None

Output

Output	Type	Description
output	string	Sends a message to begin execution of the next connected data workflow operator.

2.7 Files

Related Information

[From File \[page 296\]](#)

[List Files \[page 296\]](#)

[Monitor Files \[page 300\]](#)

[Read File \[page 303\]](#)

[Remove File \[page 306\]](#)

[To File \[page 308\]](#)

[Write File \[page 309\]](#)

2.7.1 From File

This operator extracts the path attribute from a file message.

Configuration Parameters

None

Input

Input	Type	Description
file	message.file	A File message where the path is extracted from.

Output

Output	Type	Description
path	message	The path found in the input file attributes.

2.7.2 List Files

This operator lists files and directories from various storage services, which are identified by a File Reference, for example, a pair of connection and path (qualified name) values.

Supported services are:

- [Alibaba Cloud Object Storage Service \(Alibaba Cloud OSS\)](#)
- [Amazon Simple Storage Service \(S3\)](#) (also supporting S3-compatible services)
- [Google Cloud Storage \(GCS\)](#)

- [Hadoop Distributed File System \(HDFS\)](#)
- [WebHDFS](#) (only supported via Connection Management)
- [Microsoft Azure Data Lake \(ADL\)](#)
- Microsoft Azure Data Lake Gen. 2 (ADL_V2)
- [Microsoft Azure Storage Blob \(WASB\)](#)
- [SDL](#)
- SSH File Transfer Protocol (SFTP)
- System Management Files (/files) (only supported via Manual connection. Refer to the file system presented on the System Management app, only for files found in /files/)

i Note

System Management Files (/files) does not use a Connection ID. It is the file system presented in the System Management app, only for files found in /files/. The usage of the /files/ root path (or the /vrep folder) is deprecated.

A `List` operation is triggered according to its configuration. Check the parameter `List` for further information.

i Note

Use `Connection Management` for defining connections over the manual option.

Configuration Parameters

Name	ID	Type	Description
List	mode	string	Mandatory. The operation mode. Details of each mode are detailed below. Accepted values: • On input: A file is read for every reference input received. If the reference is a directory, nothing is done and no output is produced. • Once: The file pointed by the operator configuration parameters is read once as soon as the graph is running. Default: "On input"

Name	ID	Type	Description
Connection (static)	connection	object	Points to the static connection to list from. Only available if <i>List</i> is set to Once. Default: {}
Connection (dynamic)	dynamicConnection	object	Points to the dynamic connection to list from. The connection configuration is detailed here. It uses the file connection attribute. Only available if <i>List</i> is set to On input. Default: {"configuration-Type": "From input"}
Path	path	string	Only available if List is set to Once. Points to the qualified name used in the File Reference. Must follow the qualified name encoding standard. When the Connection is set, the UI file browsing may be used to set this parameter's value. Default: /
Recursive	recursive	boolean	Whether to browse through subdirectories recursively. If false, only the files and directories at the first level of the requested path are listed. Default: false
List Directories	listDirectories	boolean	Whether to list directories. If false, only files are listed, otherwise, both are listed. Default: false

Name	ID	Type	Description
Filter	filter	string	Mandatory. Sets a specific filter for outputs. Accepted values: • None: no filter is used, all outputs occur normally. • With regular expression: only names that match the regular expression specific in Pattern are sent as output. Default: "None"
Pattern	pattern	string	The regular expression used to filter outputs. Only names that match the expression are sent as output. It uses the RE2 standard https://github.com/google/re2/wiki/Syntax 🦋 Default: ""

Input

Input	Type	Description
ref	message.fileReference	The File Reference pointing to the directory to be listed. If it points to a file, the input is replicated to the output.

Output

Output	Type	Description
ref	message.fileReference	The File Reference pointing to one file or directory resulting from the list operation. It contains the standard set of batching attributes, except for the optional message.batchCount, regarding a single operation.
error	message.error	An Error Message, in case an error was raised during an operation.

2.7.3 Monitor Files

This operator monitors file events in a directory from various storage services. Monitoring is achieved through listing files in the directory periodically.

Possible events are:

- Already exists (alreadyExists): reported for any files already present in the directory when the graph starts;
- New (new): a new file that was not present in the directory when the graph started, or a file that has been previously deleted and is recreated;
- Modified (modified): the file has been modified compared to its previous state;
- Deleted (deleted): the file previously found in the directory has been removed.

The reported file metadata is the last known one.

For each event, an output is generated as a file message with an additional `fileEvent` message attribute reporting the event. Each listing operation may also be tracked through the batching attributes.

The operator gets its connection according to the Connection property. You can either use a static connection ID from Connection Management or enter the parameters for a Manual connection.

Supported services are:

- [Alibaba Cloud Object Storage Service \(Alibaba Cloud OSS\)](#)
- [Amazon Simple Storage Service \(S3\)](#) (also supporting S3-compatible services)
- [Google Cloud Storage \(GCS\)](#)
- [Hadoop Distributed File System \(HDFS\)](#)
- [WebHDFS](#) (only supported via Connection Management)
- [Microsoft Azure Data Lake \(ADL\)](#)
- Microsoft Azure Data Lake Gen. 2 (ADL_V2)
- [Microsoft Azure Storage Blob \(WASB\)](#)
- [SDL](#)
- SSH File Transfer Protocol (SFTP)
- System Management Files (/files) (only supported via Manual connection. Refer to the file system presented on the System Management app, only for files found in `/files/`)

Note

System Management Files (/files) does not use a Connection ID. It is the file system presented in the System Management app, only for files found in /files/. The usage of the /files/ root path (or the /vrep folder) is deprecated.

Configuration Parameters

Name	ID	Type	Description
Connection (static)	connection	object	Points to the static connection to list from. Only available if <i>List</i> is set to Once . Default: {}
Connection (dynamic)	dynamicConnection	object	Points to the dynamic connection to list from. The connection configuration is detailed here. It uses the file connection attribute. Only available if <i>List</i> is set to On input . Default: {"configuration-Type": "From input"}
Path	path	string	Points to the directory to be monitored, relative to the used connection. It must be in the qualified name format. If the Connection is set with a connection ID, you may use the UI file browsing to set the path. Default: /
Recursive	recursive	boolean	Whether to monitor subdirectories recursively. If false, only the files at the first level of the requested path are monitored. Default: false

Name	ID	Type	Description
Events	events	object	Which events should be reported. Each property corresponds to an event that will be tracked if set to true. Each event is described in the beginning of this document. Default: {"alreadyExists":false,"new":true,"notModified":false,"modified":true,"deleted":false}
Filter	filter	string	Mandatory. Sets a specific filter for outputs. Accepted values: • None: no filter is used, all outputs occur normally. • With regular expression: only names that match the regular expression specific in Pattern are sent as output. Default: "None"
Pattern	pattern	string	The regular expression used to filter outputs. Only names that match the expression are sent as output. It uses the RE2 standard https://github.com/google/re2/wiki/Syntax ➡ Default: ""

Input

None

Output

Output	Type	Description
ref	message.fileReference	A file message with the an empty body. The message attribute <code>fileEvent</code> reports the event relative to the file. Possible events are described in the beginning of this document. The message also has the updated file metadata (<code>connection</code> , <code>path</code> , <code>size</code> , <code>modTime</code> and <code>isDir</code>) in the file attribute. It contains the batching attributes, except the <code>message.batchCount</code> , related to the list operation that occurs after every internal.
error	message.error	An error message, in case an error was raised during an operation.

2.7.4 Read File

Reads the content of files from various storage services.

A read operation happens according to the [Read](#) configuration. If reading **On input**, the input contains the used path. Otherwise, if reading **Once**, the path comes from the configuration.

The connection used varies with the [Connection](#) configuration. The [Connection type From input](#) is available only if reading **On input**. Otherwise, a static connection ID from **Connection Management** or a **Manual** connection can be used.

For each opeation, the output is a file message. If [Output in Batches](#) is `true`, one operation may have many outputs.

The operator ignores directory inputs and produces no output. A directory input must have `file.isDir` set to `true`. Usually these inputs come from another operator such as [List Files \[page 296\]](#).

Supported services are:

- [Alibaba Cloud Object Storage Service \(Alibaba Cloud OSS\)](#)
- [Amazon Simple Storage Service \(S3\)](#) (also supports S3-compatible services)
- [Google Cloud Storage \(GCS\)](#)
- [Hadoop Distributed File System \(HDFS\)](#)
- [Microsoft Azure Data Lake \(ADL\)](#)
- Microsoft Azure Data Lake Gen. 2 (ADL_V2).
- [Microsoft Azure Storage Blob \(WASB\)](#)
- [SDL](#)
- SSH File Transfer Protocol (SFTP).

- [System Management Files \(/files\)](#) (it does not use a Connection ID. It is the file system presented in the System Management app, only for files found in `/files/`. The usage of the `/files/` root path (or the `/vrep` folder) is deprecated.)

Configuration Parameters

Name	ID	Type	Description
Read	mode	string	Mandatory. The operation mode. Details of each mode are found below. If Output in batches is enabled, each read operation may produce many outputs. Accepted values: • On input: A read operation happens for every input received. The input informs the path to read from. • Once: A single read operation happens as soon as the graph is Running. The Path configuration informs the path to read from. Default: "On input"
Connection (static)	connection	object	Points to the static connection to read from. Only available if Read is set to Once. Default: {}
Connection (dynamic)	dynamicConnection	object	Points to the dynamic connection to read from. It uses the file connection attribute. Only available if Read is set to On input. Default: {"configuration-Type": "From input"}

Name	ID	Type	Description
Path	path	string	Points to the file to read, relative to the used connection. It must be in the qualified name format. If the Connection is set with a connection ID, you may use the UI file browsing to set the path. Only available if Read is set to Once. Default: /
Output in Batches	outputInBatches	bool	If <code>true</code>, each output uses batching attributes and has a body limited by Output batch size. Also, each operation may produce many outputs. Default: false
Output Batch Size	outputBatchSize	string	The maximum number of bytes placed in each output message. The value allows metric prefixes and an optional "i" to state binary bases (examples below). It is useful to reduce graph latency and memory usage. Only available if Output in batches is set to <code>true</code>. Example values: • "1", "1B": 1 byte • "2kb", "2KB": 2*103 bytes • "2kib", "2KiB": 2*210 bytes • "3mb", "3MB": 3*106 bytes • "3mib", "3MiB": 3*220 bytes Default: 1MB

Input

Input	Type	Description
ref	message.fileReference	A file message pointing to the file to read. The body is ignored. If it is a directory, the operator ignores it, producing no output.

Output

Output	Type	Description
file	message.file	A file message with the contents of the file in the body. The message also has the updated <code>connection</code> , <code>path</code> , and <code>size</code> values in the file attribute.
error	message.error	An Error Message, in case an error was raised during an operation.

2.7.5 Remove File

This operator removes files and directories from various storage services.

A **remove operation** takes place at every input, using the path its attributes point to.

By default, only single files are removed. If the Remove all configuration is enabled, any directory or file is removed instead.

The connection used varies with the connection configuration. The Connection type From input is available only if reading On input. Otherwise, a static connection ID from Connection Management or a Manual connection can be used.

At each successful removal, the input is replicated to the output. Only one output is given per input, even if several files are removed.

Supported services are:

- [Alibaba Cloud Object Storage Service \(Alibaba Cloud OSS\)](#)
- [Amazon Simple Storage Service \(S3\)](#) (also supporting S3-compatible services)
- [Google Cloud Storage \(GCS\)](#)
- [Hadoop Distributed File System \(HDFS\)](#)
- [WebHDFS](#) (only supported via Connection Management)
- [Microsoft Azure Data Lake \(ADL\)](#)
- Microsoft Azure Data Lake Gen. 2 (ADL_V2)

- [Microsoft Azure Storage Blob \(WASB\)](#)
- [SDL](#)
- SSH File Transfer Protocol (SFTP)
- System Management Files (/files) (only supported via Manual connection. Refer to the file system presented on the System Management app, only for files found in /files/)

i Note

System Management Files (/files) does not use a Connection ID. It is the file system presented in the System Management app, only for files found in /files/. The usage of the /files/ root path (or the /vrep folder) is deprecated.

Configuration Parameters

Name	ID	Type	Description
Connection	dynamicConnection	object	Points to the connection to remove from. By default, it uses the connection from the input header <code>file.connection</code> . Default: {"configuration-Type":"From input"}
Remove All	removeAll	bool	If set to <code>true</code> , any file or directory in the requested path is removed. Otherwise, only single files are removed, resulting in an error if the requested path points to a directory. Default: false
Ignore If Not Found	ignoreNotFound	bool	If set to <code>true</code> , any file or directory not found is ignored, instead of raising an error. Default: false

Input

Input	Type	Description
ref	message.fileReference	A File message pointing to the file or directory to be removed. The body is ignored. If it is a directory, Remove All must be set to true, otherwise an error is raised.

Output

Output	Type	Description
ref	message.file	A message identical to the one received as input, indicating the operation happened successfully.
error	message.error	An Error Message, in case an error was raised during an operation.

2.7.6 To File

This operator converts the input to a file message, taking either the payload or path. Note that the input ports are independent, and each input generates one output.

If the input comes from the port path, it is used as the path property in the resulting file message, leaving the payload empty.

If the input comes from the port in, it is used as the payload (body) of the resulting message. If in is connected to a message, it keeps the incoming file attribute or populates with an empty one if none is given, maintaining the body as the payload of the file. Payload incoming types are mapped as follows:

Type	Description
string	Used as-is
blob	Used as-is
other	Converted to JSON

Configuration Parameters

- None.

Input

Input	Type	Description
in	any.*	The payload of the file. It generates a file message using the input as the payload. If the input is a message, incoming attributes are kept, including the file properties; otherwise, they are populated with an empty path and connection.
path	any.*	The path of the file. It generates a file message as output with an empty body and the received path. If the input is a message, the body is used as the path and incoming attributes are kept, including any other file property besides the path.

Output

Output	Type	Description
file	message.file	A file with the given payload or path.

2.7.7 Write File

This operator writes files to various services.

Supported services are:

- [Alibaba Cloud Object Storage Service \(Alibaba Cloud OSS\)](#)
- [Amazon Simple Storage Service \(S3\)](#) (also supporting S3-compatible services)
- [Google Cloud Storage \(GCS\)](#)
- [Hadoop Distributed File System \(HDFS\)](#)
- [WebHDFS](#) (only supported via Connection Management)

- [Microsoft Azure Data Lake \(ADL\)](#)
- Microsoft Azure Data Lake Gen. 2 (ADL_V2)
- [Microsoft Azure Storage Blob \(WASB\)](#)
- [SDL](#)
- SSH File Transfer Protocol (SFTP)
- System Management Files (/files) (only supported via Manual connection. Refer to the file system presented on the System Management app, only for files found in /files/)

i Note

System Management Files (/files) does not use a Connection ID. It is the file system presented in the System Management app, only for files found in /files/. The usage of the /files/ root path (or the /vrep folder) is deprecated.

i Note

Use `Connection Management` for defining connections over the manual option.

Configuration Parameters

Name	ID	Type	Description
Connection	connection	object	Points to the connection to write to. Default: {"configuration-Type":"From input"}

Name	ID	Type	Description
Path Mode	pathMode	string	Indicates where the path is taken from. Accepted values: • <i>Dynamic (from input)</i>: the path is taken from the input attributes (<code>file.path</code>). • <i>Static (from configuration)</i>: uses the Path configuration parameter. • <i>Static with placeholders</i>: uses the Path configuration parameter, allowing placeholders. More details at the Path placeholders section found further below in this document. Default: "Dynamic (from input)"
Path	path	string	The path where the file is written to, relative to the connection being used. It must be in the qualified name format. If the Connection is set with a connection ID, you may use the UI file browsing to set the path. Only available if Path mode is set to Static (from configuration) or Static with placeholders. Default: /

Name	ID	Type	Description
Mode	mode	string	Affects the behavior of the operation if the file already exists. If it does not exist, the operator always creates it. Accepted values: • Overwrite: overwrites the existing file with the content • Append: appends the content to the end of the existing file. • Create Only: does not change the file if it already exists. An error may or not be reported based on the <i>If File Exists</i> configuration. Default: Create Only
If File Exists	ifExist	string	Only available if <i>Mode</i> is set to Create only. Sets whether to report an error if the file already exists. The reported error is subject to the error port semantics documented further below. Accepted values: • Fail: reports an error if the file already exists. • Ignore: ignores the input, producing no output, if the file already exists. Default: Fail

Name	ID	Type	Description
Join Batches	joinBatches	boolean	If set to false, each input is treated as a whole file. Otherwise, if the input contains the batching attributes, batches of the same file reference are joined and written in order, according to the batch index. When the operator writes the last batch, it send an output. The output attributes use the last batch input ones. This supports out-of-order batches and simultaneous files.

Input

Input	Type	Description
file	message.file	A file whose body is used as the content to be written. The <code>connection</code> and <code>path</code> values from the file attribute may be used as well, according to the operator configuration.

Output

Output	Type	Description
file	message.file	The same file as the input, but with the updated reference attributes.
error	message.error	An error message if an error was raised during an operation.

Path placeholders

Path placeholders allow dynamic values as part of the path. They can be invoked using angle brackets: the string `<counter>` is replaced by the result of the placeholder named `counter`.

Some placeholders also have a suffix, defined after the colon separator (:).

❖ Example

the string `<header:headerName>` points to the placeholder `header` with the suffix `headerName`.

Available placeholders are:

Placeholder	Description	Original Path Example	Generated Path Example
counter	An incremented integer	Take the following path: <code>/folder/ file_<counter>.txt</code>	It generates sequential operations on the paths: <code>/folder/ file_000001.txt</code> <code>/folder/ file_000002.txt</code> <code>/folder/ file_000003.txt</code>

Placeholder	Description	Original Path Example	Generated Path Example
date	The current local date in the format YYYYMMDD	Take the following path: /folder/ file_<date>.txt	It considers the server's current date (such as 01/16/2020 - January 16th 2020) and daily writing operations to generate the following paths: /folder/ file_20201216.txt /folder/ file_20201217.txt /folder/ file_20201218.txt If more than one operation is performed on the same day, the date remains the same: /folder/ file_20201216.txt /folder/ file_20201216.txt /folder/ file_20201216.txt

Placeholder	Description	Original Path Example	Generated Path Example
time	The current local time in the format HHMMSS	Take the following path: /folder/ file_<time>.txt	It considers the server's current time (such as 22h 27min 33s) and a writing operation performed at each second to generate the following paths: /folder/ file_102733.txt /folder/ file_102734.txt /folder/ file_102735.txt If more than one operation occurs at the same second, the time remains the same: /folder/ file_102733.txt /folder/ file_102733.txt
multiplicity	The multiplicity of the operator's group	Take the following path: /folder/ file_<multiplicity>.txt	It uses the multiplicity total according to where the operator is located. Default is 1 when the operator is not grouped. If the operator is grouped and <i>Multiplicity</i> is set to 3, it generates: /folder/ file_3.txt

Placeholder	Description	Original Path Example	Generated Path Example
multiplicityIndex	An integer in the range [0,multiplicity) that can be used to distinguish the multiple instances of this operator	Take the following path: /folder/ file_<multiplicityIndex>.txt	If the operator is gruped and <i>Multiplicity</i> is set to 3, three instances of the Write File operator are generated, each with a different index (0, 1 and 2). It can generate one of the following paths, depending on which Write File performed the operation: /folder/ file_0.txt /folder/ file_1.txt /folder/ file_2.txt
header	Query of the input message for a specified attribute. A suffix with the message attribute key is needed to be queried.	Take the following path: /folder/ file.<header:fileFormat>	Considering that the input port message has a fileFormat key attribute with the csv value, it generates: /folder/file.csv If key attribute is fileFormat and its value txt, it generates: /folder/file.txt This placeholder allows visting object attribute fields. For example, if the operator receives a file attribute with value { "name": "sometext.txt", "format": "txt" }, it can use the format value the following way: /folder/ file.<header:file.format>

If enabled along with the Join batches configuration, placeholders are processed only for the first batch (for example, batch with index 0).

2.8 Hadoop and Spark

Related Information

[Livy Spark Submit \[page 318\]](#)

[Submit Hadoop Job \[page 322\]](#)

2.8.1 Livy Spark Submit

This operator submits jobs to a cluster using the Livy REST API. It has 2 different modes: jar and snippet.

When one mode is chosen, the configuration tab will show only relevant configuration parameters.

1. In jar mode, you can submit an application in a way that is very similar to using spark-submit. The operator will succeed only if the underlying job is finished successfully. For instance, if a jar file is submitted to YARN, the operator status will be identical to the application status in YARN. Note that the jar file must be accessible to Livy.
2. In snippet mode, code snippets could be sent to a Livy session and results will be returned to the output port. This approach is very similar to using the Spark shell. Please, note that there are some limitations in adding jars to sessions due to LIVY-327.

As opposed to jar mode, the operator will not fail even if a code snippet sent to Livy is failed. You can change this default behavior by setting `strictCodeExecutionMode` to true in the configuration tab.

- In strict mode, the operator verifies if the output of the code snippet execution contains any errors. If any errors are found, the operator will also fail.
- In non-strict mode, the operator will ignore possible errors that happen during the snippet execution. As a consequence, users have to analyze the result manually to see if the execution is successful. This can be done by exploring the job execution output, which is sent to the output port of the operator.

Configuration Parameters

Users are responsible for setting mandatory configuration parameters. If default values are not changed, they will be sent to the target Livy endpoint, which can lead to errors. On the contrary, if (ptional)configuration parameters are not changed, they will not be included in requests to Livy.

Common configurations:

Name	ID	Type	Description
Livy Endpoint	livyEndpoint	string	Mandatory. Defines the Livy endpoint to use (please, also specify the port number). If the Livy service cannot be reached - the operator will fail during the initialization phase. Default: "http://livy-api-endpoint.com:8998"
Source Type	sourceType	string	Mandatory. Defines the type of the job that is being submitted: "jar" for using a jar-file, "snippet" - for a snippet of code. Default: "jar"
Error Handling Mode	errorHandlingMode	string	Mandatory. Defines the error handling mode: • in "default" mode, any errors from the Livy operator are immediately returned to the execution engine. As a consequence, the whole graph is failed and the remaining part of your pipeline is not executed. • in "pipeline" mode, all errors are streamed to the error port and the execution continues. This mode allows you to have different pipelines depending on the status of your Spark job submitted by Livy. Default: "default"

Name	ID	Type	Description
Proxy User	proxyUser	string	User to impersonate when starting a session or running a job. It's also used as a value of the "X-Requested-By" header in HTTP requests to avoid being blocked by the CSRF protection. If this value is empty, "X-Requested-By" will be equal to "hdfs". Default: "hdfs"
Jars	jars	string	Jars to be used in this session or batch. Default: "jar1.jar2,..."
Configuration	conf	string	The value for the "--conf" argument of spark-submit. You must input configurations exactly according to the JSON-format, and wrap it with curly braces . Default: {"key1": "value1", "key2": "value2"}
Access Token (For jar mode only)	accessToken	string	OAuth access token. Default: ""
Batch Name (For jar mode only)	batchName	string	The name of the batch. Default: "default batch name"
Jar Path (For jar mode only)	jarPath	string	Mandatory. Path to jar to be submitted. Default: "hdfs://path-to-jar"
Class Name (For jar mode only)	className	string	Mandatory. Name of the class to be executed in jar. Default: "org.com.smth.className"
Arguments (For jar mode only)	args	string	Command line arguments for the application. Default: "arg1,arg2,..."

Name	ID	Type	Description
Session Name (For snippet mode only)	sessionName	string	The name of this session. Default: "default session name"
Code Snippet (For snippet mode only)	snippet	string	Mandatory. Snippet of code that has to be submitted. Default: "snippet of code"
Snippet Type (For snippet mode only)	snippetType	string	Mandatory. Defines the kind of session that should be created for snippet execution (language of snippet). Possible values: spark, pyspark, pyspark3 or sparkr. Default: "spark"
Strict Snippet Execution Mode (For snippet mode only)	strictSnippetExecutionMode	bool	Mandatory. Defines whether operator is tolerant to errors in snippet execution output, for example, switches strict and not strict modes of snippet submitting. Default: "false"

Input

Input	Type	Description
inport	string	Accepts path to jar or snippet of code that has to be submitted. Input signal initiates job submitting.

Output

Output	Type	Description
out	string	The following information will be sent to this port if the submitted job finishes successfully: • <code>sourceType jar</code>: the corresponding Livy log • <code>sourceType snippet</code>: output of the executed snippet
error	string	If the submitted job fails and the Livy operator is using <code>pipeline</code> error handling mode, the error will be routed to this port.

2.8.2 Submit Hadoop Job

The Submit Hadoop Job operator is used to submit jobs to Hadoop clusters provided by different cloud providers. Currently the only supported Hadoop cluster type is Google Dataproc.

Supported services:

- Google Dataproc (Spark, PySpark, SparkSQL, Hive)

The operator has one input port: `jobConfig`, which is optional and used to dynamically overwrite static configurations. A dynamic job configuration is represented as a JSON object, whose structure depends on a job type and a service provider. See examples of job configurations below.

The operator has one output port: `result`. The output is represented as a message with attributes containing metadata. For example, each message will have a cluster and job configuration in its attributes as well as specific attributes for each job type. The message contains a key named `message.error`, which is `true` in case of failure and `false` in case of success.

A dynamic Dataproc Spark job configuration can look like the example below (include only needed fields):

```
{
  "sparkJobSpec": {
    "mainJarFileUri": "gs://bucket/file.jar",
    "mainClass": "org.apache.spark.ClassName",
    "args": ["10"],
    "fileUris": ["gs://bucket/file1.txt", "gs://bucket/file2.txt"],
    "jarFileUris": ["gs://bucket/jar1.jar", "gs://bucket/jar2.jar"],
    "archiveUris": ["models1.zip", "models2.zip"],
    "properties": {"prop1": "v1", "prop2": "v2"},
    "dataprocSparkDriverLogLevels": {"org.apache.spark": "DEBUG"}
  },
  "labels": {"label1": "v1", "label2": "v2"}
}
```

Next, an example for dynamic Dataproc Spark SQL job configuration (include only needed fields):

```
{
  "sparkSqlJobSpec": {
    "queryList": {"queries": ["DROP TABLE t2"]},
    "scriptVariables": {"a": "b", "spark.property": "20"},
    "fileUri": ["gs://bucket/file1.txt", "gs://bucket/file2.txt"],
    "jarFileUri": ["gs://bucket/jar1.jar"],
    "properties": {"prop1": "v1", "prop2": "v2"},
    "dataprocSparkDriverLogLevels": {"org": "DEBUG"}
  },
  "labels": {"label1": "v1", "label2": "v2"}
}
```

A dynamic Dataproc PySpark job configuration can look like (include only needed fields):

```
{
  "pySparkJobSpec": {
    "mainPythonFileUri": "gs://bucket/file.py",
    "args": ["l0"],
    "pythonFileUri": ["gs://bucket/f1.py", "gs://bucket/f2.py"],
    "fileUri": ["gs://bucket/file1.txt", "gs://bucket/file2.txt"],
    "jarFileUri": ["gs://bucket/jar1.jar", "gs://bucket/jar2.jar"],
    "archiveUri": ["models1.zip", "models2.zip"],
    "properties": {"prop1": "v1", "prop2": "v2"},
    "dataprocSparkDriverLogLevels": {"org.apache.spark": "DEBUG"}
  },
  "labels": {"label1": "v1", "label2": "v2"}
}
```

Finally, a Dataproc Hive job configuration example (include only needed fields):

```
{
  "hiveJobSpec": {
    "queryList": {"queries": ["DROP TABLE t2"]},
    "scriptVariables": {"a": "b", "hive.property": "20"},
    "fileUri": ["gs://bucket/file1.txt", "gs://bucket/file2.txt"],
    "jarFileUri": ["gs://bucket/jar1.jar"],
    "properties": {"prop1": "v1", "prop2": "v2"},
    "dataprocSparkDriverLogLevels": {"org": "DEBUG"}
  },
  "labels": {"label1": "v1", "label2": "v2"}
}
```

See the `com.sap.demo.hadoop.dataproc` example graph to check how to use this operator.

Configuration Parameters

Universal Configurations

Name	ID	Type	Description
Job Type	<code>dataproc.JobType</code>	string	Mandatory: A Dataproc job type. Default: "spark"

Name	ID	Type	Description
Labels	dataprocLabels	string	Labels to associate with the Dataproc job. Default: ""
Driver Log Levels	dataprocSparkDriverLogLevels	string	Spark driver log levels represented as a JSON object. Default: ""
Jar File URIs	dataprocJarFileUris	string	Jar file URIs separated by ','. Default: ""
Spark Command Arguments	dataprocSparkArgs	string	Spark job arguments separated by ','. Default: ""
Spark Configuration	dataprocSparkConf	string	Spark configurations. Similar to --conf parameters of spark-submit but as a JSON object. Default: ""
File URIs	dataprocSparkFileUris	string	Spark file URIs separated by ','. Default: ""
Archive URIs	dataprocSparkArchiveUris	string	Spark archive URIs separated by ','. Default: ""
Connection	dataprocConnection	object	GCP Dataproc connection details used to connect to the service. Default: ""
Connection Parameters			
Name	ID	Type	Description
Configuration Type	configurationType	string	The type of connection information is used: Manual (user input) or retrieved by the Connection Management Service. Default: ""
Connection ID	connectionID	string	The ID of the connection information to retrieve from the Connection Management Service. Default: ""

Name	ID	Type	Description
Connection Properties	connectionProperties	object	All the connection properties for manual input. Default: ""
Project ID	projectId	string	Mandatory. The GCP Project ID. Default: ""
Cluster Name	clusterName	string	Mandatory. Name of the cluster the job is sent to. Default: ""
Region	region	string	Mandatory. The region in which the cluster is located. Default: ""
Zone	zone	string	The zone in which the cluster is located. Default: ""
Keyfile	keyfile	string	JSON key file used to authenticate and access secured clusters. Default: ""

Spark Configurations

Name	ID	Type	Description
Main Jar File URI	dataprocSparkMainJarFileUri	string	Mandatory. The main Spark jar file URI. Default: ""
Class Name	dataprocSparkClassName	string	Mandatory. A class name to execute. Default: ""

PySpark Configurations

Name	ID	Type	Description
Main Python File URI	dataprocSparkMainPythonFileUri	string	Mandatory. The main PySpark file URI. Default: ""
Python File URIs	dataprocSparkPythonFileUris	string	Additional Python file URIs separated by ','. Default: ""

Hive Configurations

Name	ID	Type	Description
Continue on failure	dataprocHiveContinueOnFailure	string	Mandatory. A flag that determines whether to continue executing Hive queries if a query fails. Default: ""
Hive Configuration	dataprocHiveConf	string	Hive configurations represented as a JSON object. Default: ""
Query File URI	dataprocHiveQueryFileUri	string	A file URI with Hive queries. Default: ""
Queries	dataprocHiveQueries	string	A text field for Hive queries. i Note It cannot be used in conjunction with Query Files. Default: ""
Script Variables	dataprocHiveScriptVariables	string	A field for Hive script variables. Equivalent to the Hive command: SET name="value". Default: ""

SparkSQL Configurations

Name	ID	Type	Description
Query File URI	dataprocSparkSqlQueryFileUri	string	A file URI with Spark SQL queries. Default: ""
Queries	dataprocSparkSqlQueries	string	A text field for Spark SQL queries. i Note It cannot be used in conjunction with Query Files. Default: ""
Script Variables	dataprocSparkSqlScriptVariables	string	A field for Spark SQL variables. Equivalent to the Spark SQL command: SET name="value". Default: ""

Note

All Python and Jar files referenced must be from a HDFS or HDFS-compatible file system, or located in a Google Cloud Storage bucket. For more information, you can refer to the Dataproc v1 API documentation at https://developers.google.com/resources/api-libraries/documentation/dataproc/v1/csharp/latest/namespaceGoogle_1_1Apis_1_1Dataproc_1_1v1_1_1Data.html 📖

Input

Input	Type	Description
<code>jobConfig</code>	string	A JSON object representing a dynamic job configuration, which overwrites or appends the static properties.

Output

Output	Type	Description
<code>result</code>	message	The operator sends a message with the used jobs' attributes and a flag whether the job was successful when finished.

2.9 Multiplexer

The multiplexer maps the input of any port to all of its output ports. Additional input and output ports may be added as needed. The type of all ports is `any.*` (equivalent to `interface{}` in Go)..

Caution: All the input connections have to be of the same type.

Configuration Parameters

- None.

Input

- None (by default ports may be added in extensions and in the graph).

Output

- None (by default ports may be added in extensions and in the graph).

2.10 Processing

Related Information

[Command Executor \[page 329\]](#)
[Constant Generator \[page 330\]](#)
[Data Generator \[page 331\]](#)
[Go Message Operator \[page 332\]](#)
[Go Operator \[page 333\]](#)
[Go String Operator \[page 339\]](#)
[JavaScript Operator \[page 340\]](#)
[JavaScript \(v2\) \[page 346\]](#)
[JS Message Operator \[page 352\]](#)
[JS String Operator \[page 353\]](#)
[Message Generator \[page 354\]](#)
[Process Executor \[page 355\]](#)
[Python 3 Operator \[page 356\]](#)
[R Client 3 \[page 364\]](#)
[Stdin/out Port Muxer \[page 376\]](#)
[Streaming Analytics Projects \[page 377\]](#)

2.10.1 Command Executor

The Command Executor operator executes the given command for each arrival of a message on the stdin port. The arriving message is provided on standard input. The complete standard error and standard output of the command is available on stderr and stdout respectively after completion of the command.

Configuration Parameters

Parameter	Type	Description
cmdLine	string	The command line to be executed.

Input

Input	Type	Description
stdin	blob	The input for standard input.

Output

Output	Type	Description
stdout	blob	The output from standard output.
stderr	blob	The output from standard error.

2.10.2 Constant Generator

The Constant Generator operator represents a constant string that may be emitted into the graph. It can do so on a regular basis ('pulse' mode) or only once ('once' mode). The 'in' port can be used only to start the string constant operator when an incoming signal is received.

Configuration Parameters

Parameter	Type	Description
content	string	The content that the string constant emits.
mode	string	There are three modes: once, pulse and trigger: • once: In this mode, the content is emitted only once. • pulse: In this mode, the content is emitted repeatedly by the specified duration. • trigger: In this mode, the content is emitted every time a signal is received by the in-port.
duration	string	If in pulse mode, the duration between ticks where the content is emitted.
counter	int	The number of times this component has to receive any incoming signal to start running.

Input

Input	Type	Description
in	any	Incoming signals that reduce the 'counter' and trigger output in 'trigger' mode.

Output

Output	Type	Description
out	string	The content.

2.10.3 Data Generator

The Data Generator operator is an extension of the base operator `com.sap.system.jsengine`. It generates random sample data every 500ms.

Configuration Parameters

None

Input

None

Output

Output	Type	Description
output	string	Generated data

2.10.4 Go Message Operator

This Go operator is an example of how to use message type with a Go script.

Configuration Parameters

Parameter	Type	Description
code language	string	The programming language used in the code. Default: "go"
script	string	Mandatory. The inline script to be executed. If the script starts with <code>file: //</code> , then the script file is executed. Default: ""

Input

Input	Type	Description
input	message	Any message which body should be converted to an uppercase string.

Output

Output	Type	Description
output	message	The message which body contains an uppercase string.

For further information about customizing the Go operator, refer to the base [Go Operator \[page 333\]](#) documentation.

2.10.5 Go Operator

The Go operator allows you to execute Go code within a graph.

It supports sending outputs from input handlers or functions that run periodically. Other functionality, such as logging, reading the operator configuration parameters, and sending responses to requests are supported and documented further below.

Configuration Parameters

Name	ID	Type	Description
Code Language	codeLanguage	string	The programming language used in the code. Default: "go"
Script	script	string	Mandatory. The inline script to be executed. If the script starts with "file://", then the script file is executed. Script File: To reference a script file, the <code>script</code> configuration must start with the prefix <code>file://</code> followed by the file path. Default: ""
Maximum Concurrency	maxConcurrency	int	The maximum number of concurrent requests that are handled per operator instance. The message order is preserved only when value is 1. Default: 1

Input

None

Output

None

Creating a Go Operator Extension

To create an extension, follow these steps:

1. In the Modeler UI, go to the [Operators](#) tab and click the **+** button to add a new operator extension.
2. Select the base operator [Go Operator](#), give it a name, and click [OK](#).
3. Add the desired input and output ports.
4. Add your Go script either as:
 - An inline `script` parameter in the extension configuration.
 - As a separate file, using the prefix `file://` to specify the file name in the script parameter.
5. Make sure your script belongs to `package main`.
6. If you need something to run once on startup, define it inside `func Setup()`.
7. To run a cleanup procedure once the graph shuts down, define the `func Cleanup()` with the required procedure.
8. Define some function to deal with inputs (documented below in the Input Handling Script section) or to run periodically (documented below in the Interval-Based function section).
9. Save your operator extension.

Interval-Based Script

To build a basic interval-based script, you can follow this example:

```
package main
// here implement the function with the input port name,
// so data coming from this input will call this function
func In(val interface{}){
    message := val.(map[string]interface{})
    body := message["Body"]
    attr := message["Attributes"].(map[string]interface{})
    encoding := message["Encoding"]
    // ...
}
```

Input Handling Script

To build an input handling script, you can follow the example below.

When the operator has an input port named `in` and an output port named `out`, we may use the following script:

```
package main
var Out func(interface{})
// here implement the function with the input port name,
// so data coming from this input will call this function
func In(val interface{}){
    strValue := val.(string)
    strValue += "test"
    Out(strValue)
    // ...
}
```

Script Details

The script must belong to the `package main`. It supports the following functionalities:

Input Handling Script

To call a function based on an input incoming data, the function name should be the same as the port name, and data from that port will be sent through the function's parameter. The function must be exported (for example, it begins with an upper case letter).

All input received as parameters of the `interface{}` type, making necessary the usage of type assertion to manipulate it. To use message type, the value received through the input port should be converted to the type `map[string]interface{}`, then its members can be accessed with the keys "Attributes", "Encoding" and "Body", as in the following example.

Suppose the operator has an input port named in of message type::

```
package main
// here implement the function with the input port name,
// so data coming from this input will call this function
func In(val interface{}){
    message := val.(map[string]interface{})
    body := message["Body"]
    attr := message["Attributes"].(map[string]interface{})
    encoding := message["Encoding"]
    // ...
}
```

Interval-Based Function

To create a function that runs periodically (every X amount of milliseconds), the global variable `Timer` must be set with the desired interval, and the function must be implemented as `func Tick()`.

The `Timer` value may be either an `int64` with the interval in milliseconds or a `time.Duration`.

Interval-based function example:

```
package main
var Timer int64 = 500
func Tick() {
    // function to be called every 500ms
    // changes here
}
```

Sending Outputs

To send an output via an output port, a global exported variable must be declared, matching the desired output port's name. Its type must be `func(interface{})`.

Output functions example:

Suppose the operator has an input port named in:

```
package main
// sends a value to port "out1" when called
var Out1 func(interface{})
// sends a value to port "out2" when called
var Out2 func(interface{})
```

Setup and Cleanup

To run a procedure before any other function from the script is called, the Setup function may be set. That function is **not** able to send outputs.

If some cleanup is needed when the graph is shutdown, the Cleanup function may be set. That function is **not** able to send outputs.

In the example below, the operator has an output port named out:

```
package main
var Log func(string)
func Setup() {
    Log("graph started")
}
func Cleanup() {
    Log("graph finished")
}
```

Handling Messages

To modify a message, a copy of it should be done first. A helper function CopyMessage is provided for that sake. It only does shallow copy of the message and its attributes. Further changed values should be copied separately.

Suppose the operator has two input ports named in and in2 and an output port named out, both of type message:

```
package main
var CopyMessage func(map[string]interface{}) (map[string]interface{}, error)
var Out func(interface{})
func In(val interface{}){
    // the message must be copied before modifying it
    m, err := CopyMessage(val.(map[string]interface{}))
    if err != nil {
        panic(err)
    }
    // modifying an attribute reference
    // no further copy needed
    m["Attributes"].(map[string]interface{})["foo"] = "bar"
    // modifying the body's reference
    // no further copy needed
    m["Body"] = []byte("my-new-body")
    Out(m)
}
func In2(val interface{}){
    // the message must be copied before modifying it
    m, err := CopyMessage(val.(map[string]interface{}))
    if err != nil {
        panic(err)
    }
    attrs := m["Attributes"].(map[string]interface{})
    // modifying an attribute value
    // a copy is needed
    oldFoo := attrs["foo"].([]string)
    newFoo := make([]string, len(oldFoo)+1)
    // append "foo" to the slice
    copy(newFoo, oldFoo)
    newFoo[len(newFoo)-1] = "foo"

    // modifying the body's contents
    // a copy is needed
    oldBody := m["Body"].([]byte)
    newBody := make([]byte, len(oldBody))
```



```
// set the first byte to 1
copy(newBody, oldBody)
newBody[0] = 0x01

attrs["foo"] = newFoo
m["Body"] = newBody
Out(m)
}
```

Handling Multiplicity

Two variables can be declared to obtain multiplicity-related information:

- **Multiplicity**: the multiplicity of the operator's group.
- **MultiplicityIndex**: an integer in the range [0,Multiplicity] that can be used to distinguish the multiple instances of this operator.

Both variables must be of type `int`.

Variable declaration:

```
var Multiplicity int
var MultiplicityIndex int
```

Logging

To log messages during the Go operator execution, a function called `Log` must be defined and its parameter is string type. The formatted version `Logf` is also available with two parameters: `string, ...interface{}`. All logs will be added as Debug logs on the Modeler's tracer.

Logging example:

```
package main
var Log func(string)
var Logf func(string, ...interface{})
func Setup() {
    Log("starting go operator")
}
func In(val interface{}){
    strValue := val.(string)
    strValue += "test"
    Logf("input:%s", strValue)
    // ...
}
```

Error logging is also supported, following the same `Log` signature formats, with the following function names: `Error` or `Errorf`. Note that the error does not stop the graph, only writing the error to the tracer.

Configuration

Configuration calls provide the ability to read the configuration parameters provided in the operator. The function parameter should be the property title.

The function `AsObject` is also supported, and retrieves the whole operator configuration as a map, where each member points to a parameter with the configuration name as its key.

Configuration retrieval example:

Suppose we have configuration parameters with the following keys: `myStr`, `myInt`, `myBool`, `myObj`:

```
package main
```

```

var GetString func(string) string
var GetInt func(string) int
var GetBool func(string) bool
var GetObject func(string) map[string]interface{}
var AsObject func() map[string]interface{}
func In(val interface{}){
    strValue := GetString("myStr")
    intValue := GetInt("myInt")
    boolValue := GetBool("myBool")
    objValue := GetObject("myObj")
    entireConfig := AsObject()
    if entireConfig["myStr"].(string) == strValue {
        // this is true as these are equivalent
    }
    // ...
}

```

Operator Directory

To access the operator directory, a configuration value is available: `workingDirectory`. It can be accessed through the `GetString` function. Supposing the operator ID is `com.sap.system.myGoOp`, the following Go code would retrieve the operator path `files/vflow/operators/com/sap/system/myGoOp/`:

```

package main
var GetString func(string) string
func getDir() {
    workingDir := GetString("workingDirectory")
}

```

Message Response

`SendResponse` sends either a message response or error as the response to a message request, and may be called from the Go script.

Message response example:

```

package main
var SendResponse func(map[string]interface{}, map[string]interface{}, error)
func Input(val interface{}){
    message := val.(map[string]interface{})
    body := message["Body"]
    attr := message["Attributes"].(map[string]interface{})
    encoding := message["Encoding"]
    SendResponse(message, message, nil)
    // ...
}

```

Graph Handle

The `GetGraphHandle` function provides the current running graph handle as a string.

Graph handle example:

```

package main
var GetGraphHandle func() string
func Input(val interface{}){
    handle := GetGraphHandle()
    // ...
}

```

2.10.6 Go String Operator

This Go operator increments and outputs a counter value with the input, based on a Go script.

Configuration Parameters

Parameter	Type	Description
code language	string	The programming language used in the code. Default: "go"
script	string	Mandatory. The inline script to be executed. If the script starts with <code>file: //</code> , then the script file is executed. Default: ""

Input

Input	Type	Description
input	string	Any string that should be output with the counter.

Output

Output	Type	Description
output	string	The string appended to counter value.

For further information about customizing the Go String operator, refer to the base [Go Operator \[page 333\]](#) documentation.

2.10.7 JavaScript Operator

The JavaScript operator allows for execution of JavaScript snippets within a graph.

Configuration Parameters

Name	ID	Type	Description
codelanguage	codelanguage	string	The programming language used in the snippet. Default: "javascript"
script	script	string	Mandatory. The inline script to be executed. If the script starts with <code>file://</code> , then the script file is executed. Default: ""

Input

None

Output

None

Creating a JavaScript Extension

1. Create an extension of the JavaScript component and add the desired in and out ports.
2. Add your JavaScript snippet as an inline "script" parameter in extension configuration or put the code into a separate file and specify the filename with the prefix `file://` to reference it.

To create an extension, follow these steps:

1. In the Modeler UI, go to the [Operators](#) tab and click the **+** to add a new operator extension.
2. Select the base operator [Javascript Operator](#), name it, and click **OK**.
3. Add the desired input and output ports.
4. Add your JavaScript as one of the following:

- an inline script parameter in the extension configuration;
 - as a separate file, using the prefix `file://` to specify the file name in the script parameter;
5. Save your operator extension.

Basic Example

The following examples count all incoming messages on the port `input` and writes the count to the port `output`.

```
var counter = 0
$.setPortCallback("input",onInput)
function onInput(ctx,s) {
    counter++
    $.output(counter)
}
```

Generators

Generators are functions that will be executed before the event processing loop starts.

```
var counter = 0
$.addGenerator(gen)
$.addGenerator(gen)
function gen(ctx) {
    for (var i = 0; i < 3; i++ ) {
        $.output(counter)
        counter++
    }
}
```

This example will produce values 0, 1, 2, 3, 4, 5 on the output port `output`.

Timers

Timers will be called at given intervals. Timers are not preemptive. Thus, the given interval provides only the lower bound of the interval at which the timer function will be called. All timers will be stopped prior to the execution of `shutdown` handlers.

Timer-based function example

The example below produces the values 0, 1, 2, and so on on the port `output` until the graph is shut down.

```
var counter = 0
$.addTimer("1s",t1)
function t1(ctx) {
    $.output(counter)
    counter++
}
```

The function `addTimer()` returns a handler that can be used to disable later through the function `removeTimer()`.

Disabling a Timer

In the following example, the function `t1()` will be called three times. After that, the timer will be removed and `t1()` will not be called anymore.

```
var counter = 0
var timerId = $.addTimer("100ms",t1)
function t1(ctx) {
    counter++
    console.log("in t1() counter == ", counter)
    $.output(counter)
    if (counter == 3)
        $.removeTimer(timerId)
}
```

Handling Messages

To modify a message, a copy of it should be done first. A helper function `CopyMessage` is provided for that purpose. It does only a shallow copy of the message and its attributes. Further changed values should be copied separately.

Suppose the operator has two input ports named `in` and `in2` and an output port named `out`, both of type `message`:

```
$.setPortCallback("in", onIn)
$.setPortCallback("in2", onIn2)
function onIn(ctx,s) {
    // the message must be copied before modifying it
    var m = $.copyMessage(s);
    // modifying an attribute reference
    // no further copy needed
    m.Attributes["foo"] = "bar";
    // modifying the body's reference
    // no further copy needed
    m.Body = "my-new-body"
    $.out(m)
}
function onIn2(ctx,s) {
    // the message must be copied before modifying it
    var m = $.copyMessage(s);
    // modifying an attribute value
    // a copy is needed
    var oldFoo = m.Attributes["foo"];
    if (Array.isArray(oldFoo)) {
        var newFoo = oldFoo.slice() + "foo";
        m.Attributes["foo"] = newFoo;
    }
    // modifying the body's contents
    // a copy is needed
    // set the first value to 1
    var oldBody = m.Body;
    if (Array.isArray(oldBody)) {
        var newBody = oldBody.slice();
        newBody[0] = 0x01;
        m.Body = newBody;
    }
}
```

```
$.out(m)
}
```

Shutdown Handlers

Shutdown handlers are functions that will be executed after event processing loop execution.

This example outputs two values when the graph reaches its termination state. Each value represents the number of inputs received on the port `input`.

For example, if the operator received seven inputs, it will output 7 and 7 when the graph is terminating.

```
var counter = 0
$.setPortCallback("input",onInput)
function onInput(ctx,s) {
    counter++
}
function shutdown(ctx) {
    $.output(counter)
}
$.addShutdownHandler(shutdown)
$.addShutdownHandler(shutdown)
```

This example outputs two values when the graph reaches its termination state. Each value represents the number of inputs received on the port `input`.

Configuration API

Configuration API provides the possibility to read configurations provided via graph description.

```
function gen(ctx) {
    $.str($.config.getString("str"))
    $.int($.config.getInt("int"))
    $.bool(""+$.config.getBool("bool"))
}
$.addGenerator(gen)
```

The example above will read the operator's configurations and provide the given values on the output ports.

Graph Termination

The `done` function can be used to signal this operator has finished its execution. This will allow the graph to transition to `completed` if the other operators have finished their execution successfully.

The `fail` function raises an error that sets the graph's status to `dead`. The message provided to fail is truncated after 64k characters.

The example below terminates a graph, succeeding if the received value from the port `input` is even, or failing otherwise:

```
$.setPortCallback("input", onInput)
function onInput(ctx,s) {
    v = parseInt(s)
    if (v % 2 === 0) {
        $.done()
    } else {
        $.fail("unexpected value received")
    }
}
```

Handling Multiplicity

There are two functions that provide multiplicity-related information:

- `$.getMultiplicity()`: returns the multiplicity of the operator's group.
- `$.getMultiplicityIndex()`: returns an integer in the range `[0$.getMultiplicity() ]` that can be used to distinguish the multiple instances of this operator.

Message Response Handling

The `sendResponse` function provides the ability to send either a response or error message to the request-response initiating operator. This function is only valid if the incoming data is of type message and it contains the response callback as part of a request-response message exchange.

The example below sends a response or error depending on the current counter value.

```
var counter = 0
function onInput(ctx,s) {
    counter++
    if (counter < 100) {
        var msg = {};
        msg.Body = "pong " + counter;
        $.sendResponse(s, msg, null);
    } else {
        $.sendResponse(s, null, Error("Usage limit reached " + counter));
    }
}
```

Send Message to Logging App

The `log` function enables users to send messages to the SAP Data Intelligence Logging app. The parameters for this function are: `message`, `severity`, `messageCode`, and `details`. The `severity` parameter is optional and uses one of the following enum values:

- `logSeverity.ERROR`

- `logSeverity.INFO`
- `logSeverity.WARNING`

The `messageCode` and `details` parameters are optional as well.

The function awaits the logging API call to finish. The client is set to a 10-second timeout for the client communication. If an error occurs or timeout is reached, the log message will be logged in the tracer, with the same severity passed as parameter to the function. Other errors will kill the graph. The Log API should provide strong guarantees.

The following example sends a message with `INFO` severity to the logging app, a value for the `messageCode` and `details` parameters.

```
$.log("message to log", $.logSeverity.INFO, "10003", "this is a detailed message to the log message");
```

Subgraph (Call a Graph From a JavaScript Operator)

The `instantiateGraph` function provides the possibility to call a graph from the JavaScript operator. You can provide parameters for the graph and handlers for output ports. `instantiateGraph` returns a graph object when graph is started or fail script execution. The graph object has the following methods:

- `wait` blocks script execution until subgraph execution is finished; `wait` returns the graph status.
- `stop` initiates subgraph termination. It returns without waiting for graph termination to be completed.

```
$.addGenerator(gen)
// This example shows basic usage of a subgraph
// It calls a graph with a parameter, provides input to a port,
// checks a port's output and stops the graph
var graphName = "com.sap.demo.subgraph.call-and-wait.sub"
var subgraphInput = "subgraph-input"
var paramValue = "value1"
function gen(ctx) {
  // start subgraph
  var g
  try {
    g = $.instantiateGraph(graphName,
      {"param1": paramValue},
      // handle subgraph output port with name 'output'
      {"output":function(ctx,s){
        // check subgraph output
        if (s !== subgraphInput + paramValue) {
          $.fail("Unexpected subgraph output: " + s + " Expected: " + prefix
+paramValue)
        }
        // stop subgraph
        g.stop()
      }}
    )
  } catch(e) {
    $.fail(e.message)
    return
  }
  // write string in subgraph input port with name 'input'
  g.input(subgraphInput)
  // wait for graph execution is finished
  status = g.wait()
  // check graph status
```

```

    if (status !== $.graphStatus.completed) {
      $.fail("Subgraph is failed")
    } else {
      $.done()
    }
  }
}

```

2.10.8 JavaScript (v2)

The JavaScript (v2) operator allows you to execute JavaScript snippets within a graph.

It works similarly to the Javascript operator and supports the same API.

However, its execution differs from that of the Javascript operator when a blocking call to a Go function is performed (for example, when sending data out from an output port):

- Javascript operator: blocks it until this blocking call returns. Until then, no part of the code can be executed
- Javascript (v2) operator: allows another part of the code to be executed until the blocking call returns, when its `allowInterrupt` property is set to `true`. This behavior can be considered as a workaround for the missing asynchronous call method from the script code to a Go function in the Javascript operator API.

Configuration Parameters

Name	ID	Type	Description
Code Language	codelanguage	string	The programming language used in the snippet. Default: "javascript"
Script	script	string	Mandatory. The inline script to be executed. If the script starts with <code>file://</code> then the script file is executed. Default: ""
Allow Interrupt	allowInterrupt	bool	Possibility to interrupt a blocking call to a Go function so that another part of the script code can be executed Default: true

Input

None

Output

None

Creating a JavaScript Extension

To create an extension, follow the steps below:

1. In the Modeler, go to the [Operators](#) tab and click the **+** button to add a new operator extension.
2. Select the base operator [Javascript Operator](#), give it a name, and click **OK**.
3. Navigate to your newly created extension and:
 1. Add the desired input and output ports;
 2. Add your JS script either as: a) an inline script parameter in the extension configuration; b) a separate file, using the prefix `file://` to specify the file name in the script parameter.
 3. Add your JavaScript snippet as an inline `script` parameter in the extension configuration or put the code into a separate file and use the prefix `file://` to specify the file name.
4. Save your operator extension by clicking the disk icon.

Basic Example

The following example counts all incoming messages on the port `input` and writes the count to the port `output`.

```
var counter = 0
$.setPortCallback("input",onInput)
function onInput(ctx,s) {
    counter++
    $.output(counter)
}
```

Generators

Generators are functions that are executed before the event processing loop starts

The example below produces values 0, 1, 2, 3, 4, 5 on the port output:

```
var counter = 0
$.addGenerator(gen)
$.addGenerator(gen)
function gen(ctx) {
    for (var i = 0; i < 3; i++ ) {
        $.output(counter)
        counter++
    }
}
```

Timers

Timers are called at given intervals. Timers are not preemptive. Thus, the given interval provides only the lower bound of the interval at which the timer function is called. All timers are stopped prior to the execution of shutdown handlers.

This example produces the values 0, 1, 2... on the port output until the graph is shut down.

```
var counter = 0
$.addTimer("1s",t1)
function t1(ctx) {
    $.output(counter)
    counter++
}
```

The function `addTimer()` returns a handler that can be used to disable the timer later through the function `removeTimer()`.

In this example, the function `t1()` is called three times. After that, the timer is removed and `t1()` is no longer called.

```
var counter = 0
var timerId = $.addTimer("100ms",t1)
function t1(ctx) {
    counter++
    console.log("in t1() counter == ", counter)
    $.output(counter)
    if (counter == 3)
        $.removeTimer(timerId)
}
```

Handling messages

To modify a message, you should first make a copy of it. A helper function `CopyMessage` is provided for that sake. It only makes a shallow copy of the message and its attributes. Further changed values should be copied separately.

Assuming the operator has two input ports named `in` and `in2` and an output port named `out`, both of type `message`:

```
$.setPortCallback("in", onIn)
$.setPortCallback("in2", onIn2)
function onIn(ctx,s) {
    // the message must be copied before modifying it
    var m = $.copyMessage(s);
    // modifying an attribute reference
    // no further copy needed
    m.Attributes["foo"] = "bar";
    // modifying the body's reference
    // no further copy needed
    m.Body = "my-new-body"
    $.out(m)
}
function onIn2(ctx,s) {
    // the message must be copied before modifying it
    var m = $.copyMessage(s);
    // modifying an attribute value
```

```

    // a copy is needed
    var oldFoo = m.Attributes["foo"];
    if (Array.isArray(oldFoo)) {
        var newFoo = oldFoo.slice() + "foo";
        m.Attributes["foo"] = newFoo;
    }
    // modifying the body's contents
    // a copy is needed
    // set the first value to 1
    var oldBody = m.Body;
    if (Array.isArray(oldBody)) {
        var newBody = oldBody.slice();
        newBody[0] = 0x01;
        m.Body = newBody;
    }
    $.out(m)
}

```

String Coding API

String Coding API provides the possibility to convert a byte array encoded in UTF-8 into a string.

This example assumes that the input data is a message with its `body` being a byte array encoded in UTF-8 and converts this body into a string.

```

$.setPortCallback("input",onInput);
function onInput(ctx, s) {
    var bodystr = $.decodeString(s.Body);
    // do some processing on this bodystr string
    // ...
    $.output(bodystr);
}

```

Shutdown Handlers

Shutdown handlers are functions that are executed after event processing loop execution.

The example below outputs two values when the graph reaches its termination state. Each value represents the number of inputs received on the port input.

For example: if the operator receives 7 inputs, it outputs 7 and 7 when the graph is terminating:

```

cvar counter = 0
$.setPortCallback("input",onInput)
function onInput(ctx,s) {
    counter++
}
function shutdown(ctx) {
    $.output(counter)
}
$.addShutdownHandler(shutdown)
$.addShutdownHandler(shutdown)

```

Configuration API

Configuration API provides the possibility to read the operator configuration values provided via graph description.

This example reads the operator configuration values `myStr`, `myInt` and `myBool` using a Generator function.

```
function gen(ctx) {
  $.str($.config.getString("myStr"))
  $.int($.config.getInt("myInt"))
  $.bool(""+$.config.getBool("myBool"))
}
$.addGenerator(gen)
```

Graph Termination

The `done` function can be used to signal the operator has finished its execution. This allows the graph to transition to completed if the other operators finished their execution successfully.

The `fail` function raises an error that sets the graph's status to *dead*.

The example below terminates the operator if the input value is even. It raises an error otherwise:

```
$.setPortCallback("input", onInput)
function onInput(ctx,s) {
  v = parseInt(s)
  if (v % 2 === 0) {
    $.done()
  } else {
    $.fail("unexpected value received")
  }
}
```

Handling Multiplicity

There are two functions that provide multiplicity-related information:

- `$.getMultiplicity()`: returns the multiplicity of the operator's group.
- `$.getMultiplicityIndex()`: returns an integer in the range `[0, $.getMultiplicity())` that can be used to distinguish the multiple instances of this operator.

Send Message to Logging App

The `log` function enables users to send messages to the Data Intelligence Logging app. The parameters for this function are: `message`, `severity`, `messageCode`, and `details`. The `severity` parameter is optional and uses one of the following enum value:

- logSeverity.ERROR
- logSeverity.INFO
- logSeverity.WARNING

The messageCode and details parameters are optional as well.

The function awaits the logging API call to finish. The client is set to a 10-second timeout for the client communication. If an error occurs or timeout is reached, the log message is logged in the tracer, with the same severity passed as parameter to the function.

The example below sends a message with `INFO` severity to the logging app, a value for the `messageCode` and `details` parameters.

```
$.log("message to log", $.logSeverity.INFO, "10003", "this is a detailed message to the log message");
```

Subgraph (Call a Graph From the JS Operator)

The `instantiateGraph` function provides the possibility to call a graph from the JavaScript operator. You can provide parameters for the graph and handlers for output ports. `instantiateGraph` returns a graph object when graph is started or fail script execution.

The graph object has the following methods:

- `wait`: block script execution until subgraph execution is finished. `wait` returns the graph status
- `stop`: initiate subgraph termination. It returns without waiting for graph termination to be completed

```
$.addGenerator(gen)
// This example shows basic usage of a subgraph
// It calls a graph with a parameter, provides input to a port,
// checks a port's output and stops the graph
var graphName = "com.sap.demo.subgraph.call-and-wait.sub"
var subgraphInput = "subgraph-input "
var paramValue = "value1"
function gen(ctx) {
  // start subgraph
  var g
  try {
    g = $.instantiateGraph(graphName,
      {"param1": paramValue},
      // handle subgraph output port with name 'output'
      {"output":function(ctx,s){
        // check subgraph output
        if (s !== subgraphInput + paramValue) {
          $.fail("Unexpected subgraph output: " + s + " Expected: " + prefix
+paramValue)
        }
        // stop subgraph
        g.stop()
      }}
    )
  } catch(e) {
    $.fail(e.message)
    return
  }
  // write string in subgraph input port with name 'input'
  g.input(subgraphInput)
  // wait for graph execution is finished
```

```

status = g.wait()
// check graph status
if (status !== $.graphStatus.completed) {
  $.fail("Subgraph is failed")
} else {
  $.done()
}
}

```

2.10.9 JS Message Operator

This operator is an example on how to use the message type with the JavaScript operator.

Configuration Parameters

Parameter	Type	Description
codeLanguage	string	The programming language used in the snippet. Default: "javascript"
script	string	Mandatory. The inline script to be executed. If the script starts with <code>file://</code> , then the script file is executed. Default: ""

Input

Input	Type	Description
input	string	Any string that will trigger the counter output.

Output

Output	Type	Description
output	message	The message whose body contains an uppercase string.

For further information on customizing the JavaScript operator, follow the [JavaScript Operator \[page 340\]](#) documentation.

2.10.10 JS String Operator

This operator is an example on how to handle inputs and generate outputs.

Configuration Parameters

Parameter	Type	Description
codelanguage	string	The programming language used in the snippet. Default: "javascript"
script	string	Mandatory. The inline script to be executed. If the script starts with <code>file://</code> , then the script file is executed. Default: ""

Input

Input	Type	Description
input	string	Any string that will trigger the counter output.

Output

Output	Type	Description
output	string	The counter value.

For further information on customizing the JavaScript operator, follow the [JavaScript Operator \[page 340\]](#) documentation.

2.10.11 Message Generator

The Data Generator operator is an extension of the base operator `com.sap.system.jsengine`. It generates a message with random sample data every 500ms.

Configuration Parameters

None

Input

None

Output

Output	Type	Description
output	message	Generated message

2.10.12 Process Executor

The Process Executor operator starts a process and provides given contiguous streams to it. The operator finishes when the forked process terminates. A non-zero return code is propagated as an abnormal operator termination.

Configuration Parameters

Name	ID	Type	Description
Command Line	cmdLine	string	Mandatory. The command line to be executed.
Output stderr to trace	TraceStderr	bool	Offers the option to send the stderr output of the process also sent to the Modeler trace of the process. The errors are sent as INFO traces with a prepended [PROC STDERR]. Default: false

If the operator is used as a base operator to define an extended operator, there are two additional options, which are available in the operator editor:

Name	ID	Type	Description
code language	codeLanguage	string	If the executable is a script, this indicates the programming language used. Default: ""
script	script	string	This parameter is not shown in the list with the other configuration parameters, but has a separate tab offering the script editor. The script can be either defined as inline script or a file can be used. In the latter case, the operator configuration later shows "file://" as the script file to be executed. Default: ""

Input

Input	Type	Description
<code>stdin</code>	stream	The contiguous stream to be provided on standard input of the forked process.

Output

Output	Type	Description
<code>stdout</code>	stream	The contiguous stream arriving from standard output of the forked process.
<code>stderr</code>	stream	The contiguous stream arriving from standard error of the forked process.

2.10.13 Python 3 Operator

With the Python 3 Operator you can define a script that offers some convenience functions provided by the API object. For example, you can set up a callback that is called for every new data received in the "datain" port by writing `api.set_port_callback("datain", callback_name)`. Other conveniences provided by the API are described below.

An extantion of the Python3 Operator can be created directly in the Modeler by choosing the **Repository** tab.

i Note

This operator runs on Python 3.6.

i Note

Each callback registered in the script runs in a **different thread**. As the script developer, you should handle potential concurrency issues such as race conditions. You can, for instance, use primitives from the Python threading module. You can check this on Python.org's official documentation at <https://docs.python.org/3/library/threading.html> .

Configuration Parameters

Name	ID	Type	Description
script	script	string	Mandatory. The inline script to be executed. If the script starts with "file://", it specifies the path to the script file. Default: ""

Input

None.

Output

None.

Basic Examples

Count all incoming messages on the "input" port and write the count to the "output" port.

```
counter = 0
def on_input(data):
    global counter
    counter += 1
    api.send("output", counter)
api.set_port_callback("input", on_input)
```

Notice that the "output" port of an operator containing the above script should be of type `int64`, because the variable `counter` is of Python type `int`. See the section *Correspondence between Modeler types and Python types* for more information on how to choose your port types.

`api.set_port_callback` can also be used to wait for two input ports to have received data before calling the handler:

```
def on_input(data1, data2):
    api.send("output", data1 + data2)
api.set_port_callback(["input1", "input2"], on_input)
```

api.send(port, data)

```
Send `data` to outport named `port`.
Args:
    port (str): Name of output port.
    data (...): Objectr to be sent.
```

Port Callbacks

api.set_port_callback(ports, callback)

```
This method associates input `ports` to the `callback`. The `callback` is called
only when there are
messages available in all ports in `ports`. If this method is called multiple
times for the same port group,
the old `callback` is replaced by a new one. Different port groups cannot
overlap. The same callback
function cannot be reused in different port groups.
Args:
    ports (str|list[str]): input ports to be associated with the callback.
    `ports` can be a list of strings
                        with the name of each port to be associated or a
string if you want to associate the
                        callback to a single port.
    callback (func[...]): a callback function with the same number of arguments
as elements in `ports`.
                        The arguments are passed to `callback` in the same
order of their corresponding
                        ports in the `ports` list.
```

TIP: If you want to reuse the same callback function in multiple port groups, you can have a function that returns your function of interest:

```
def get_my_callback():
    def my_callback(data):
        # some code here
    return my_callback
api.set_port_callback("inport1", get_my_callback())
api.set_port_callback("inport2", get_my_callback())
```

This way, a new function (with different ID) is generated each time, but the callback behavior remains the same.

api.remove_port_callback(callback)

```
Deregister the `callback` function. If it does not exist, the method exits
quietly.
Args:
    callback: Callback function to be removed.
```

Generators

Generators are functions that are executed before the event processing loop starts. They are executed in the order they are added.

```
counter = 0
def gen():
    global counter
    for i in range(0, 3):
        for i in xrange(0, 3):
            api.send("output", counter)
            counter += 1
api.add_generator(gen)
api.add_generator(gen)
```

This example produces values 0,1,2,3,4,5 on the output port "output".

Timers

api.add_timer(period, callback)

Multiple distinct periodic callbacks can be added. If an already added callback is added again, the old period is replaced by the new `period`. Timers are not preemptive. Thus, the given interval provides only the lower bound of the interval at which the timer function is called. A zero `period` implies that the callback is called as fast as possible. If you want two callbacks with identical behavior to be run simultaneously then you need to create two different functions with identical body or create a factory function that defines an inner function and returns it. Each time the factory function is called a new function (with different ID) is returned, but with identical behavior. See the tip in the `set_port_callback` section to see an example of a factory function.

Args:

`callback (func)`: Callback function to be called every `milliseconds`.
`period (str)`: Period between calls of `callback`. It is passed as a string. For example, "-1.3h3m2us" means minus (1.3 hours 3 minutes and 2 microseconds), which is converted to: $-(1.3 \times 3600 + 3 \times 60 + 2 \times 10^{-6}) = -4860.000002$ seconds. Available suffixes are: h, m, s, ms, us, ns. They represent hours, minutes, seconds, milliseconds, microseconds, and nanoseconds, respectively. If the input uses multiple units, it needs to follow the same order presented in the previous sentence. That is, "2h3s" is allowed, but "3s2h" is not. Only strings "0", "+0", "-0" are allowed not to have a unit of time suffix (they are optional in these cases). Signs are optional and are allowed just in the beginning of the string.

Example:

```
counter = 0
def t1():
```

```

    global counter
    api.send("output", counter)
    counter += 1
    api.add_timer("1s", t1)

```

This example produces value 0,1,2, and so on on port "output" until graph shutdown.

api.remove_timer(callback)

```

Deregister timer callback.
Args:
    callback (func): Callback function to be removed.

```

api.update_timer(period, callback)

```

Update the period of an existing `callback`.
No error is thrown if the callback does not exist.
Args:
    callback (func): Callback function for which the period is updated. If
`callback` is not registered,
        nothing changes.
    period (str): String number representing a new period to overwrite the old
value.

```

Shutdown Handlers

Shutdown handlers are functions that are executed in the order they are added after the event processing loop execution.

```

counter = 0
def on_input(data):
    global counter
    counter += 1
api.set_port_callback("input", on_input)
def shutdown1():
    print ("shutdown1: %d" % counter)
def shutdown2():
    print ("shutdown2: %d" % counter)
api.add_shutdown_handler(shutdown1)
api.add_shutdown_handler(shutdown2) counter = 0
def on_input(data):
    global counter
    counter += 1
api.set_port_callback("input", on_input)
def shutdown1():
    print "shutdown1: %d" % counter
def shutdown2():
    print "shutdown2: %d" % counter
api.add_shutdown_handler(shutdown1)
api.add_shutdown_handler(shutdown2)

```

This example prints "shutdown1: 7" and "shutdown2: 7" if 7 values are provided on "input" port.

Configuration API

The API object provides the possibility to read configurations defined on the operator's editor view or added in the configuration panel in the graph's view. If you define a config named "foo", you can access it by calling `api.config.foo`. Configuration field names cannot contain spaces. The "script" and "codelanguage" config fields will not appear in `api.config` as they are mandatory configuration parameters and should not be used by the end user.

If you add a JSON object to a configuration field in the Modeler, it becomes a dictionary in Python.

Logging

To log messages, use `api.logger.info("some text")`, `api.logger.debug("")`, `api.logger.warn("")`, or `api.logger.error("")`.

Error Handling

You can raise any exception inside the script. As a result, the exception message is logged and the operator is terminated. If the exception is thrown inside a thread different than the script's main thread. In this case, you need to call `api.propagate_exception(e)`, where **e** is the exception object to handle it correctly.

i Note

Do not write to `stderr` because that could have unintended consequences. Also, do not use `api.logger.fatal` or `api.logger.critical`; otherwise the graph stops. Raise an exception or use `api.propagate_exception` if you want to stop the graph and log the error.

Halting Execution

If you want to halt the graph's execution with an error, follow the instructions from the **Error Handling** section. To stop the graph without an error, a **Python Operator** can send a signal to a **Graph Terminator** operator.

If it is necessary to stop the operator's execution (without stopping the graph as a whole), you can deregister all its callbacks. Notice that stopping a single operator can lead the data flow in the graph as a whole to stop.

i Note

Be aware that using the Python command `sys.exit(code)` causes only the current Python thread to exit and we do not recommend its use. We also do not advise using the Python command `os._exit(code)`, which causes the whole Python process to abort and the graph to stop with an error. If you want the whole graph to stop with an error, follow the instructions from the **Error Handling** section.

repo_root and subengine_root

You can access the `repo_root` and `subengine_root` path using `api.repo_root` and `api.subengine_root`, respectively.

graph_name, graph_handle, and group_id

You can access the `graph_name`, `graph_handle` and `group_id` values using `api.graph_name`, `api.graph_handle` and `api.group_id`, respectively.

- `graph_name`: Identifier of a saved graph. Example: `com.sap.dataGenerator`.
- `graph_handle`: Unique identifier of the graph instance. It is referred to as Runtime Handle in the modeler user interface.
- `group_id`: Unique identifier of the group where the operator is running.

multiplicity and multiplicity_index

Two variables can be accessed to obtain multiplicity-related information `api.multiplicity` and `api.multiplicity_index`:

- `multiplicity`: The multiplicity of the operator's group.
- `multiplicity_index`: An integer in the range `[0, multiplicity]` that can be used to distinguish the multiple instances of this operator.

Port Names

You can get the name of the input and output ports for the current operator instance by calling the methods `api.get_inport_names()` and `api.get_outport_names()`, respectively.

You can check which inports or outports are connected by using the `api.is_inport_connected` and `api.is_outport_connected` dictionaries, which have the name of the port as key and a boolean indicating whether it is connected as value.

Example:

```
if api.is_outport_connected["outport1"]:  
    r = do_some_expensive_calculation()  
    api.send("outport1", r)
```

Message Type

You can access the message type in the API object as `api.Message`. The message type can be built such as the following: `api.Message(body, attributes)`, where `body` can be any object, and `attributes` should be a

dictionary of str to object, or None. The body argument is mandatory while attributes is optional and defaults to None. The body and attributes of a message object `msg` can be accessed as `msg.body` and `msg.attributes`, respectively.

Correspondence Between Modeler Types and Python Types

The Modeler types are the ones that are allowed in the operator ports. For example, if you have an input port of type `blob` then the Python object that you are going to receive as argument to your port callback is of type `bytes`. If the output port of your operator has type `string`, you should send a Python object of type `str` in its output port.

Modeler	Python3
string	str
blob	bytes
int64	int
uint64	int
float64	float
byte	int
message	api.Message

You can also set a Python operator port type to `python36`. In this case, you are able only to connect this port to other Python operators. This can be useful if you need to send Python specific data types to other Python operators. Note that connections between `python36` ports that cross group boundaries are not allowed, and leads to runtime error.

FAQ

Where should I place initialization code for my operator?

The script is executed just once. The callback functions defined in the script can be executed multiple times, but the commands from the script's outermost scope will be executed just once. This implies that one can simply place initialization code in the body of the script. Below we show an example:

```
# Hypothetical script for Python2Operator
# In the operator's initialization, we might want to setup a connection
# with a database, for example (`setup_connection` is an hypothetical function):
db = setup_connection(api.config.host, api.config.port)
def my_callback_func(data):
    global db
    db.write(data) # hypothetical call
api.set_port_callback("input", my_callback_func)
```

Alternatively, you can also place initialization code in a generator callback that is registered with `api.add_generator(func)`.

2.10.14 R Client 3

The R Client 3 operator runs R code defined by the user in the Rserve.

You can specify the code you want to run it by writing an R script and adding callbacks by using the predefined API object.

For example, you can set a callback function, `myFunc`, which is called when you receive new data in the port `input1`, and then send data on port `output2` by writing `api$setPortCallback(c("input1"), c("output2"), "myFunc")`.

The functionalities offered by the API object are the following:

- `api$setPortCallback(inports, outports, functionName)`
- `api$addTimer(period, outports, functionName)`
- `api$setSwitchCallback(inports, outports, functionName)`
- `api$addShutdownHandler(functionName)`

Each of those functionalities will be discussed in later sections.

Your script will run during the initialization of the R Client operator. For code that should run after the initialization, you can add and define callbacks in the script, by using methods from the API object such as: `setPortCallback`, `addTimer`, `setSwitchCallback`, and `addShutdownHandler`.

! Restriction

TLS connection with Rserve is not supported.

This operator is supported on R 3.3.3 and 3.5.1.

Configuration Parameters

Name	ID	Type	Description
Script	script	string	Mandatory. Inline script to be executed, or if the entered text starts with "file://" then it will be interpreted as the name of the script file to be executed. Default: "file://script.R"

Name	ID	Type	Description
Switch Start Off	switchStartOff	boolean	In case, a switch callback exists, then the other callbacks will be disabled from the graph's start if this field is true. Default: "true"
Switch Affects Timer Callbacks	switchAffectTimerCallbacks	boolean	In case a switch callback exists, then the timer callback will be affected by the state of the switch if this field is true (the port callbacks are always affected). Default: "true"
Debug Mode	debugMode	boolean	Set to true to get better error messages, but the performance may be affected when enabled. Default: "true"
Table Name by Outport	tableNameByOutport	array	A map from outport name to the table name that will be used when a data frame is sent on the given outport. If a data frame is sent on an outport that is not defined in this configuration, then the resulting message.table will have the table name filled as 'rclient' + <outportName>. Default: []

Connection Parameters

i Note

Use `Connection Management` for defining connections over the manual option.

Name	ID	Type	Description
Connection	connection	object	Holds information about the services' connections. Use connection type "Manual" only when using a Connection ID is not possible.
Configuration Type	configurationType	string	The type of connection information that will be used. Manual (user input) or Configuration Manager (retrieved by the Connection Management service). i Note It is recommended to choose <i>Configuration Manager</i> as the configuration type and select an existing <i>Connection ID</i>. The <i>Manual</i> configuration type must be used only for testing purposes. Default: "Manual"
Connection ID	connectionID	string	The ID of the connection information to retrieve from the Connection Management service. The connection information needs to be of type RSERVICE. If the Host field in your connection information is left empty, then a new Rserve will be launched automatically and the rest of the credentials will be ignored. This field is visible when <i>Configuration Type</i> is set to Configuration Manager. Default: "true"
Connection Properties	connectionProperties	object	This field is visible when <i>Configuration Type</i> is set to Manual .

Name	ID	Type	Description
Host	host	string	Address of a running Rserve. If left empty, an Rserve will be launched and a port will be found automatically. Default: ""
Port	port	int	Port of the Rserve. If the Host is left empty, this field will be ignored. Default: 0
User	user	string	Username for authenticating in Rserve if required. If the Host field is empty, the supplied username will be ignored. Default: ""
Password	password	string	Password for authenticating in Rserve if required. If the Host field is empty, the supplied password will be ignored. Default: ""

Input

None.

Output

None.

Allowed Types and (Data Pipeline)/R Type Equivalences

Allowed Types in In/Out Ports

- string

- blob (byte)
- float64
- int64
- uint64
- arrays of the types above (string, float64, and so on)
- byte
- message
- any

SAP Data Intelligence Modeler/R Type Equivalences

SAP Data Intelligence Modeler	R Type Equivalences
string	character
blob	raw
float64	double/numeric (64 bits)
int64	integer (signed 32 bits)
uint64	integer (signed 32 bits)
slice or array	vector
byte	integer (signed 32 bits)
message*	list(Body,Attributes,Encoding)
message.table**	data frame

* message with field encoding different from 'table'. Example of message creation in R: `m <- list(Body=11, Attributes=list(field1="foo"), Encoding="bar")`.

** message with field encoding equal to 'table'. In R it is represented as a data frame.

The table above only covers types that can be directly used in the operators' ports. More information related to types, which can be used in inner fields of the message data type, can be found in the Go/R Type Equivalences and Conversions section.

i Note

The R Client operator does not warn you or show error if data loss happens during numeric conversions between Go and R. Some uint64 values, for instance, cannot be represented neither as a double nor as an integer in R. R Client does not specify what the result will be in those cases.

i Note

Some R types are not supported during Modeler and R conversions, such as, complex numbers, matrices, and factors. Sending them on the R Client output port may at times not cause error, but the result is not defined.

API Object Methods

api\$setPortCallback(inports, outports, functionName):

This method registers a callback named `functionName` that will be called when all ports in `inports` have data waiting to be read. The `outports` argument specify the name of zero or more outports to which the returned values from `functionName` should be sent. The values returned by the function are expected to be returned with the list data type in R. So the last line of the function should contain something like: `list(output1=value1, output2=value2)`, where `output1` and `output2` are the outport names that were also specified in the `outports` argument of the method `api$setPortCallback`. If no outport was specified in `api$setPortCallback`, then the last line of the callback function can be anything because its return value will be ignored anyway.

If you specify outports for the callback, but in some calls you want to skip sending to the outport, then you can return `NULL` as the function return. For example, this is useful when you accumulate some data in a global variable for some calls and then in a later callback you want to send the accumulated information. So in the first call you return `NULL` and in the later call you return `list(output=accumulated)`. In this manner, nothing is sent to the outport in the first call, only in the later one.

❖ Example

```
acc = 0
on_input12add <- function(data1, data2) {
  acc <- acc + data1 + data2
}
on_input12mult <- function(data1, data2) {
  acc <- acc * data1 * data2
  list(output1=acc)
}
on_input3pow <- function(data3) {
  list(output1=acc^data3, output2=data3^data3)
}
api$setPortCallback(c("input1", "input2"), c(), "on_input12add")
api$setPortCallback(c("input1", "input2"), c("output1"), "on_input12mult")
api$setPortCallback(c("input3"), c("output1", "output2"), "on_input12pow")
```

The script above has three callbacks: "on_input12add", "on_input12mult", and "on_input3exp". Both "on_input12add", "on_input12mult" functions wait for data to be available in the port group ("input1", "input2"). When the data becomes available, the data from "input1" is passed to the first argument of the callbacks and the data from "input2" is passed to the second argument of the callback. When more than one callback is associated with a port group (as in this case), the callbacks are executed sequentially in the order that the callbacks were added in the script.

The callback "on_input3pow" is associated with a port group that is disjointed from the previous one. In this case, the port group consists of just one input port "input3". Notice that the callback should be the name of a function variable that is in scope. Inport names should coincide with the name of an existing inport, and the number of ports in a port group needs to equal the number of arguments of the callback associated with this group. It is important to notice that all port groups cannot overlap. For example, you cannot add a callback associated with ports ("in1", "in2") and another associated with port ("in1") or ports ("in2", "in3"). However, you are allowed to add more than one callback to the same port group. The union of all port groups does not need to cover all existing input ports of the operator. That means that some input ports can be left unused.

The function `on_input12add` will add `acc` (a global variable) with `data1` and `data2`, and assign the result to `acc` again. Because the line `api$setPortCallback(c("input1", "input2"), c(), "on_input12add")` didn't specify output ports, the return value of `on_input12add` will be ignored. This callback's objective is just to update the `acc` global variable.

On the other hand, the returned value of `on_input12mult` is not ignored. Instead, the value on the returned list with key equal `output1` will be fed into the output port of the same name. This intent was signaled by specifying "output1" in the list of outputs in the `setPortCallback` method. When we specify that, we always need to return an R list with a key for each output name in the last line of the callback.

`api$addTimer(period, outputs, functionName):`

This method registers a callback named `functionName` that will be repeatedly called on every `period` units of time. The `period` is passed as a string. For example, "1.3h3m2us" means (1.3 hours 3 minutes and 2 microseconds).

Available suffixes are: h, m, s, ms, us, ns. They represent hours, minutes, seconds, milliseconds, microseconds, and nanoseconds, respectively. Only the strings "0", "+0", and "-0" are allowed to not have a unit of time suffix (they are optional in these cases). Signs are optional and are allowed just in the beginning of the string.

The `outputs` argument specify the name of zero or more outputs to which the returned values from `functionName` should be sent. The values returned by the function are expected to be returned with the list data type in R. So the last line of the function should contain something like: `list(output1=value1, output2=value2)`, where `output1` and `output2` are the output names that were also specified in the `outputs` argument of the method `api$addTimer`. If no output was specified in `api$addTimer`, then the last line of the callback function can be anything because its return value will be ignored.

If you specify outputs for the callback, but in some calls you want to skip sending to the output, then you can return NULL as the function return. For example, this is useful when you accumulate some data in a global variable for some calls and then in a later callback you want to send the accumulated information. So in the first call you return NULL and in the later call you return `list(output=accumulated)`. In this manner, nothing is sent to the output in the first call, only in the later one.

❖ Example

```
counter <- 0
t1 <- function() {
  counter <- counter + 1
  list(output=counter)
}
api$addTimer("1s", c("output"), "t1")
```

This example produces value 1, 2, 3, and so on on every second on output "output" until the graph shutdown.

`api$setSwitchCallback(inports, outputs, functionName):`

This method adds a callback that is able to control the switch variable. When the switch is 'on', all other callbacks become enabled. When the switch is 'off', all other callbacks become disabled and no data will be consumed in their ports.

The callback `functionName` added with `setSwitchCallback` will always need to return a boolean indicating what value the switch should assume. For example, suppose that you added a switch callback as: `api$setSwitchCallback(inports=c("inputX", "inputY"), outputs=c("outputA", "outputB"), "foo")`. Then the function `foo` should look like this:

```
foo <- function(dataX, dataY) {
  out1 <- dataX + dataY
  out2 <- dataX * dataY
  if (dataX %% dataY == 0) {
    position <- FALSE
  } else {
```

```

    position <- TRUE
  }
  list(switchPosition=position, outputA=out1, outputB=out2)
}

```

The last line of switch callback always needs to have the keyword `switchPosition` in the returned list. Notice that the boolean returned will not be sent in any output unless you include a port called 'switchPosition' of type byte in the operator and also specify this port in the `outputs` parameter of the `api$setSwitchCallback` method. In case you specify, then the boolean will be converted to a byte and sent in the 'switchPosition' port, in addition to being used internally to set the value of the switch.

There are two operator configuration parameters related to the switch callback: `switchStartOff` and `switchAffectTimerCallbacks`. If `switchStartOff` is set to true, then the switch will start in the off position when the graph is started. The `switchAffectTimerCallbacks` configuration indicates whether the switch should affect (block or unblock) the timer callbacks too. On the other hand, the port callbacks are always affected, and the shutdown, and switch callbacks are never affected by the switch position. That is the shutdown, and switch callbacks are always enabled (unblocked).

Unlike `api$setPortCallback`, `api$setTimer`, and `api.addShutdownHandler`, `api$setSwitchCallback` can set only one callback. If you use the method more than once, the first callback will be replaced by the new one.

api\$addShutdownHandler(functionName):

This method will register the callback `functionName` to be called upon the graph's shutdown. If more than one callback is added, they will be called sequentially in the order that they were added in the script.

R Client will try to execute all shutdown callbacks even in when the operator is being stopped due to an error. Also, if one shutdown callback fails the next ones will still be executed. R Client only will not try to execute the shutdown callbacks when Rserve is dead or if an error happens before the shutdown callbacks registration is completed.

❖ Example

```

counter <- 0
on_input <- function(data) {
  counter <- counter + 1
}
api$setPortCallback(c("input"), c(), "on_input")
shutdown1 <- function() {
  cat("shutdown1: ", counter)
}
shutdown2 <- function() {
  cat("shutdown2: ", counter)
}
api$addShutdownCallback("shutdown1")
api$addShutdownCallback("shutdown2")

```

The above example prints "shutdown1: 7" and "shutdown2: 7" if 7 values have been provided on input port before the graph stops.

Error Messages

If you leave the Host configuration empty, Rserve will be launched inside the graph's container and **R Client** will be able to save the STDERR stream of the Rserve process. In this case, in case of failure, the accumulated

STDERR stream will be concatenated to the error message shown in the UI. Notice that having Debug Mode equal to true may change what Rserve prints in STDERR. Sometimes it may be useful to set Debug Mode to false because it may cause STDERR to show new information.

When using an external Rserve by filling the Host configuration, the R Client will not be able to save Rserve's STDERR. In this case, the user can only resort to enabling Debug Mode for improving the error messages.

Go/R Type Equivalences and Conversions

This section is useful if you are connecting R Client with a script Go operator.

Message Type

The Go message type can be represented in R either as a generic list object or as a `data.frame`.

A non-message.table in Go will be represented as a list in R with the following fields: Body, Attributes, Encoding. The Body field can hold any R object that can be serialized to Go (see SAP Data Intelligence/R Type Equivalences section). The Attributes should be an R list mapping field names to R objects. The Encoding should be a character vector.

- **Message Table**

A message.table in Go has to follow a certain format and it is automatically converted to a `data.frame` when sent to R Client. A `data.frame` is also converted to a message.table when received from R Client. If the encoding field of the message is different than "table", then the equivalent type in R will be a list with the fields Body, Attributes, and Encoding.

In a message.table we assume the type of each column is constant. In the next two subsections we show how the data in the columns of a message.table are converted to their R equivalent and how the data in the columns of an R data.frame are converted to their Go equivalent. Currently, only the following R types for data.frames columns are supported when it is send to the R Client output: `logical`, `character`, `integer`, and `double`.

- **Go message.table to R data.frame**

When a message.table is sent to R, the `class` field on the column metadata of the message header is ignored. The conversion is solely based on the Go type of the data on the table cells. The table below shows the mapping of the type of the columns of a message.table to the ones of an R data.frame.

Go	R
bool	logical
string	character
float32 or float64	double (64 bits)
int*	integer (signed 32 bits)

* `int` refers to the following Go types: `int8`, `uint8`, `int16`, `uint16`, `int32`, `uint32`, `int64`, `uint64`, `int`, and `uint`.

❖ Example

```
map[string]interface(){
  "Body": []interface(){
    []interface(){ "alk", uint64(3), true, 1.3},
    []interface(){ "", int8(-2), false, math.Inf(-1)},
  },
  "Attributes": map[string]interface(){
    "table": map[string]interface(){
      "version": int16(1),
      "columns": []interface(){
        map[string]interface(){ "name": "stringCol",
        map[string]interface(){ "name": "intCol",
        map[string]interface(){ "name": "boolCol",
        map[string]interface(){ "name": "doubleCol",
      },
    },
  },
  "Encoding": "table",
}
```

is converted to `data.frame(stringCol=c("alk", ""), intCol=c(3L, -2L), boolCol=c(TRUE, FALSE), doubleCol=c(1.3, -Inf), stringsAsFactors=FALSE)`.

- **R data.frame to Go message.table**

When an R `data.frame` is sent to Go, it will be converted to a `message.table` with the class field automatically filled for each column metadata and the data on its body will be converted according to the mapping below:

R	Go	class
logical	bool	bool
character	string	string
double (64 bits)	float64	float
integer (signed 32 bits)	int32	integer

The table name field of the resulting `message.table` will be either 'rclient' + <outputName> or the name specified in the configuration field Table Name By Output. We also set all columns as nullable (go to [Table Messages](#) to see definitions for this field).

❖ Example

`data.frame(stringCol=c("alk", ""), intCol=c(3L, -2L), boolCol=c(TRUE, FALSE), doubleCol=c(1.3, -Inf), stringsAsFactors=FALSE)` is converted to:

```
map[string]interface(){
  "Body": []interface(){
    []interface(){ "alk", int32(3), true, 1.3}, []interface(){ "",
    int32(-2), false, math.Inf(-1)},
  },
  "Attributes": map[string]interface(){
    "name": "rclientoutput"
    "table": map[string]interface(){
```

```

        "version": int64(1),
        "columns": []interface{}{
            map[string]interface{}{"name": "stringCol", "class":
"string", "nullable": true},
            map[string]interface{}{"name": "intCol", "class":
"integer", "nullable": true},
            map[string]interface{}{"name": "boolCol", "class": "bool",
"nullable": true},
            map[string]interface{}{"name": "doubleCol", "class":
"float", "nullable": true},
        },
    },
    "Encoding": "table",
}

```

The example above assumes the `Table Name By Outport` field was not specified for the used `outport` and that the used `outport` was named `output`. Observe that in the `data.frame` creation in R we specified `stringsAsFactors=FALSE`. This is necessary because by default data frame creation in R converts strings to factors and we currently do not support factors in conversions to `message.table`.

Non-Table-Message Go to R Conversions Examples

- `"abcd" → "abcd"`
- `[]string{"al", "alkj"} → c("al", "alkj")`
- `[] []string{{"al", "alkj"}, {"al2", "alkj2"}} → list(c("al", "alkj"), c("al2", "alkj2"))`
- `int64(1) → 1L`
- `1.0 → 1.0`
- `[] []float64{{1, 2}, {3,4}} → list(c(1, 2), c(3, 4))`
- `[]byte{1, 2, 3} → as.raw(c(1,2,3))`
- `map[string]interface{}{"Body": 3, "Attributes": nil, "Encoding": "foo"} → list(Body=3, Attributes=list(), Encoding="foo")`
- `nil → NA`

Non-Data-Frame R to Go Conversions

This section is divided in two subsections. The first shows how conversions are done when the target Go type is known. The second covers the case when the target Go type is not known (when output port is of type any or when data is inside the body of a `message`).

- **Target Go Type is Known**
In this context the output port of the R Client indicates the target Go type for the R object be converted to.

♣ Example

R → Go (known target type)

- **outport type uint64:**
 - `1L → uint64(1)`
 - `2.7 → uint64(2)`
- **outport type int64:**
 - `1L → int64(-1)`
 - `2.7 → int64(-2)`

- output type `float64`:
 - `1L → float64(1.0)`
 - `2.7 → float64(2.7)`
- output type `[]uint64`:
 - `1.3 → []uint64{1}`
 - `c(1.3) → []uint64{1}`
 - `c(1.3, 2.7) → []uint64{1, 2}`

- **Target Go Type is Unknown**

The target Go type might not be known when the port type remains any even after the port types are propagated or when the data is inside the body of a message. In those cases all numeric and vector of numeric types in R will be converted to `float64` and slice of `float64` in Go, respectively (except in `data.frame` to `message.table` conversions where double is converted to `float64`, but integer is converted to `int32`). Singleton vectors in R (`c(1)`) will be converted to scalars in Go (`1.0`). On the other hand, singleton lists (`list(1)`) will be converted to a singleton slice in Go (`[]float64{1.0}`).

❖ Example

R → Go (Unknown Target Type)

- `1L → float64(1)`
- `2.7 → float64(2.7)`
- `c(1.3) → float64(1.3)`
- `list(1) → []float64{1}`
- `c(1.3, 2.7) → []float64{1.3, 2.7}`
- `list(1.3, 3L) → []float64{1.3, 3.0}`
- `list(Body=3, Attributes=list()) → map[string]interface{}{"Body": 3.0, "Attributes": nil}`
- `list(c(1,2,3), c(4,5,6)) → [][]float64{{1, 2, 3}, {4, 5, 6}}`
- `list(c(1,2,3), c("alkj")) → []interface{}{[]float64{1, 2, 3}, "alkj"}`
- `c("one","two","three") → []string{"one", "two", "three"}`
- `list("one","two","three") → []string{"one", "two", "three"}`
- `NULL → nil`
- `NA → nil`
- `list(1, "lakj", c(2, 3, 4), c("alkj", "qoiu")) → []interface{}{1.0, "lakj", []float64{2, 3, 4}, []string{"alkj", "qoiu"}}`
- `list(list(1L, "abc", 2.0), list(), list("def", 3)) → [][]interface{}{{1.0, "abc", 2.0}, {}, {"def", 3.0}}`
- `list(c(1), c(1, 2), c(3, 4)) → []interface{}{1.0, []float64{1, 2}, []float64{3, 4}}`

In the last example, note that `c(1)` is converted to `1.0` and not to `[]float64{1.0}`. This is because singleton vectors in R are converted to scalars in Go.

Docker Tags

The default R Client operator already comes with three tags: "rserve", "rjsonlite", and "rmsgpack". Those are needed to select the right Docker image for the container where the operator's group will run. Those tags are only relevant when the user leaves the [Host](#) configuration field empty and thus causing a new Rserve process to be launched inside the operator's group container. If you are going to connect to an already existing Rserve by specifying the [Host](#) configuration, then you don't need those tags. You can remove those tags by creating a new operator that extends the R Client operator. Removing the tags is not mandatory when connecting to an external Rserve, but it is recommended because it will lower the requirements for the Docker image selected for the operator's group. If your external Rserve doesn't already have the `jsonlite` and `msgpack` packages installed then they will be installed in the first time you run the R Client operator connected to that [Host](#). The graph may take a while to start due to the installation time of those packages in your external Rserve.

2.10.15 Stdin/out Port Muxer

The stdmuxer maps multiple input ports to stdin of a process and multiplexes the output of stdout to multiple output ports. The base type of all ports is stream.

Muxer Protocol

The communication between the SAP Vora Pipeline Engine and the forked process requires a small protocol for multiplexing over stdin and stdout:

```
| 1byte | 8byte |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| port | size of data (uint64 big endian) | data |
+-----+-----+-----+-----+-----+-----+-----+-----+-----+...
```

port is the ID of an in or out port. The port IDs are assigned dynamically to all bound ports and passed into the process via environment variables with the prefixes `VFLOW_MUX_INPORT_` and `VFLOW_MUX_OUTPORT_`. The suffix is the port name in uppercase. For example, if you have an in port named `myPort`, the variable name is `VFLOW_MUX_INPORT_MYPORT`. If the variable of a port is not available in the process, it was not bound. Port IDs are only unique for a certain port direction; that is, 0 can be the ID of both in and out ports.

Configuration Parameters

Parameter	Type	Description
CmdLine	string	The command line of the process to be forked. Default: ""

Parameter	Type	Description
Env	string	Additional environment variables to be provided to the process. Default: ""
InheritEnv	string	If set to true, the forked process inherits the environment. Default: "false"
TraceStderr	string	If set to true, the standard error of the forked process is traced to the parent process with severity ERROR. Default: "false"

Input

None

Output

None

2.10.16 Streaming Analytics Projects

This operator represents a streaming analytics project. The operator is based on streaming lite, which is a compact version of SAP HANA streaming analytics. As such, each instance of the operator starts an instance of the streaming lite server.

This streaming lite instance does not depend on SAP HANA in any way. If you wish to read from or write data to an SAP HANA database, you can do so by adding the Database Input or Output adapter into your streaming analytics project.

Streaming analytics is ideally suited for situations where data arrives as events happen, and where there is value in collecting, understanding, and acting on this data right away. Some examples of data sources that produce streams of events in real time include:

- Sensors
- Smart devices
- Web sites (click streams)
- IT systems (logs)

- Financial markets (prices)
- Social media

Data flows into streaming projects from various sources, such as adapters or operator ports that connect the sources to the streaming lite server. The streaming projects contain business logic, which they apply to the incoming data, typically in the form of continuous queries and rules. These streaming projects are entirely event-driven, turning the raw input streams into one or more derived streams that can be captured in a database, sent as alerts, posted to downstream applications, or streamed to live dashboards.

At its most basic level, a project consists of streams, windows, and adapters.

- Adapters connect a stream or window to a data source or destination.
- A stream processes incoming events without retaining and storing data, and produces output events according to an applied continuous query.
- A window receives data but can also retain and store data. Incoming events can add, update, or delete rows in the window's table.

You can also define one or more input and output operator ports in your project to receive and send data.

Configuration Parameters

Title	ID	Type	Description
Script	script	string	Contains Continuous Computation Language (CCL), which specifies the functionality of the project. CCL is the primary event processing language of streaming analytics and is based on Structured Query Language (SQL). Default: "file://script.ccl"

Title	ID	Type	Description
CCX	ccx	string	Already compiled CCL (CCX) that is used to start the project. • If this value is empty, the script property must have a value specified as that is then compiled into CCX and used to start the project. • If a value is provided, the script property value is ignored and the compiled CCX is used to start the project. Default: ""
CCR	ccr	string	Optional runtime project configuration (CCR) file that governs specific runtime properties of a project. Default: ""
Command Port	command port	string	If specified, this port number is used by the project to listen for commands. If the value is 0 or left empty, a random free port number is used instead. Default: ""

Title	ID	Type	Description
Input Format	input format	string	The format used to parse rows received on input ports. • CSV: Incoming records need to be in comma-separated values (CSV) format and follow the RFC4180 standard. Headers are not supported. Multiple records can be batched together. • JSON: Input can be only a JSON array. Each of its elements can be either: ◦ A JSON array representing a data row where its elements are in the same order as the schema of the target input stream or window in the project CCL. If the input has opcodes parameter is set to true, the first array element must contain an opcode character. ◦ A JSON object whose keys are the column names in the schema of the target input stream or window in the project CCL. Spare keys are ignored. If the input has opcodes parameter is set to true, the JSON object must contain an <code>__opcode</code> field with an opcode character value specified.

Title	ID	Type	Description
			Default: "CSV"
Output Format	output format	string	The format used to build rows that are sent on output ports. • CSV: Outgoing records are in CSV format and follow the RFC4180 standard. Headers are not included and multiple records are batched together according to the values specified in the output batch size and output batch timeout parameters. • JSON: Output is a JSON array that forms a batch of records. The size is based on the values specified in the output batch size and output batch timeout parameters. Each array element is either a JSON object or an array depending on the value of the JSON record type parameter. Default: "CSV"

Title	ID	Type	Description
JSON Record Type	json record type	string	The type of JSON record to output for each outgoing data row. - Object: Each outgoing record is a JSON object whose keys are the column names in the schema of the source output stream or window in the project CCL. If the add opcodes to output parameter is set to true, the JSON object will contain an <code><__opcode></code> field with an opcode character value specified. - Array: Each outgoing record is a JSON array representing a data row. Its elements are in the same order as the schema of the source output stream or window in the project CCL. If the add opcodes to output parameter is set to true, the first array element contains an opcode character. Default: "Object"

Title	ID	Type	Description
Input Has Opcodes	input has upcodes	boolean	Specifies whether incoming data rows contain an opcode character. If set to <code>false</code>, the insert opcode value is used. If set to <code>true</code>, each incoming data row will include one of the following opcodes: • <code>i</code> (insert): Insert the record. Throw an error if a record with the same key already exists. • <code>u</code> (update): Update an existing record. Throw an error if a record with the specified key does not exist. • <code>d</code> (delete): Delete a record. Throw an error if a record with the specified key does not exist. • <code>p</code> (upsert): Insert the record if the key does not exist, update the record otherwise. • <code>s</code> (safedelete): Delete the record if it exists, do nothing if it doesn't exist. Default: "true"
Add Opcodes to Output	add opcodes to output	boolean	Specifies whether the outgoing data rows contain an opcode character. Default: "true"
Output Batch Size	output batch size	string	Specifies how many outgoing rows, per output port, should be included in a batch. If the value is empty or set to less than 2 rows, each outgoing row is sent individually. Default: ""

Title	ID	Type	Description
Output Batch Timeout (ms)	output batch timeout	string	Specifies how many milliseconds to collect output rows for a batch before sending out the current batch and starting a new one. If the value is left empty or is less than 10 milliseconds, the default timeout of 10000 milliseconds (10 seconds) is used. That is, the shortest duration you can specify is 10 milliseconds. Default: ""
Stats Frequency (sec)	stats frequency	string	Specifies how often, in seconds, to print out an INFO level message with message flow statistics. For example, if set to 10 seconds, an INFO level message will print out every 10 seconds. Default: ""

Title	ID	Type	Description
Hana Services	hana services	array	An array of objects containing connection parameters for SAP HANA databases that can be used as data services. To be able to access the data services from the project CCL, the name of each object needs to match the corresponding service property in the project CCL. All array elements can be either: • Manual: All required SAP HANA connection properties need to be specified in the properties editor. • Configuration Manager: Existing connections that are type <code>HANA_DB</code> and defined in Connection Management can be selected from a drop-down list. Default: ""

Title	ID	Type	Description
ODBC Services	odbc service	array	An array of objects containing connection parameters for generic ODBC data sources such as databases. They can be used as ODBC data services in the project CCL. To be able to access them from the project CCL, the name of each object needs to match the corresponding service property in the project CCL. Each object contains reference to an ODBC DSN that needs to exist in the runtime environment. You can use <code>com.sap.streaming Dockerfile</code> to generate such DSN entries, as shown in an example in the bottom section of the Dockerfile. Default: ""
Services File	services file	string	The location and name of the file generated by the operator, which contains all the data service entries available for use in the project CCL. Specifying this value may be useful for debugging issues with data service connectivity. If the value is left empty, the temporary file is used instead. Default: ""
Log4j Properties	log4j properties	string	Specifies the log4j tracing properties that are used by the operator. By default, it includes output to console as well as to the logs/Flow-Agent.log file at INFO trace level. You can change these settings individually. Default: ""

Input

- Any input port names must match an input stream or window in the project CCL. The operator publishes any incoming data rows to that input stream or window. Only string and message type ports are supported.
- None.

Output

Output	Type	Description
outError	string	If mapped, the operator errors are sent to this port and the graph continues running. If not mapped, the graph terminates with the operator error.
committedMessage	message	Notifies about commits of input messages. If mapped, for any input message that has a value specified for the <code>message.commit.token</code> header attribute the operator will emit an output message with an empty body and all header attributes from the original input message preserved. Before emitting this output message the operator will publish and commit all input rows contained in the corresponding input message.

Any other output port names must match an output stream or window in the project CCL. The operator subscribes to that output stream or window for any outgoing data rows. Only string and message type ports are supported.

Functional Restrictions

The Streaming Analytics operator is based on streaming lite, which is a compact version of SAP HANA streaming analytics. The operator supports the following adapters:

- All adapters built using the adapter toolkit. However, you can use only them in managed mode, which means the adapter starts and stops with the streaming analytics project.
- The Streaming Web Output adapter is also supported.
- Database Input and Output adapters.

Do not use the binaries or SDKs that come with the streaming install image to communicate with and send data to and from the streaming project. Instead, use the input and output ports on the operator, as well as any of the supported adapters.

There are additional functionalities and components that the operator does not support. These are based on the restrictions for streaming lite. For complete details, see the [Streaming Lite](#) section in the SAP HANA Streaming Analytics: Developer Guide. Keep these in mind as you use the Streaming Analytics operator and graphs.

For more information on building a streaming analytics project, see the [SAP HANA Streaming Analytics: Developer Guide](#) and [SAP HANA Streaming Analytics: CCL Reference](#).

2.11 SAP HANA

Related Information

[Initialize HANA Table \[page 388\]](#)

[Read HANA Table \[page 392\]](#)

[Run HANA SQL \[page 397\]](#)

[Write to HANA Table \[page 399\]](#)

2.11.1 Initialize HANA Table

The Initialize HANA Table operator initializes one or more tables on an SAP HANA database.

Configuration Parameters

Name	ID	Type	Description
Connection	connection	object	Mandatory. The object containing the connection parameters to a HANA instance. Default: {}

Name	ID	Type	Description
Configuration Mode	configMode	string	How the operator obtains the values for the table name, column, specification, and primary key. Accepted values: • Static (from configuration): The operator does not expect any input. It initializes a table based on the configuration properties <code>Table name</code>, <code>Table columns</code>, and <code>Primary key</code>. It initializes the table as soon as it starts running, writes the relevant output, and becomes idle. • Dynamic (from input): The operator uses each input message to determine the name, columns, and primary key of the target table. These are obtained from the standard attributes in a <code>Table Message</code>. Default: " "
Table Name	tableName	string	The name of the table that is initialized in <code>Static</code> mode. Default: ""
Table Columns	tableColumns	array	The list of columns that are used, in <code>Static</code> mode, if the table needs to be created. This is used only when the table does not yet exist or <code>Initialization mode</code> is set to either <code>Drop</code> option.

Name	ID	Type	Description
Primary Key	primaryKey	array	A list of column names to be used as a primary key in <code>Static</code> mode. Default: []
Table Type	tableType	string	The type of table that is created. Accepted values: • Row • Column Default: Row
Initialization Mode	initMode	string	How the operator should initialize the table(s). If the table does not yet exist, the operator only creates it. If the table already exists: • Create: it performs no action. • Truncate: it empties the table using a <code>TRUNCATE</code> statement. • Delete: it empties the table using a <code>DELETE</code> statement. • Drop (Restrict): it drops and re-creates the table using a <code>DROP (. . .) RESTRICT</code> statement. • Drop (Cascade): it drops and re-creates the table using a <code>DROP (. . .) CASCADE</code> statement.

Name	ID	Type	Description
Connection timeout	connectionTimeout	string	The maximum amount of time to wait for the server. If at any time the operator does not hear from the server within this period of time, the connection is closed. If set to zero, the operator waits for the server indefinitely. Default: "50s"

Input

Input	Type	Description
input	message.table	A Table Message that specifies which table is initialized. The operator uses the following values from the input: - Name (<code>Attributes.table.name</code>) - Columns (<code>Attributes.table.columns</code>) - Primary Key (<code>Attributes.table.primaryKey</code>)

Output

Output	Type	Description
success	message.table	A Table Message describing the table that was successfully initialized. In <code>Dynamic</code> mode, this output also includes any additional attributes that were present in the input, as well as the original message body.

Output	Type	Description
error	message.error	An Error Message, in case an error was raised during an operation.

2.11.2 Read HANA Table

The Read HANA Table operator reads data from a table in SAP HANA.

Configuration Parameters

Name	ID	Type	Description
Connection	connection	object	Mandatory. The object containing the connection parameters to a HANA instance. Default: {}

Name	ID	Type	Description
Configuration Mode	configMode	string	Specifies how the operator determines the table (and which of its columns) to be read. Accepted values: - Static (from configuration): the operator does not expect any input. It reads from the table identified in <i>Table name</i>. If <i>Columns</i> is empty, all columns are retrieved. After outputting all relevant data, the operator becomes idle. - Dynamic (from input): the operator uses each input message to determine the name of the source table and which columns to fetch. These are obtained from the standard attributes in a Table Message. Default: " "
Table Name	tableName	string	The name of the source table in <i>Static</i> mode. Default: ""

Name	ID	Type	Description
Columns	Columns	array	The list of columns that are retrieved in <code>Static</code> mode. i Note Due to internal technical limitations, the operator is currently unable to distinguish between <code>TINYINT</code> and <code>BOOLEAN</code> columns, outputting values of either type as the former. To bypass this constraint, enable the Boolean flag for a column to have its value converted to class <code>bool</code>.
Read mode	readMode	array	Determines how the operator should read the table. Only used in <code>Static</code> mode. Accepted values: • Once: The operator reads the table only once, then become idle. • Poll: The operator periodically polls the table for data. Default: <code>[]</code>
Poll Period	pollPeriod	string	When <i>Read mode</i> is set to <i>Poll</i>, this property determines the interval between consecutive queries. Default: <code>"1s"</code>

Name	ID	Type	Description
Batching	batching	string	Determines how the operator's output should be batched. Accepted values: • None: no batching is performed. All rows in the target table are sent in a single output. • Fixed size: the output is broken into separate messages whose sizes is dictated by <code>Batch size</code>. When placed in a group with multiplicity, the different instances of the operator take measures to avoid repetition. Refer to the Multiplicity section for details and examples.
Batch Size	batchSize	integer	When <code>Batching</code> is set to <code>Fixed size</code>, this property determines the number of rows in each output. Default: 100
Connection timeout	connectionTimeout	string	The maximum amount of time to wait for the server. If at any time the operator does not hear from the server within this period of time, the connection is closed. If set to zero, the operator waits for the server indefinitely. Default: "50s"

Input

Input	Type	Description
input	message.table	A Table Message that specifies a table to be read. This is only applicable in <code>Dynamic</code> mode. The operator uses the following values from the input: • Name (<code>Attributes.table.name</code>) • Columns (<code>Attributes.table.columns</code>): if present and non-empty, specifies which columns are to be queried. Otherwise, all table columns are read.

Output

Output	Type	Description
success	message.table	A Table Message describing the table that was read and is holding the queried rows. In <code>Dynamic</code> mode, this output also includes any additional attributes that were present in the input.
error	message.error	An Error Message, in case an error was raised during an operation.

Multiplicity

When configured to perform fixed-size batching, this operator takes its multiplicity into account when performing queries. Its instances progressively query the table in chunks of rows, avoiding overlaps. Because each instance performs a separate query for each chunk, keep in mind that simultaneous changes to the source table may affect the behavior of this operator.

As an example, consider a source table of 65 rows and a Read HANA Table operator with multiplicity of 3 and Batch size of 10. Assuming the table remains unchanged during execution, the following reads might take place (due to concurrency, the order of events is not reliable):

- instance 0 outputs rows 0–9, 30–39 and 60–65;
- instance 1 outputs rows 10–19 and 40–49;
- instance 2 outputs rows 20–29 and 50–59.

2.11.3 Run HANA SQL

The Run HANA SQL operator executes user-provided SQL statements on an SAP HANA database.

Configuration Parameters

Name	ID	Type	Description
Connection	connection	object	Mandatory. The object containing the connection parameters to a HANA instance. Default: ""
Configuration Mode	configMode	string	Determines how the operator obtains the SQL code. Accepted values: • Static (from configuration): the operator executes the statement(s) in SQL and become idle. • Dynamic (from input): the operator executes SQL statement(s) from each input message. Default: " "
SQL	sql	string	The SQL statement(s) to be executed when in Static mode. This value is ignored in Dynamic mode. Default: ""

Name	ID	Type	Description
Transaction Level	transactionLevel	string	Determines how the operator should group statements into database transactions. Accepted values: • Per input: A separate transaction is used for each input, and each transaction encompasses all statements in that input. In <code>Static</code> mode, this means that the entire code in SQL is executed in a single transaction. • Per statement: A separate transaction is used for each statement, regardless of whether they came from the same input message. In <code>Dynamic</code> mode, each input message may override this configuration by setting the <code>transactionLevel</code> property in the <code>hana</code> object: <pre>msg.Attributes.hana = { transactionLevel: "Per input" };</pre> This applies exclusively to that input and bears no effect over upcoming ones. Default: "Per input"
Connection timeout	connectionTimeout	string	The maximum amount of time to wait for the server. If at any time the operator does not hear from the server within this period of time, the connection is closed. If set to zero, the operator waits for the server indefinitely. Default: "50s"

Input

Input	Type	Description
input	message.*	A message whose body is a string containing SQL commands. This only applies in <code>Dynamic</code> mode.

Output

Output	Type	Description
success	message.table	A Table Message holding retrieved data. For statements that do not query data, an empty table is output to signal completion. In <code>Dynamic</code> mode, this output also includes any additional attributes that were present in the input.
error	message.error	An Error Message, in case an error was raised during an operation.

2.11.4 Write to HANA Table

The Write to HANA Table operator ingests data into one or more tables on an SAP HANA database.

The target table(s) must already exist at initialization time (in Static mode) or whenever input is received (in Dynamic mode). If table creation is intended as part of the same pipeline, consider using the Initialize HANA Table operator in conjunction with this one.

Configuration Parameters

Name	ID	Type	Description
Connection	connection	object	Mandatory. The object containing the connection parameters to a HANA instance. Default: ""
Configuration Mode	configMode	string	Determines how the operator obtains the target table name. Accepted values: - Static (from configuration): The operator ingests all incoming data into the table specified in Table Name. - Dynamic (from input): The operator uses each input message to determine the destination. The table name obtained from the standard attributes in a Table Message. Default: " "
Table Name	tableName	string	The name of the table to be used as fixed destination in <code>Static</code> mode. Default: ""
Statement Type	statementType	string	Determines whether data is to be ingested using an <code>INSERT</code> or an <code>UPSERT</code> statement. Default: "INSERT"

Name	ID	Type	Description
Connection Timeout	connectionTimeout	string	The maximum amount of time to wait for the server. If the operator does not hear from the server within this period of time, the connection is closed. If set to zero, the operator waits for the server indefinitely.. Default: "50s"

Input

Input	Type	Description
input	message.table	A Table Message containing the data to be inserted.

Output

Output	Type	Description
success	message.table	A Table Message describing the table that was successfully initialized. In <code>Dynamic</code> mode, this output also includes any additional attributes that were present in the input, as well as the original message body.
error	message.error	An Error Message, in case an error was raised during an operation.

2.12 SAP Integration

Related Information

[Cloud Data Integration Consumer \[page 402\]](#)

[SAP Analytics Cloud Formatter \[page 403\]](#)

[SAP Analytics Cloud Producer \[page 406\]](#)

[SAP CP EM Consumer \[page 408\]](#)

[SAP CP EM Producer \[page 411\]](#)

[SAP CPI-PI iFlow \[page 414\]](#)

2.12.1 Cloud Data Integration Consumer

The Cloud Data Integration Consumer operator (`com.sap.dh.cloud_data_integration`) is implemented in the subengine model and uses Flowagent for execution. To effectively read from it, it is necessary to connect this operator to a Flowagent-based producer operator (for example, Flowagent File Producer).

Configuration Parameters

Parameter	Type	Description
Source	string	JSON object containing the remote object reference, schema, and filters of the required entity set.
Page size	number	Number of records to be fetched within one page (request).
Polling period (in minutes)	number	The duration of polling new records in Delta mode.
Extraction mode	string	Supports 2 modes of data extraction, Full and Delta.
Subscription ID	string	Unique external ID for creating subscriptions, required only in Delta mode.

Input

Input	Type	Description
inDataSet	string	Optional input port for the dataset in case not configured via the UI.
inTrigger	any	Optional port for triggering the operator execution.

Output

Input	Type	Description
outConfig	string.flowagent.config	Flowagent type port, which only Flowagent producers can consume (for example, Flowagent File Producer). To get the data as string, Flowagent CSV Producer can be used as producer.
outError	string	If mapped, the operator errors are sent to this port and the graph continues running. If not mapped, the graph terminates with the operator error.

2.12.2 SAP Analytics Cloud Formatter

Converts `message.table` input data to message format before sending to SAP Analytics Cloud. Use the SAP Analytics Cloud Formatter operator before the SAP Analytics Cloud Producer operator.

Configuration Parameters

Name	ID	Type	Description
Tenant	Tenant	string	Mandatory. Enter the SAP Analytics Cloud tenant name where data is pushed.
Target Dataset	Target Dataset	string	Mandatory. The available options are:

Name	ID	Type	Description
			Append: Choose an existing dataset where records are added to existing content. Replace: Overwrite all data in an existing dataset. New: Create a dataset. Default: "New"
Target Dataset ID	Target Dataset ID	string	Mandatory when the Target Dataset option is set to Append or Replace. i Note Replace will overwrite all data in a dataset with every request sent to SAC. Adjust Records per Request and Maximum Time in Buffer settings accordingly to prevent data loss. i Note Find the dataset ID in the SAP Analytics Cloud application by browsing to the desired dataset and extracting the ID from the URL query. For example, the dataset ID is shown in bold in this query: <code>https://SAC_HOST_NAME/sap/fpa/ui/tenants/TENANT_NAME/app.html#;view_id=ds-datasethome;datasetId=123456789ABCDEF</code>
Dataset Name	Dataset Name	string	Required when the Target Dataset option is set to New. Enter a unique name for a new dataset. i Note All datasets are created at the root level of your SAP Analytics Cloud file system, where they can be moved from within the SAP Analytics Cloud application.
Dataset Description	Dataset Description	string	Optional. Enter a description of the dataset contents.
Input Contains Column Headings	Input Contains Column Headings	bool	Required. The available options are: True: Operator uses the input column names as output column names. False: Define the output column names in the Output Schema option. You can use this option when the input dataset has column headings. There may be times when you want to change the column headings on output. Default: "false"
Output Schema	Output Schema	array	Required when the Input Contains Column Headings option is set to False. Define the output columns by clicking the pencil icon. Click + Add. Enter the column name, type, and length. Click Save. Repeat for the

Name	ID	Type	Description
			rest of the columns in the dataset. Use the up and down arrows to move the column names in the order in the dataset.
Records per Request	Records per Request	integer	Required when the Target Dataset option is set to Append. Enter a value greater than 0. The operator attempts to send the specified number of records in one request towards the SAP Analytics Cloud operator. The actual number may be less due to the Maximum Time in Buffer option.
Maximum Time in Buffer	Maximum Time in Buffer	integer	Required when the Target Dataset option is set to Append. Time is in milliseconds (ms). Set the value to 0 to send the data immediately. Set the value higher to include more records in the specified batch. For example, when Records per Request is set to 1000, and Maximum Time in Buffer is set to 10 ms, then the operator may not have enough time to send a batch of 1000 records. Therefore, only a portion of the records is output. If you want the records sent immediately, you might set Records per Request to 1 and Maximum Time in Buffer to 0.

Request Limits

The SAP Analytics Cloud application imposes these limits.

Option	Limit	Description
HTTP Request Size	100 MB	Maximum dataset size per HTTP request.
Row Count	2 Billion	Maximum accumulative dataset row count
Column Count	1000	Maximum dataset column count

Input

Input	Type	Description
Input	message.table	Table message with records to send to the SAP Analytics Cloud application. See Table Messages for more information on the <code>message.table</code> format.

Output

Output	Type	Description
Output	Message	Formatted message sent to the SAP Analytics Cloud Producer operator.

2.12.3 SAP Analytics Cloud Producer

Sends data to the SAP Analytics Cloud application.

Use the SAP Analytics Cloud Formatter operator before this operator so that the data is input in the correct format.

Configuration Parameters

Name	ID	Type	Description
host	host	String	Optional. The service's host optionally including the port
includeResponseHeaders	includeResponseHeaders	String	Optional. A list of response header names that should be included in the output message. Overridden by header <code>openapi.include_response_headers</code> .
oauth2Flow	oauth2Flow	String	Required. The flow used by "oauth2" security scheme. Currently the only possible value for this option is "accessCode", which corresponds to "authorization code" grant type.
oauth2TokenUrl	oauth2TokenUrl	String	Required. The token URL used by "oauth2" security scheme.
oauth2AuthUrl	oauth2AuthUrl	String	Required. The authorization URL used by "oauth2" security scheme when the flow is set to "accessCode".
oauth2ClientId	oauth2ClientId	String	Required. The client ID used by "oauth2" security scheme.
oauth2ClientSecret	oauth2ClientSecret	String	Required. The client secret used by "oauth2" security scheme. If no secret is configured for this OAuth client, leave the option empty.
oauth2AdditionalHeaders	oauth2AdditionalHeaders	String	Optional. Optional headers added in a token request, given as <code>{ "header1": "value", ... }</code> .
tlsCertificate	tlsCertificate	String	Optional. The file path to the TLS certificate file.
tlsKey	tlsKey	String	Optional. The file path to the TLS private key file.
tlsCa	tlsCa	String	Optional. The file path to the TLS certificate authority file.
tlsSkipVerify	tlsSkipVerify	Boolean	Required. When set to <i>True</i> , the client does not verify the server's certificate chain nor the host name.

Name	ID	Type	Description
timeout	timeout	Integer	Required. The timeout value in milliseconds to wait for a response.

i Note

When configuring the OAuth redirect URI in SAP Analytics Cloud, the format should be as follows:

```
http(s)://SAP_Datahub_host/app/pipeline-modeler/service/v1/runtime/internal/redirect
```

i Note

Any intermediate servers (for example, load-balancer) must be configured to not drop HTTP X-Forwarded-headers so that the pipeline instance can build the correct redirection URL.

Request Limits

i Posting Instructions

SAP Analytics Cloud imposes the following limits:

Limit	Value	Description
HTTP Request Size	100 MB	Maximum dataset size per HTTP request
Row Count	2 Billion	Maximum accumulative dataset row count
Column Count	1000	Maximum dataset column count

Input

Input	Type	Description
in	Message	Input message, where parameters are set in the headers and the optional content in body.

Output

Output	Type	Description
out	Message	Output message, where parameters are set in the headers and the optional content in body.

2.12.4 SAP CP EM Consumer

Receives messages from a queue in SAP Cloud Platform Enterprise Messaging and provides them for consumption in SAP Data Intelligence.

Configuration Parameters

Name	ID	Type	Description
CP EM Connection	emConnection		Connection to an SAP Cloud Platform Enterprise Messaging service. Default: none

Name	ID	Type	Description
Source Queue	source	string	SAP Cloud Platform Enterprise Messaging queue from which messages are read. The queue must exist on the SAP Cloud Platform Enterprise Messaging service instance. If the queue does not exist, an error message is sent to the <code>error</code> output. The queue name must start with the Message Client's namespace followed by a slash (<code>/</code>) and one or more alphanumerical components separated by a slash (<code>/</code>). Each component may contain capital or lowercase letters (<code>A-Z</code>, <code>a-z</code>), digits (<code>0-9</code>), underscores (<code>_</code>), hyphens (<code>-</code>), and dots (<code>.</code>). The queue name (including namespace) must not be longer than 164 characters. Default: none
No. of Reconnect Tries	reconnTries	integer	Number of attempts to reconnect the SAP CP EM Consumer to the SAP Cloud Platform Enterprise Messaging service after it has been disconnected (either due to an error or by the server; in the former case the error is provided at the <code>error</code> output port). Default: 0
Reconnect Wait Time	reconnWait	integer	Time in ms to wait between two consecutive reconnection attempts. Default: 0

Name	ID	Type	Description
Max. no. of Messages	maxCount	integer	Number of messages that is read from the SAP Cloud Platform Enterprise Messaging service before disconnecting the consumer. If set to 0, the consumer is not automatically disconnected from the SAP Cloud Platform Enterprise Messaging service. Default: 0
Log after Messages	logCount	integer	Number of received messages after which a log entry (stating the message count) is written. If set to 0, no message counts are logged. Default: 0

Input

None.

Output

Input	Type	Description
payload	message	A message containing the payload data received from SAP Cloud Platform Enterprise Messaging.
done	string	Indicates that the given <i>Max. no. of Messages</i> have been received by outputting <code>true</code> . This port never outputs anything if the <i>Max. no. of Messages</i> parameter is not set or set to 0.
error	message.error	A message containing an error.

2.12.5 SAP CP EM Producer

Publishes messages from SAP Data Intelligence to a topic in SAP Cloud Platform Enterprise Messaging.

Configuration Parameters

Name	ID	Type	Description
CP EM Connection	emConnection		Connection to an SAP Cloud Platform Enterprise Messaging service. Default: none
Target Topic	target	string	SAP Cloud Platform Enterprise Messaging topic to which messages are written. It must conform to the namespace of the Message Client; otherwise, an error message is sent to the <code>error</code> output. The topic name must start with the Message Client's namespace followed by a slash (<code>/</code>) and one or more alphanumerical components separated by a slash (<code>/</code>). Each component may contain capital or lowercase letters (<code>`A-Z`</code> , <code>`a-z`</code>), digits (<code>`0-9`</code>), and hyphens (<code>`-`</code>). The topic name (including namespace) must not be longer than 150 characters. Default: none

Name	ID	Type	Description
No. of Reconnect Tries	reconnTries	integer	Number of attempts to reconnect the SAP CP EM Consumer to the SAP Cloud Platform Enterprise Messaging service after it has been disconnected (either due to an error or by the server; in the former case the error is provided at the error output port). Default: 0
Reconnect Wait Time	reconnWait	integer	Time in ms to wait between two consecutive reconnection attempts. Default: 0
Max. no. of Messages	maxCount	integer	Number of messages that is read from the SAP Cloud Platform Enterprise Messaging service before disconnecting the consumer. If set to 0, the consumer is not automatically disconnected from the SAP Cloud Platform Enterprise Messaging service. Default: 0
Log after Messages	logCount	integer	Number of received messages after which a log entry (stating the message count) is written. If set to 0, no message counts are logged. Default: 0

Input

Input	Type	Description
payload	message	A message containing the payload data to be published via SAP Cloud Platform Enterprise Messaging.

Output

Input	Type	Description
ack	number	The number of messages that have successfully been sent to the SAP Cloud Platform Enterprise Messaging service by the SAP CP EM Producer. This output is generated after each successfully published message.
done	string	Indicates that the given <i>Max. no. of Messages</i> have been received by outputting <code>true</code> . This port never outputs anything if the <i>Max. no. of Messages</i> parameter is not set or set to 0.
error	message.error	A message containing an error.

2.12.6 SAP CPI-PI iFlow

The SAP CPI-PI iFlow operator provides the possibility to trigger iFlows in an SAP CPI system. It calls the set CPI system sending the received input data as payload. The response is emitted via the outbound port. Currently, only HTTP Basic Authentication is supported.

Configuration Parameters

Name	ID	Type	Description
Connection	connection	object	Mandatory. Connection parameters and credentials setting the CPI backend to communicate with.
Use Custom HTTP verb	customHttpVerbToggle	boolean	Enables providing a custom HTTP verb instead of a standard one (as defined in RFC 7231). Default: <code>false</code>
HTTP verb	httpVerb	string	Select or enter the HTTP verb that should be used for making the request. Toggles between a text field and a list according to whether a custom HTTP verb should be used. Default: <code>POST</code>
iFlow path	iFlowPath	string	Mandatory: Points to the iFlow that shall be triggered. This is the portion of the URL following the slash after the hostname or domain (<code>https://my.cpisystem.sap.com/</code>). This information can be found in CPI's dashboard after deploying a given iFlow. Must not start with a slash (<code>/</code>).
CSRF Protection Enabled for iFlow	csrfProtectediFlow	boolean	Needs to be enabled for iFlows with the very common CSRF protection activated. Default: <code>true</code>

Name	ID	Type	Description
Content Type	contentType	string	The content type that will be stated in the outgoing request. Common values are <code>application/json</code>, <code>application/xml</code> or <code>text/xml</code>. Default: <code>application/json</code>
Retry attempts	retryAttempts	integer	In case an HTTP(S) request to the CPI system fails, it will be retried <code>n</code> times. Only responses in the range of <code>5xx</code> status codes trigger a retry; <code>4xx</code> status codes will lead to an immediate failure (for example, authentication issues). Waiting time between requests will grow exponentially but is capped at 20 seconds. Default: 1
Verify TLS certificates	verifyTLSCerts	boolean	In case the CPI system uses a self-signed certificate, you may want to disable the certificate validation performed during the TLS handshake.  Caution Use this with caution as the replying host's identity will no longer be ensured. Default: <code>true</code>

Name	ID	Type	Description
Outgoing request timeout	requestTimeout	string	Defines the maximum amount of time a request outgoing to the set CPI system is allowed to take before being canceled (this does not necessarily stop the triggered iFlow inside CPI). Valid suffixes are ms, s, m, h. Setting this to 0s results in no timeout. Independently, underlying software or hardware layers may enforce shorter timeouts. Default: 60s

Input

Input	Type	Description
input	message	Receives DH message objects. By setting the attribute <code>cpi.iflowpath</code>, it is possible to dynamically override the set iFlow path. By setting attributes with the prefix <code>cpi.customheader</code>, it is possible to define custom headers (and also override other headers like <code>content-type</code>) in a programmatic way. For example, a header named SOAPAction could be set by adding an attribute <code>cpi.customheader.SOAPAction</code> to the message sent to the operator.

Output

Output	Type	Description
output	message	Emits messages containing the response of the CPI system. In case an error occurred, the error message, if any, will be contained. The <code>message.error</code> attribute will be set accordingly.

2.13 SAP Machine Learning Core Operators

This contains Modeler operators and Modeler graph templates that support core functions in SAP Data Intelligence.

Usage

Under the section “SAP Machine Learning Core Operators”, we provide a set of core operators for creating machine learning pipelines. These operators provide capabilities such as model training and serving and integrate with the scenario management features of the platform.

Artifact-Based Flow

These operators interact by exchanging artifact information. In the pipeline, only a reference to artifacts is passed between operators. The data itself is not sent through the pipeline, but always resides on the data lake.

Artifacts are handled by operator ports of type `message.artifact`. Messages of that type contain all relevant information in the message header and have an empty body. The header attribute `artifactID` is required and uniquely identifies the artifact. Additional attributes are provided, such as the artifact name or training job status, and are useful information in debugging the graph.

Artifact Producer and Consumer

The artifact producer operator allows any blob to be stored as an artifact, which can be used in subsequent ML pipeline stages.

The artifact consumer outputs the content of an artifact as a blob and allows the artifacts to be used in operators, which expect the content instead of an artifact reference.

Training Operator

The training operator allows to receive multiple data as an input for training. Ingoing artifacts can be specified via `id` in the operator configuration. The `id` field can be filled with a placeholder, which is filled out before executing the pipeline. To obtain a value helper in scenario manager, please use a placeholder of the following form: `${ARTIFACT: [MODEL | DATASET] : <NAME>}` (you need to choose between `DATASET` and `MODEL` and specify an artifact name for `.`). Alternatively, you can add custom in-ports of type `message.artifact` to the operator. These in-ports allow to integrate the training operator in an artifact based flow. Similarly, additional ports can be added to the operator for outgoing artifacts.

A set of training images is provided. See the documentation at [ML Training \[page 454\]](#) for the current list.

Inside the training script, incoming data can be access via the "sapdi" python module:

```
artifact = sapdi.get_artifact(<artifact alias>)
path = artifact.get_path()
```

To register an output artifact use the following function:

```
out_artifact = sapdi.create_artifact(<artifact alias>,
file_type=sapdi.artifact.ArtifactFileType.ZIP, description="Best mnist model
ever", artifact_name="MNIST")
path = (out_artifact.get_path())
```

In case of `ArtifactFileType.ZIP`, the output folder is automatically zipped before the artifact is generated. Other options are `FILE` or `FOLDER`. The is either the name of the port, or specified together with the artifact id in the operator configuration.

Model Operator

The model operator accepts a model artifact as part of the operator configuration. When the pipeline starts executing, the model is deployed and requests coming in via the operator in port are served by the model.

Modeler Operators

- [Artifact Producer \[page 420\]](#)
- [Artifact Consumer \[page 419\]](#)
- [Model Serving Operator \[page 444\]](#)
- [Submit Metrics \[page 453\]](#)
- [ML Training \[page 454\]](#)

Related Information

[Artifact Consumer \[page 419\]](#)

[Artifact Producer \[page 420\]](#)

[HANA ML Forecast Operator \[page 424\]](#)

[HANA ML Inference Operator \[page 432\]](#)

[HANA ML Training Operator \[page 437\]](#)

[Model Serving Operator \[page 444\]](#)

[Submit Metrics \[page 453\]](#)

[ML Training \[page 454\]](#)

2.13.1 Artifact Consumer

Receives the `id` of a registered artifact and outputs its metadata. It is intended to be followed by an operator to retrieve the artifact file(s) from the Semantic Data Lake (SDL). For an example of usage of this operator, refer to the `Python Consumer` template pipeline.

Input

Input	Type	Description
inArtifact	message	The metadata of the artifact to be consumed. Only <code>Attributes.artifact.id</code> is used. <pre>{ "Attributes": { "artifact": { "id": "<artifactID>", "name": "data1", "kind": "dataset" } } }</pre>

Output

Output	Type	Description
OutArtifact	message	The input message with additional attributes including the artifact's file reference as specified in a <code>message.file</code> object. For artifacts consisting of one file, this message can be consumed by the Read File [page 303] operator for the retrieval of the artifact file. If the artifact path corresponds to a folder, a List Files [page 296] operator (with the option <i>List directories</i> set to "False") can precede the Read File [page 303] operator. These operators should be configured to Read and List "On input", resp., and to receive the connection from input.

Output	Type	Description
outError	message.error	Upon error, an error message if the port is connected. Otherwise, the operator fails and terminates the graph. See the error handling documentation for details on the format of error messages.

2.13.2 Artifact Producer

The Artifact Producer operator registers the artifact (dataset, model or other) given as input and returns the artifact's generated ID.

It supports two modes for artifact registration:

- Given the contents of a file, it uploads the file to the Semantic Data Lake (SDL) and registers it as part of the current ML scenario.
- Given the location of an existing file or folder in the SDL, it registers the file or folder as part of the current ML scenario.

For the first mode, the operator is intended to work together with the Write File operator. For an example of usage of this operator, refer to the Python Producer template pipeline.

Note

- The operator does not support receiving interleaving messages with batches corresponding to different artifacts.
- The operator cannot be used with multiplicity greater than one.
- The operator can only be used in pipelines created from the [ML Scenario Manager](#) app.

Configuration Parameters

Name	ID	Type	Description
Artifact Name	artifactName	string	Mandatory. The name of the artifact to be registered. Default: ""
Artifact Kind	artifactKind	string	Mandatory. The type of the artifact. • Possible values: ◦ dataset ◦ model ◦ other Default: dataset

Name	ID	Type	Description
Artifact Description	description	string	Can be used to describe the produced artifact. Default: ""
Filename Prefix	prefix	string	Prefix for the artifact's file-name. Only applicable if the input message contains file contents. Default: ""
Filename Suffix	suffix	string	Suffix for the artifact's file-name. Only applicable if the input message contains file contents. Default: ""

Input

Input	Type	Description
InArtifact	message	A message including the artifact file and parameters. - Attributes (optional): - File properties as specified in a <code>message.file</code> object. <code>path</code>: if the message body is empty, the location of the file(s) in the SDL to be registered as an artifact (in this case connection should equal <code>DI_DATA_LAKE</code>). - If files are to be uploaded in batches, <code>batching</code> attributes. - Body: if not empty, the body represents the contents of the file to be registered as an artifact. It must be a <code>blob([]byte in Golang)</code> - Encoding (optional): encoding of the body. - Examples: - One file upload: <pre>{ "Attributes": { }, "Encoding": "blob", "Body": [104 101 108 108 111] }</pre> - One file upload in batches: <pre>{ "Attributes": { "batchIndex": 3, "lastBatch": "false", "batchCount": 9 }, "Encoding": "blob",</pre>

Input	Type	Description
		<pre>"Body": [101 109 104 102 110] }</pre> Register files in the Semantic Data Lake: <pre>{ "Attributes": { "file": { "connection": "DI_DATA_LAKE", "path": "/ shared/ml/ datasets/ dataset1.csv" } }, "Encoding": "blob", "Body": [] }</pre>
inFileReturn	message.file	Used to consume the output of a Write File operator when uploading files. By default, this port should be connected to the output of a Write File operator.

Output

Output	Type	Description
OutArtifact	message	The generated metadata of the artifact. The following metadata are included in the message: <pre>{ "Attributes": { "artifact": { "id": "<artifactID>", "name": "data1", "kind": "dataset" } } }</pre>

Output	Type	Description
outError	message.error	Upon error, an error message if the port is connected. Otherwise, the operator fails and terminates the graph. See the error handling documentation for details on the format of error messages.
outFileSend	message.file	The file information to be sent to the Write File operator to upload the artifact. By default, this port should be connected to the input of a Write File operator.

2.13.3 HANA ML Forecast Operator

The HANA ML Forecast operator connects to an SAP HANA database and leverages algorithms from SAP HANA Predictive Analytics Library (PAL) or SAP HANA Automated Predictive Library (APL) that do not require a model to be persisted. Instead, the model is generated on-the-fly based on the input data, and a series of results are then forecast automatically.

Input data for the forecast is supplied either from a table in the SAP HANA database (using the `Dataset` parameter) or via the `data` input port. This behavior can be changed in the [Perform Inference on Input Data](#) configuration, described below. In case of successful execution, the operator outputs the forecasted results.

For a common use case of this operator, check the [HANA ML Forecast Template \[page 682\]](#).

Configuration Parameters

Name	ID	Type	Description
Connection	connection	object	Mandatory. The object containing the connection parameters to a SAP HANA database instance with PAL or APL libraries installed. By default, the operator sets the SAP HANA database schema to the username of the connection. Default: {}

Name	ID	Type	Description
Data source	data_source	enum	Mandatory. The source of the data used for forecasting. If set to <code>Input Data Port</code>, the operator uses data incoming to data port to perform the forecast (see format in request examples below) . If set to <code>HANA DB</code>, a database table must be specified in the Dataset configuration. Default: <code>Input Data Port</code>
Dataset	tableName	string	This field is only available if the data source is set to <code>HANA DB</code>. It represents the table name containing the data to be used to generate the model. Refer to the Case Sensitivity section below for applicable naming rules. Default: <code>""</code>
Algorithm		enum	Mandatory. The algorithm that will be used to train the model (see Supported Algorithms [page 431]). Default: <code>Single Exponential Smoothing (PAL)</code>
Key Column	key	string	Name of the column containing the inference table key. Refer to the Case Sensitivity section below for applicable naming rules. Default: <code>""</code>

Name	ID	Type	Description
Parameters for Fitting		object	This field is only available for ARIMA (PAL) and Auto ARIMA (PAL) algorithms. It represents any additional parameters for the training. These should be provided in JSON format and are dependent on the algorithm selected. All fitting parameter values are case-sensitive. For details on the parameters supported for the training, please check the algorithm documentation available here. For example: { "endog" : "Y" } Default: {}
Parameters for Prediction		object	This field is only available for ARIMA (PAL) and Auto ARIMA (PAL) algorithms. It represents additional parameters for the prediction. These should be provided as JSON format and are dependent on the algorithm selected. All fitting parameter values are case-sensitive. For details on the parameters supported for the training, please check the algorithm documentation available here. For example: <pre>{ "forecast_length": 10 }</pre> Default: {}

Name	ID	Type	Description
Parameters		object	This field is available for the Exponential Smoothing and Change Point Detection algorithms. It represents any additional parameters used for fitting the model. These should be provided in JSON format and are dependent on the algorithm selected. All parameter values are case-sensitive. For details on the parameters supported for the prediction, please check the algorithm documentation available here. For example: { "endog" : "Y" } Default: {}
Send Inference Results to		enum	Destination of the inference results. Default: Output port
Results Table Name		string	This field is only available if Send Inference Results to is set to HANA table or Both. Name of the HANA table to store the inference results. Refer to the Case Sensitivity section below for applicable naming rules. Default: ""

Name	ID	Type	Description
Insert Mode		enum	This option is only available if <code>Send Inference Results</code> is set to HANA table or Both. Action to take in case the HANA table specified above already exists. Possible values are: • Fail: Graph aborts with an error • Overwrite: Existing data is deleted and replaced with the result of the current inference. • Append: The result of the current inference is added to the existing table. Default: Fail
Output Format		enum	Format of the forecast output. Default: Column-oriented
Show Advanced Settings		boolean	Displays advanced settings for operator. Default: False
Retry Attempts		integer	The number of times the operator will retry an operation that fails. If a negative value is provided, the operator will use the default value instead. This field is only visible if <code>Show Advanced Settings</code> is set to True. Default: 3
Retry Period (ms)		numeric	The waiting time in milliseconds between consecutive retry attempts. If a negative value is provided, the operator will use the default value instead. This field is only visible if <code>Show Advanced Settings</code> is set to True. Default: 500

Input

Input	Type	Description
data	message	If Data Source is set to Input Data Port, training and forecast is performed using data incoming on this port. It should contain data in json format in the message body. Please ensure the type of data used in each of the fields is supported by the underlying HANA algorithm (supported types may vary according to the HANA DB version). You may check details for each algorithm in its respective page in the SAP Help Portal, for example: Single Exponential Smoothing for HANA 2.0 SPS04. The operator supports requests incoming from the two endpoints below, which are exposed by default by the HANA ML Forecast Template [page 682] • <code>/v1/inference</code>: Supports payloads containing inference data only. For example:

```
{
  "ID": [1, 2, 3],
  "Y": [10, 20, 30]
}
```

- `/v2/inference`: Supports payloads containing data and/or parameters, which vary according to the selected algorithm.

❖ Example

For Exponential Smoothing and Change Point Detection algorithms:

```
{
  "data": {
    "ID": [1,
2, 3],
    "Y": [10,
20, 30]
```

Input	Type	Description
		<pre> }, "hyper_parameters" : { "hyper_param1": "value1" }, "parameters": { "param1": "value1" } } </pre> For ARIMA and Auto ARIMA algorithms: <pre> { "data": { "ID": [1, 2, 3], "Y": [10, 20, 30] }, "hyper_parameters" : { "hyper_param1": "value1" }, "fit_parameters": { "param1": "value1" }, "inference_parameters": { "param1": "value1" } } </pre> Additionally, the operator supports changing the selected algorithm upon each request: <pre> { "data": { "ID": [1, 2, 3], "Y": [10, 20, 30] }, "algorithm": "Triple </pre>

Input	Type	Description
		<pre>Exponential Smoothing (PAL)", "key": "ID" }</pre>

Output

Output	Type	Description
result	message	The forecast results.

Case Sensitivity

Identifiers in SAP HANA are case-insensitive by default, so they are implicitly capitalized. When written in double-quotes, however, they become case-sensitive and must always be spelled exactly the same as when they were created. With the exception of Hyper Parameters section, when using this operator, you must take case sensitivity into account: if a schema, table, or column was created using quotes, then its name in the operator's configuration must also be in quotes and in the same case.

The snippet below shows some examples and explanations on case sensitivity:

```
-- Since we're not using quotes, mySchema will be capitalized: MYSCHEMA.
CREATE SCHEMA mySchema;
-- myTable, B and c are all case-sensitive.
CREATE TABLE MYSCHEMA."myTable" (A INTEGER, "B" VARCHAR(1), "c" TIMESTAMP);
-- This statement will run successfully because b is not in quotes, so it
-- will be capitalized, becoming B, which is the correct name of the column.
INSERT INTO myschema."myTable" (a, b, "c") VALUES (0, 'x', NOW());
-- This statement will fail because c has to be quoted and lowercase.
INSERT INTO myschema."myTable" (a, b, c) VALUES (1, 'y', NOW());
```

Supported Algorithms

Time Series Algorithms

- [ARIMA \(PAL\)](#)
- [Auto ARIMA \(PAL\)](#)
- [Auto Time Series \(APL\)](#)
- [Change-Point Detection \(PAL\)](#)
- [Auto Exponential Smoothing \(PAL\)](#)

- [Single Exponential Smoothing \(PAL\)](#)
- [Double Exponential Smoothing \(PAL\)](#)
- [Triple Exponential Smoothing \(PAL\)](#)

2.13.4 HANA ML Inference Operator

Connects to a SAP HANA database instance and applies the specified machine learning model, leveraging SAP HANA Predictive Analytics Library (PAL) or SAP HANA Automated Predictive Library (APL). The input data for inference is supplied either from a table in the SAP HANA database (using the `Inference Dataset` parameter) or via the `inferenceData` input port. In case both are supplied, only data from the `Inference Dataset` parameter is processed.

In case of successful execution, it outputs the result of the inference.

This operator is designed to support the following use cases:

- Inference on data incoming through HTTPS requests: Select this mode by setting configuration `Data Source` = "Input Data Port" (see below). For an example of this use case, refer to the [HANA ML Inference on API Requests \[page 680\]](#) graph template.
- Inference on existing dataset on HANA: Select this mode by setting configuration `Data Source` = "HANA DB" (see below). For an example of this use case, see the [HANA ML Inference on Dataset \[page 680\]](#) graph template.

Configuration Parameters

Name	ID	Type	Description
Connection	connection	object	Mandatory. The object containing the connection parameters to a SAP HANA database instance with PAL or APL libraries installed. By default, the operator processing sets the SAP HANA database schema to the username of the connection. Default: {}
Key Column	key	string	Mandatory. Name of the column containing the inference table key. Refer to the Case Sensitivity section below for applicable naming rules. Default: ""

Name	ID	Type	Description
Data Source	data_source	enum	Mandatory. The source of the data used for inference. If set to Input Data Port, the operator uses data incoming to inferenceData port to perform the inference. If set to HANA DB, a database table specified in the Inference Dataset configuration is used to perform the inference, and any data incoming through the port is ignored. Default: Input Data Port
Inference Dataset	tableName	string	This field is only available if the previous configuration is set to HANA DB. It represents the table containing the data to be used for inference. The table must contain the fields in the same order as the table used for the training, with the exception of the target column (if applicable). Refer to the Case Sensitivity section below for applicable naming rules. Default: ""
Parameters for Prediction	-	-	Additional parameters for the prediction. They should be provided as JSON and are dependent on the algorithm selected. All prediction parameter values are case-sensitive. For details on the parameters supported for the prediction, please check the algorithm documentation available at the Python Client API for Machine Learning documentation.

Name	ID	Type	Description
Send Inference Results to	-	enum	Destination of the inference results. Default: Output port
Results Table Name	-	string	This field is only available if <code>Send Inference Results to</code> is set to HANA table or Both. Name of the HANA table to store the inference results. Refer to the Case Sensitivity section below for applicable naming rules. Default: ""
Insert Mode	-	enum	This option is only available if <code>Send Inference Results to</code> is set to HANA table or Both. Action to take in case the HANA table specified above already exists. Possible values are: • Fail: Graph aborts with an error. • Overwrite: Existing data is deleted and replaced with the result of the current inference. • Append: The result of the current inference is added to the existing table. Default: Fail
Output Format	-	enum	Format of the inference output. Default: Column-oriented
Show Advanced Settings	-	boolean	Displays advanced settings for operator. Default: False

Name	ID	Type	Description
Retry Attempts	-	integer	The number of times the operator retries an operation that fails. If a negative value is provided, the operator will use the default value instead. This field is only visible if <code>Show Advanced Settings</code> is set to <code>True</code>. Default: 3
Retry Period (ms)	-	numeric	The waiting time in milliseconds between consecutive retry attempts. If a negative value is provided, the operator uses the default value instead. This field is only visible if <code>Show Advanced Settings</code> is set to <code>True</code>. Default: 500

Input

Input	Type	Description
model	blob	The model that is used for the inference.

Input	Type	Description
inferenceData	message	If Data Source is set to Input Data Port, inference is applied to data incoming on this port. It should contain inference data in json format in the message body, having the fields in the same order as the table used for the training, with the exception of the target column (if applicable). Please ensure the type of data used in each of the fields is supported by the underlying HANA algorithm (supported types may vary according to the HANA DB version). You may check details for each algorithm in its respective page in the SAP Help Portal, for example: Classification Decision Trees for HANA 2.0 SPS04. The operator supports requests incoming from the two endpoints below, which are exposed by default by the HANA ML Inference graph template. • <code>/v1/inference</code>: Supports payloads containing inference data only. For example: <pre> { "ID": ["2"], "X1": [-1.0872], "X2": ["A"] }</pre> • <code>/v2/inference</code>: Supports payloads containing data and inference parameters. For example: <pre> { "data": { "ID": ["2"], "X1": [-1.0872], "X2": ["A"] }, "inference_parameters": { "param1": "value1" } }</pre>

Output

Output	Type	Description
result	message	The result of the inference.

Case Sensitivity

Identifiers in SAP HANA are case-insensitive by default, so they are implicitly capitalized. When written in double-quotes, however, they become case-sensitive and must always be spelled exactly the same as when they were created. With the exception of Hyper Parameters section, when using this operator, you must take case sensitivity into account: if a schema, table, or column was created using quotes, then its name in the operator's configuration must also be in quotes and in the same case.

The snippet below shows some examples and explanations on case sensitivity:

```
-- Since we're not using quotes, mySchema will be capitalized: MYSCHEMA.  
CREATE SCHEMA mySchema;  
-- myTable, B and c are all case-sensitive.  
CREATE TABLE MYSCHEMA."myTable" (A INTEGER, "B" VARCHAR(1), "c" TIMESTAMP);  
-- This statement will run successfully because b is not in quotes, so it  
-- will be capitalized, becoming B, which is the correct name of the column.  
INSERT INTO myschema."myTable" (a, b, "c") VALUES (0, 'x', NOW());  
-- This statement will fail because c has to be quoted and lowercase.  
INSERT INTO myschema."myTable" (a, b, c) VALUES (1, 'y', NOW());
```

2.13.5 HANA ML Training Operator

Connects to a SAP HANA database and generates a machine learning model using the specified algorithm leveraged by SAP HANA Predictive Analytics Library (PAL) or SAP HANA Automated Predictive Library (APL). In case of successful execution, it outputs the generated model and its associated metrics (if a test dataset is provided or dataset is split into training/testing).

i Note

This operator is designed to work together with the [Artifact Producer \[page 420\]](#) operator, which should be connected to the `model` output port.

For a common use case of this operator, check the [HANA ML Training Template \[page 681\]](#).

Configuration Parameters

Name	Type	Description
Connection	object	Mandatory. The object containing the connection parameters to a SAP HANA database instance with PAL and APL libraries installed. By default, the operator sets the SAP HANA database schema to the username of the connection. Default: {}
Dataset	string	Mandatory. The name of the table containing the dataset. Refer to the Case Sensitivity section below for applicable naming rules. Default: ""
Features	array	Mandatory. An array of objects describing the columns from the dataset that will be considered as features for the model. <pre>[{"name": "ID", "type": "VARCHAR", "size": 50}, {"name": "X1", "type": "DOUBLE"}, {"name": "X2", "type": "DOUBLE"}]</pre> Example: Default: []
Task	enum	The type of model that will be generated. Different algorithms will be available for different tasks. Default: Classification
Algorithm	enum	Mandatory. The algorithm that will be used to train the model (see Supported Algorithms [page 443]). Default: Decision Tree Classifier (PAL)

Name	Type	Description
Split Dataset for Testing	boolean	Example: Whether you would like to split dataset into training and test set. This field is only available for task of types Classification and Regression.
Seed	numeric	This field is available if Split Dataset for Testing is set to True. It indicates the seed used to initialize the random number generator in order to split the dataset into training and test set. The system time is used if the supplied number is equal to zero. If no value is provided or the provided value is less than zero, an internally fixed value is used. Default: None
Test Ratio	numeric	This field is available if Split Dataset for Testing is set to True. It represents the ratio from the dataset that is used as test data. The value must be in the closed interval between 0 and 1. Values outside the range are truncated (for example, 2 will be truncated to 1). If the value is 0 or not specified, no test is performed and no metrics are calculated. For example, if value is 0.2, approximately 80% of data rows that are on Dataset table are used for training and 20% for scoring the model, generating performance metrics. Note This functionality is only available in SAP HANA systems with PAL installed.

Name	Type	Description
Test Dataset	string	This field is available if <code>Split Dataset for Testing</code> is set to False. It represents the name of the table containing the test dataset to be used to score the model, generating performance metrics. If no <code>Test Dataset</code> is provided, no test is performed and no metrics will be calculated. Default: ""
Key Column	string	Name of the column containing the training table key. Refer to the Case Sensitivity section below for applicable naming rules. Default: ""
Target Column	string	Mandatory. Name of the column containing the target variable in the training table. If the target column is also present in the Training Features field, it are disregarded as feature and considered as target only. This field is only available for task of types <code>Classification</code> and <code>Regression</code>. Refer to the Case Sensitivity section below for applicable naming rules. Default: ""

Name	Type	Description
Hyper Parameters	object	Additional parameters that can be provided to fine-tune the model. They should be provided as a JSON and its format is dependent on the algorithm selected. All hyper parameter values are case-sensitive. For a comprehensive list of hyper parameters, please check the HANA ML API documentation at Python Client API for machine learning algorithms. Example for kmeans: <pre>{ "n_clusters": 2, "distance_level": "euclidean", "max_iter": 100 }</pre> Example for neural network: <pre>{ "categorical_variable": ["X1"], "hidden_layer_size": "(10, 10)", "hidden_layer_size_options": ["(10, 10)", "(10, 10)"], "activation": "tanh", "output_activation": "tanh", "learning_rate": 0.001, "random_state": 1, "weight_init": "all-zeros", "momentum": 0.0001 }</pre> i Note For <code>hidden_layer_size</code> and <code>hidden_layer_size_options</code> parameters, the tuple contents should be inside a string. Default: {}

Name	Type	Description
Parameters for Fitting	object	Additional parameters for the training. They should be provided as JSON format and are dependent on the algorithm selected. All fitting parameter values are case-sensitive. For details on the parameters supported for the training, check the algorithm documentation at Python Client API for machine learning algorithms. Example: <code>{"class_map0": "class A", "class_map1": "class B"}</code> Default: <code>{}</code>
Show Advanced Settings	boolean	Displays advanced settings for operator. Default: <code>False</code>
Retry Attempts	integer	The number of times the operator will retry an operation that fails. If a negative value is provided, the operator will use the default value instead. This field is only visible if <code>Show Advanced Settings</code> is set to <code>True</code>. Default: <code>3</code>
Retry Period (ms)	numeric	The waiting time in milliseconds between consecutive retry attempts. If a negative value is provided, the operator will use the default value instead. This field is only visible if <code>Show Advanced Settings</code> is set to <code>True</code>. Default: <code>500</code>

Input

Input	Type	Description
trigger	any	A port that triggers the operator as soon as it receives any content. The content is ignored by the operator.

Output

Output	Type	Description
model	blob	The generated model in binary format.
metrics	message	A key-value pair dictionary containing the metrics and their respective values. Metrics will be generated only if tests were executed during the training.

Case Sensitivity

Identifiers in SAP HANA are case-insensitive by default, so they are implicitly capitalized. When written in double-quotes, however, they become case-sensitive and must always be spelled exactly the same as when they were created. With the exception of Hyper Parameters section, when using this operator, you must take case sensitivity into account: if a schema, table, or column was created using quotes, then its name in the operator's configuration must also be in quotes and in the same case.

The snippet below shows some examples and explanations on case sensitivity:

```
-- Since we're not using quotes, mySchema will be capitalized: MYSCHEMA.  
CREATE SCHEMA mySchema;  
-- myTable, B and c are all case-sensitive.  
CREATE TABLE MYSCHEMA."myTable" (A INTEGER, "B" VARCHAR(1), "c" TIMESTAMP);  
-- This statement will run successfully because b is not in quotes, so it  
-- will be capitalized, becoming B, which is the correct name of the column.  
INSERT INTO myschema."myTable" (a, b, "c") VALUES (0, 'x', NOW());  
-- This statement will fail because c has to be quoted and lowercase.  
INSERT INTO myschema."myTable" (a, b, c) VALUES (1, 'y', NOW());
```

Supported Algorithms

Classification Algorithms

- [Decision Tree Regressor \(PAL\)](#) - CART is the default training algorithm. It can be changed in [Hyper Parameters](#) section.
- [Hybrid Gradient Boosting \(PAL\)](#)
- [Logistic Regression \(PAL\)](#)
- [Neural Network \(PAL\)](#)
- [Random Forest \(PAL\)](#)
- [Auto Classifier \(APL\)](#)
- [Gradient Boosting \(APL\)](#)
- [Gradient Boosting Binary \(APL\)](#)

Regression Algorithms

- [Decision Tree Regressor \(PAL\)](#) - CART is the default training algorithm. It can be changed in [Hyper Parameters](#) section.
- [Hybrid Gradient Boosting \(PAL\)](#)
- [Linear Regression \(PAL\)](#)
- [Neural Network \(PAL\)](#)
- [Random Forest \(PAL\)](#)
- [Auto Regressor \(APL\)](#)
- [Gradient Boosting \(APL\)](#)

Clustering Algorithms

- [DBSCAN \(PAL\)](#)
- [KMeans \(PAL\)](#)
- [Unsupervised Clustering \(APL\)](#)

Time Series Algorithms

- [ARIMA \(PAL\)](#)
- [AutoTimeSeries \(APL\)](#)

2.13.6 Model Serving Operator

The Model Serving operator deploys Machine Learning (ML) models and handles inference request for the deployed models.

It does three main functions:

- **Deploy Model(s):**
When a pipeline is started, the operator deploys one or more models (as configured) and exposes an HTTP/REST endpoint for inference requests.
- **Inference a model:**
When a pipeline is running and the model deployment is successful, the operator becomes a client that sends your inference requests to one of the deployed models.
- **Undeploy Model(s):**
When a pipeline is stopped, the operator undeploys your model server to free up the resources.

Prerequisites

- **Model Artifacts:** More than one model can be deployed on a model server. Each model is exported in [TensorFlow SavedModel](#)  format or exported using SAP DI Python SDK. The whole model should be zipped and registered as a model artifact in an ML Scenario.

Configuration Parameters

Parameter	Type	Description
artifactId	string	Mandatory if the <code>models</code> array is empty. Artifact ID of an ML model created from a training or model producing pipeline. It must be set to <code>{ARTIFACT:MODEL}</code> to be able to choose available models in ML Scenario Manager.

Parameter	Type	Description
Models	array	Mandatory if <code>artifactId</code> is not configured or value is empty. This is a list to add one or more models to be served. Each item in the list represents a model and contains two fields: Name and ID (Mandatory). Multiple models can be deployed only using <code>Models</code> configuration: • Name (type string, optional): field value will be used as <code>Model Name</code> during the deployment. ◦ This is an optional field. If it's empty, the ID (<code>artifactId</code>) value would be used as a <code>Model Name</code>. ◦ Name field value must be unique in the Models Configuration. The graph will fail and terminate when duplicate names are defined. ◦ You can use only alphanumeric characters, hyphens, and underscores. The value must not exceed 64 characters, otherwise the graph will terminate. • ID: field is to receive the value as <code>artifactId</code> of a ML model created from a training or model producing pipeline. ◦ It should be configured as <code>{ARTIFACT:MODEL:<modelname>}</code> so that it can be used in <i>ML Scenario Manager</i> to select from the registered models. <code>modelname</code> value should be configured as registered model name in the <i>ML Scenario Manager</i> so that model with that <code>modelname</code> will be listed in the drop down

Parameter	Type	Description
		during the pipeline deployment. i Note Specifying both <code>artifactId</code> and <code>Models</code> is not allowed. The graph will not ruin when both <code>artifactId</code> and <code>Models</code> configuration are specified.
<code>modelRuntimeId</code>	string	Optional. Model runtime ID. Supported model runtimes are: - TensorFlow 1.15 (<code>tf-1.15</code>) in TensorFlow SavedModel format. - ML Serving 1.1 (<code>mlserving-1.1</code>) in the model format exported using the SDK (for more information, see the tutorial Deploying scikit-learn Models in an Inference Service). Default: <code>tf-1.15</code>
<code>resourcePlanId</code>	string	Optional. Resource Plan ID. Accepted values are <code>basic</code>, <code>basic.8x</code>, <code>standard</code>, and <code>advanced</code>. Default: <code>basic</code>
<code>replicas</code>	string	Optional. Number of replica. Specify how many replica of model server instances that host your model. For example, if you specified 2 replicas with the resource plan <code>basic</code>, the operator will deploy your model in two identical model server with basic resource plan. Default: 1

Parameter	Type	Description
batchConfig	string	Optional. Only applicable to <code>tf-1.x</code> runtimes. Batch Config Parameter for a TensorFlow model server. If you are deploying your batch-enabled TensorFlow models in GPU, you may want to enable batching to improve the performance of your inference request. we've enabled the following parameters to enable batching: • <code>maxBatchSize</code> • <code>batchTimeoutMicros</code> • <code>maxEnqueuedBatches</code> Please see TensorFlow Serving Batch-ing guide  for details. Default: <code>null</code>

Input

Input	Type	Description
input	message	Mandatory. Accepts JSON data (inference request) in a message's body. For TensorFlow models, the JSON data format is based on the TensorFlow Predict API Request Format as shown in the example below. For Python models using <code>mlserving-1.1</code> runtime, JSON format is based on how the predict function is written by the author who exported the model and processing the input. For example, here is a sample request for TensorFlow Inception model: An example request for the inception model is: <pre>{ "signature_name": "predict_images", "inputs": { "images": [{ "b64": "<<Put base64 encoded strings here>>" }] } }</pre>

Input	Type	Description
message	input	It can have the following attributes: <code>mso.model_name</code>: Mandatory when more than one model is deployed. This is to specify the model name for which inference request is invoked. This is optional in case of single model is deployed (either by single model configured in <code>Models</code> or <code>artifactId</code>), then it is obvious that the inference request for that single model deployed and hence not needed to be passed as an input. <code>mso.method</code>: Default value is <code>predict</code>. This is to specify which method to be called to the Model Serving Operator. It can have the values as <code>predict</code> or <code>info</code>. <code>predict</code>: This will invoke the inference request to the model server for the given model name and return that response. <code>info</code>: This will list down the models deployed in the model server and would be useful to see the <code>model_name</code> and <code>artifactId</code> of the models used in this model server deployment.

Output

Output	Type	Description
logs	string	The logs returned by the model deployment process. This port can be connected to a wiretap, for example, so that you can monitor the model deployment progress.

Output	Type	Description
output	message	The response message from the model containing the predictions. For example, here is a sample response from TensorFlow Inception model: <pre> { "outputs": { "classes": [["soap dispenser", "plate rack", "face powder", "comic book", "rubber eraser, rubber, pencil eraser"]], "scores": [[6.88705, 6.11946, 4.94852, 4.74013, 4.69681]] } }</pre>

For more information, see [TensorFlow REST API Documentation](#) .

How to Use the Operator in a Online Inference Pipeline

Procedure

1. Add OpenAPI servlow operator.
 - a. Add an OpenAPI servlow operator in your graph.
 - b. Set `basePath` in the operator configuration, for example, `myinference`. This path will be used in the inference request.
2. **Optional:** (Required only if you are deploying multiple models) Add your custom Preprocessing Operator:
 - a. Add any custom Operator (such as a Python-based operator) that has the logic to preprocess the HTTP request received from the OpenAPI servlow operator.
 - b. Connect the output port `out` of the OpenAPI servlow operator to the `input` port of your custom preprocessing operator.

- c. Connect the `output` port of the preprocessing operator to the `input` port of the Model Serving operator.

The custom operator should have the logic to construct (during inference) the input of message type with attributes `mso.model_name` and `mso.method` with the values based on the received HTTP request `uri` and `http` request method. When these attributes have been constructed, the Model Serving operator is called.

❁ Example

If the HTTP request method is `GET`, the `mso.method` attribute value can be set as `info` to get a list of deployed models.

The model name (for which inference to be made) can be parsed from the HTTP request `uri` path parameter such as `<OpenAPI Servlow URL>\v1\models\<model_name>`, where `<model_name>` is one of the model names configured in the `Models` configuration. This value will have to be set as a value for `mso.model_name`.

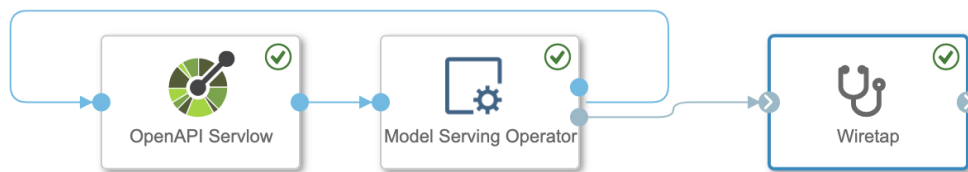
Based on the values of `mso.method` and `mso.model_name`, the following methods are called and return the response to the OpenAPI Servlow operator as response:

- `info` – method to list down the models deployed in this model server.
- `predict` – to call the predict method for the given model name in the model server.

3. Add the Model Serving operator.
 - a. Add an Model Serving operator to your graph.
 - b. Connect the ports:
 - If you are deploying a single model, connect the output port `out` of the OpenAPI Servlow operator to the `input` port of the Model Serving operator.
 - If you are deploying multiple models, connect the output port `out` of the custom preprocessing Operator (as mentioned above) to the `input` port of the Model Serving operator.
 - c. Set the configuration parameters of Model Serving operator.
 - d. Connect the `output` port of the Model Serving operator to the input port `ret` of the OpenAPI Servlow operator.
4. Add Wiretap operator:
 - a. Add a Wiretap operator to your graph.
 - b. Connect the output port `logs` of the Model Serving operator to the input port `in` of the Wiretap operator.
5. Run the graph.

Example

An example of an online inference pipeline is shown below (without preprocessor as mentioned in Step 2).



2.13.7 Submit Metrics

The Submit Metrics operator submits your model metrics to the ML Tracking API. The metrics can be viewed in ML Scenario Manager.

The metrics port accepts inputs as a JSON Object. Ensure that each name or value pair corresponds to a "Metric Name": "Metric Value" pair.

For example, the expected JSON format for the metrics port is as follows:

```
{"Accuracy": "0.88", "Specificity": "0.92"}
```

In this example, when you run your graph, "Accuracy" and "Specificity" values will be posted to the ML tracking API. The metrics will be associated with the "Runtime Handle" of the graph that produced them.

i Note

Consider the following in relation to this operator:

- All values must be strings.
- Your graph will terminate if you do not submit metrics successfully.
- The pipeline must be created via ML Scenario Manager.

Configuration Parameters

None

Input

Input	Type	Description
metrics	message	Accepts a JSON object with metrics that the pipeline submits to the ML Tracking API.

Output

Output	Type	Description
response	message	Outputs a string response if the metrics have been submitted successfully.

Related Information

[ML Scenario Manager](#)

2.13.8 ML Training

Use this operator to run a Machine Learning training script. This operator runs on Python 3.6.x.

Configuration Parameters

Parameter	Type	Description
Artifacts	list	Artifacts (for example, datasets) used as an input. The name acts as an alias to retrieve the artifact from inside the training job script. Use a placeholder for the ID to allow selection of the artifact to be used in the scenario manager UI. The placeholder has the form <code>{ARTIFACT:MODEL:<MODEL_NAME>}</code> (for models), or <code>{ARTIFACT:DATASET:<DATASET_NAME>}</code> (for datasets). You need to replace the <code><MODEL_NAME>/<DATASET_NAME></code> with the to be filtered name.
Environment Variables	list	Environment variables used in the training process. Use a JSON array notation of name, value objects.

Parameter	Type	Description
Script	string	The actual script that defines the machine learning model. Inside the script, the Python module <code>sapdi</code> is provided to access input artifacts and create output artifacts. See the SDK documentation for details.

Input

Input	Type	Description
artifacts	message	An input port is added from the UI for each artifact that you want to pass. The job is triggered when all ingoing artifacts are available. A message specifying an artifact is expected to have an attribute <code>artifact</code> whose object contains an artifact name and an <code>id</code>. For example, <pre>{ 'Attributes': { 'artifact': { 'id': '5438b78c-cbcc-4ac1- aacf-92e52755e9da', 'name': 'train'} }, 'Body': '' }</pre> The artifact can then be accessed in your training script using the <code>sapdi</code> Python module. Note By default, this operator does not support in ports to receive data other than artifacts.

Output

Output	Type	Description
logs	string	The logs returned by the training process. Polling occurs every 20 seconds. This port can be connected, for example, to a terminal.
artifacts	message	You can add custom output ports from the UI. For every such port, an artifact can be created inside the training script using the <code>sapdi</code> Python module. The artifact is sent through the out port after the job has been completed.

i Note

By default, this operator does not support custom out ports of type message to output data other than artifacts.

2.14 SAP Vora

Related Information

[Vora Avro Decoder \[page 457\]](#)
[Vora Avro Encoder \[page 461\]](#)
[Vora Avro Ingestor \[page 462\]](#)
[Vora Client \[page 470\]](#)
[Vora Ingestor \[page 473\]](#)
[Vora Loader \[page 477\]](#)
[Vora Table Consumer \[page 481\]](#)

2.14.1 Vora Avro Decoder

This operator allows you to decode coded (Avro, CSV, and JSON) messages and offer the possibility to process the decoded messages using some operators before passing the data to SAP Vora Ingestor to storing the data in SAP Vora.

For direct ingestion into SAP Vora, use the SAP Vora Avro Ingestor operator instead. An Avro schema is used to describe the structure of the record and it is used to derive the Vora table schema. This Avro schema may be included in the message if it is an Avro message. Otherwise, it can be configured either in this operator's configuration or included in the message itself under attribute "avro.schema".

When using CSV messages, each line is interpreted using the provided Avro schema. When there is no header specified in CSV, the value is directly assigned to the corresponding field (for example, the nth field of CSV is assigned to the nth field of Avro record). When a header is specified, the value is assigned to the field with the corresponding field name.

The following table describes the supported Avro primitive and logical types and how they are translated into Vora SQL types.

Avro	go	js	CSV (sample values)	Vora
boolean	bool	bool	true	BOOLEAN
int	int32	Number	9	INTEGER
long	int64	Number	99	BIGINT
float	float32	Number	99.9	FLOAT
double	float64	Number	99.99	DOUBLE
bytes	string	string	aGVsbG8=	VARCHAR(*)
string	string	string	hello	VARCHAR(*)
decimal(p,s)	string	string	1987.74	DECIMAL(p,s)
date	string	string	2017-08-29	DATE
time-millis	string	string	15:28:50.345	TIME
time-micros	string	string	15:28:50.345678	TIME
timestamp-millis	time.Time	string	2017-08-29 15:28:50.345	TIMESTAMP
timestamp-micros	time.Time	string	2017-08-29 15:28:50.345678	TIMESTAMP
fixed(n)	string	string	68656c6c6f	VARCHAR(2n)

i Note

aGVsbG8= and 68656c6c6f are the base64 and hexadecimal representations of value hello, respectively.

The above Avro to Vora type association may be customized using the extension properties. The following table describes the supported extension properties.

Extension Property	Supported in Avro Types	Property Type	Description
colName	all types	string	Specify the column name.
size	int	Number	Specify the bit size 8, 16, 32, 64 to use TINYINT, SMALL-INT, INTEGER, BIGINT, respectively.
maxLength	string, bytes	Number	Specify the maximal length.
primaryKey	all types	bool	Specify whether the field is primary key.

A record may be nested arbitrarily but must be bounded so that it can be flattened to a fixed table column definition. In the derived table column definition, the column names correspond to the fully qualified field names of the record (for example, each field name in the nested structure is concatenated. For example, a field named bar under its parent field named foo is named as foo_bar.)

When using CSV records, records are represented by rows of CSV lines. The first line may represent the field or column names, commonly known as the header line. In this case, the ordering of the fields do not necessarily match the ordering of the fields defined in the avro schema. Furthermore, some fields may be omitted if there is no value corresponding to those fields. If no header line is present, the ordering of the values must match the ordering of the fields defined in the avro schema.

When using JSON records, records are represented as a JSON array of maps or arrays. The former assumes a map-based representation of a record, where each key value pair in the map is assigned to its corresponding field. The latter assumes the structure similar to CSV, where an array of values is used to represent a record. In this case, the first array may represent the header line. The JSON records may be given as a structured object (for example, as an array of maps or arrays) or in its serialized form.

The body of the decoded message is an array of records, where each record is an array of go lang typed values.

In addition, the decoded message contains the metadata in attribute "vora.record.definition". This metadata has the following properties that describe the types of the decoded message and determine the target database column types.

Property Name	go type	js type	Description
recName	string	string	The record name that is used to derive the table name (for example, "edevicerecord")

Property Name	go type	js type	Description
tableName	string	string	The optional table name (for example, "edevicetable").
partitioningCriteria	string	string	The optional partitioning criteria (for example, "HASH (code) MIN PARTITIONS 4 MAX PARTITIONS 4 INTERVAL (ts) '10' MINUTE")
fieldNames	[]string	string[]	The field names (for example, ["idx", "code", "magnitude", "ts"])
fieldTypes	[]string	string[]	The field types in go lang type (for example, ["int", "string", "double", "timestamp-millis"])
fieldNillables	[]bool	boolean[]	The field nillables (for example, [false, false, false, false])
fieldPrimaryKeys	[]bool	boolean[]	The field primary keys (for example, [true, false, false, true])
colNames	[]string	string[]	The table column names (for example, ["idx", "code", "magnitude", "ts"])
colTypes	[]string	string[]	The table column types (for example, ["INTEGER", "VARCHAR(25)", "DOUBLE", "TIMESTAMP"])

If the body of the decoded message is modified or rearranged, the above metadata must be adjusted to match the modified body.

Configuration Parameters

Parameter	Type	Description
defaultAvroSchema	string	The default Avro schema to be used when the incoming message does not include its schema. This parameter is mandatory for Avro messages without schema, CSV or JSON messages. Default: ""
varLimit	integer	The default size limit for the varchar columns, where 0 indicates unlimited, for example. '*'. Default: 0
format	string	The input message format. The accepted values are "avro", "csv", or "json". Default: "avro"
csvComma	rune	The delimiter character code for the CSV format. For example, 44 for ','; 59 for ';'; 124 for ' '. Default: ','
csvHeaderIncluded	bool	The input CSV message contains the header line. Default: false

Input

Input	Type	Description
in	message	Accepts messages containing Avro, CSV, or JSON messages in the body.

Output

Output	Type	Description
out	message	Messages with record objects in the body and a header <code>vora.record.definition</code> that contains the structure definition of the record.
error	message	If this port is not connected, any processing error will lead to the termination of the operator. If this port is connected, a message with the header property <code>message.response.error</code> with the original body payload is emitted from this port.

2.14.2 Vora Avro Encoder

This operator allows you to create Avro, CSV, or JSON messages from the Vora messages containing record objects. This operator can be used as the inverse operator of the SAP Vora Avro Decoder operator to reconstruct the original messages, or the equivalent messages in another format or in different structures. The schema must be provided as an Avro schema in this operator's configuration or included in the message itself under attribute `"avro.schema"`. This schema is used to assemble the input data into the target output message.

The schema must be provided in this operator's configuration or included in the message itself under attribute `"avro.schema"`. This schema is used to assemble the input data into the target output message.

Configuration Parameters

Parameter	Type	Description
defaultAvroSchema	string	The Avro schema to be used to generate the output message. Default: ""
format	string	The output message format. The accepted values are <code>"avro"</code> , <code>"csv"</code> , or <code>"json"</code> . Default: <code>"avro"</code>

Parameter	Type	Description
csvComma	rune	The delimiter character code for the CSV format. For example, 44 for ','; 59 for ';'; 124 for ' '. Default: ","
csvHeaderIncluded	bool	The input CSV message contains the header line. Default: false

Input

Input	Type	Description
in	message	Accept messages containing record objects in the body.

Output

Output	Type	Description
out	message	Messages whose body content are formatted as Avro, CSV, or JSON messages.
error	message	If this port is not connected, any processing error will lead to the termination of the operator. If this port is connected, a message with the header property message.response.error with the original body payload is emitted from this port.

2.14.3 Vora Avro Ingestor

This operator allows you to dynamically ingest data into SAP Vora based on the incoming Avro or other text messages in CSV and JSON.

An Avro schema is used to describe the structure of the record and the Vora table schema is derived from this schema. More precisely, the target table will be automatically created with the Avro table name and the table

columns derived from the Avro record fields. If the table already exists, its columns must match those Avro record fields.

The Avro schema can be provided in several ways. If the message is an Avro message, the schema may be included in the message. Otherwise, the schema can be as a detached schemas either in this operator's configuration or included in the message itself under attribute "avro.schema". When using a detached schema, the undeclared fields are ignored during arguments building for ingestion.. If the input data needs to be processed before being stored, use SAP Vora Avro Decoder and SAP Vora Ingestor instead.

The following table describes the supported Avro primitive and logical types and how they are translated into Vora SQL types.

Avro	go	js	CSV, js (sample values)	Vora
boolean	bool	bool	true	BOOLEAN
int	int32	Number	9	INTEGER
long	int64	Number	99	BIGINT
float	float32	Number	99.9	REAL
double	float64	Number	99.99	DOUBLE
bytes	string	string	aGVsbG8=	VARCHAR(*)
string	string	string	hello	VARCHAR(*)
decimal(p,s)	string	string	1987.74	DECIMAL(p,s)
date	string	string	2017-08-29	DATE
time-millis	string	string	15:28:50.345	TIME
time-micros	string	string	15:28:50.345678	TIME
timestamp-millis	time.Time	string	2017-08-29 15:28:50.345	TIMESTAMP
timestamp-micros	time.Time	string	2017-08-29 15:28:50.345678	TIMESTAMP
fixed(n)	string	string	68656c6c6f	VARCHAR(2n)

Note

aGVsbG8= and 68656c6c6f are the base64 and hexadecimal representations of value hello, respectively.

i Note

Date, time-millis, time-micros, timestamp-millis, and timestamp-micros values may be represented as an integer based representation based on the Avro's type definition instead of the above string based representation.

Avro	go	js	CSV, js (sample values)
date	int64	Number	17407
time-millis	int64	Number	55730345
time-micros	int64	Number	55730345678
timestamp-millis	int64	Number	1504020530345
timestamp-micros	int64	Number	1504020530345678

The above Avro to Vora type association may be customized using the extension properties. The following table describes the supported extension properties.

Extension Property	Supported in Avro Types		Description
colName	all types	string	Specify the column name.
size	int	Number	Specify the bit size 8, 16, 32, 64 to use TINYINT, SMALL-INT, INTEGER, BIGINT, respectively.
maxLength	string, bytes	Number	Specify the maximal length.
primaryKey	all types	bool	Specify whether the field is primary key.

In addition to the above extension properties applicable to the fields, the following extension properties are applicable to the record level.

Extension Property	Property type	Description
tableName	string	Specify the table name.
partitioningCriteria	string	Specify the partitioning criteria.

A record may be nested arbitrarily but must be bounded so that it can be flattened to a fixed table column definition. In the derived table column definition, the column names correspond to the fully qualified field names of the record (for example, each field name in the nested structure is concatenated. For example, a field named bar under its parent field named foo is named as foo_bar).

When using CSV records, records are represented by rows of CSV lines. The first line may represent the field or column names, commonly known as the header line. In this case, the ordering of the fields do not necessarily

match the ordering of the fields defined in the Avro schema. Furthermore, some fields may be omitted if there is no value corresponding to those fields. If no header line is present, the ordering of the values must match the ordering of the fields defined in the avro schema.

When using JSON records, records are represented as a JSON array of maps or arrays. The former assumes a map-based representation of a record, where each key value pair in the map is assigned to its corresponding field. The latter assumes the structure similar to CSV, where an array of values is used to represent a record. In this case, the first array may represent the header line. The JSON records may be given as a structured object (for example, as an array of maps or arrays) or in its serialized form.

The following sample graphs use this operator.

Configuration Parameters

Name	ID	Type	Description
Connection Type	connectionType	string	Generating CSV messages with datagenerator and directlyThe connection to SAP Vora can be configured directly using dsn or indirectly using connection. Default: "dsn"
DSN	dsn	string	A valid data source name in the format v2 : // host : port / ? binary = true. Make sure that you add / ? binary = true to the end, because only binary transfer is available for the SAP Vora Transaction Coordinator. Default: "v2://localhost:2204/?binary=true"
User	user	string	The user name if the connection is configured using dsn. Default: ""
Password	password	string	The password if the connection is configured using dsn. Default: ""

Name	ID	Type	Description
Connection	connection	object	A valid connection configuration provided by Connection-Manager.
Auto Aggregation	aggregation	bool	Enables the automatic aggregation of records to trigger a series of bulk inserts independently of the number of records contained in each message. This automatic aggregation assumes a single processing sequence and does not support aggregating records from multiple processing sequences. Default: false
Aggregate - Max Bytes	aggregateMaxBytes	int	Limits the maximal size of the aggregated records in bytes under the auto-aggregation mode. Until the aggregated records reach this limit, the records are aggregated. Default: 4194304
Aggregate - Max Records	aggregateMaxRecs	int	Limits the maximal number of the aggregated records under the auto-aggregation mode. Until the aggregated records reach this limit, the records are aggregated. Default: 1000
Aggregate - Max Time	aggregateMaxTime	int	Limits the maximal time in milliseconds to wait until flushing the aggregated records that have not reached the size constraints. Default: 2000
Database Schema	databaseSchema	string	The database schema name. Default: ""

Name	ID	Type	Description
Partition Key Regex	partitionKeyRegex	string	A regular expression to select a sequence of Avro record field names to be bound to the arguments to the specified partition function. Details: When generating the partitioning scheme (the actual binding of arguments to parameters of partition keys) the system deduces the matching of actual columns to parameters of the partition scheme using this regular expression. Example: Let's suppose we have a record with fields <code>col1 VARCHAR(500)</code>, <code>col2 BIGINT</code>, <code>col3 DATE</code>, and a hash partition function "<code>pf(pa, pb)</code>". Then, specifying <code>partitionKeyRegex, .*1 .*3</code> will select <code>col1</code> and <code>col3</code> to be bound to the parameters <code>pa</code> and <code>pb</code>, respectively. Default: ""
Partition Criterion	partitionCriterion	string	A partition function. The use of this parameter is accompanied by the use of parameter <code>partitionKeyRegex</code> to define a partition scheme. Alternatively, a partition scheme can be specified in the provided Avro schema using extension property <code>partitioningCriteria</code>. Default: ""
Table Type	tableType	string	The table type ("STREAMING"). Default: "STREAMING"

Name	ID	Type	Description
Ingestion Mode	ingestionMode	string	The ingestion mode INSERT or UPSERT Default: "INSERT"
Primary Key Regex	primaryKeyRegex	string	A regular expression to select some Avro record field names to be used as the primary keys. The primary keys may be specified in the Avro schema using extension property "primaryKey". This parameter is only useful when the primary keys are not specified in the Avro schema. Default: ""
Default Avro Schema	defaultAvroSchema	string	The default Avro schema to be used when the incoming message does not include its schema. This parameter is mandatory for Avro messages without schema or CSV messages. Default: ""
Variable Limit	varLimit	integer	The default size limit for the varchar columns, where 0 indicates unlimited, for example, '*'. The size limit may be specified in the Avro schema using extension property "maxLength" per record field. Default: 0
Format	format	string	The input message format. The accepted values are "avro" or "csv". Default: "avro"

Name	ID	Type	Description
Use Schema Registry	useSchemaRegistry	bool	The input Avro message containing the schema ID whose corresponding schema is stored at some schema registry. i Note The schema itself needs to be retrieved offline and set to parameter defaultAvroSchema. Default: false
CSV Comma	csvComma	rune	The delimiter character code for the CSV format. For example, 44 for ','; 59 for ';'; 124 for ' '. Default: ','
CSV Header Included	csvHeaderIncluded	bool	The input CSV message contains the header line. Default: false

Input

Input	Type	Description
in	message	Accepts messages containing Avro, CSV, or JSON messages in the body.

Output

Output	Type	Description
out	message	Messages with the header properties that indicate the commit progress. If the input message does not contain a commit token (for example, message header <code>message.commit.token</code>), no output will be generated.
error	message	If this port is not connected, any processing error will lead to the termination of the operator. If this port is connected, a message with the header property <code>message.response.error</code> with the original body payload is emitted from this port.

2.14.4 Vora Client

The Vora Client operator works as a client for the SAP Vora Transaction Coordinator and allows you to run statements on it.

Graph `com.sap.demo.vora.client` demonstrates how to use this operator.

Configuration Parameters

Name	ID	Type	Description
Connection	connection	object	The object containing the connection parameters to a Vora instance.

Name	ID	Type	Description
Terminate on Error	terminateOnError	bool	A flag that indicates whether or not the graph should stop if this operator encounters an error. If set to <code>false</code>, an output may represent an error, in which case its attribute <code>message.error</code> will be <code>true</code> and its body will contain a string describing the error. i Note This setting does not apply to errors during initialization. In such cases, errors will always cause the graph to fail. Default: <code>true</code>
Retry Attempts	numRetryAttempts	int	The number of times to retry an unsuccessful operation before raising an error. Default: <code>5</code>
Retry Period (ms)	retryPeriodInMs	int	The waiting time in milliseconds between consecutive retry attempts. Default: <code>500</code>
Connection Timeout (ms)	connectionTimeoutInMs	int	The number of milliseconds to wait for when connecting to Vora before timing out. Default: <code>60000</code>
Statement Timeout (ms)	statementTimeoutInMs	int	The maximum number of milliseconds each SQL statement is allowed to take. When this limit is reached, the operation is canceled and retried (obeying Retry Attempts). If set to <code>0</code>, no timeout is set. Default: <code>0</code>

Input

Input	Type	Description
sql	message	Accepts SQL code in a message's body to be executed by the SAP Vora Transaction Coordinator. • If the attribute <code>message.request.id</code> is present, it will be preserved on the output. • If a response callback is present: ◦ If result is not connected, the callback will be invoked. ◦ If result is connected, the callback will be forwarded to the output without being invoked.

Output

Output	Type	Description
result	message	A message containing the result of an operation. If the operation retrieved rows, they will be stored in the message body as <code>[]map[string]interface{}</code>, where the keys in each map are the names of each selected column. Otherwise, the body will be <code>null</code>.

See `terminateOnError` for error outputs.

Supported Types

When executing queries, the retrieved data will be output according to the following type equivalence:

Column Type	Output Type
TINYINT, SMALLINT, INTEGER, BIGINT	int64
REAL, DOUBLE	float64

Column Type	Output Type
DECIMAL	string ("x.y")
VARCHAR, CHAR	string
DATE	string ("YYYY-mm-dd")
TIME	string ("HH:MM:SS")
TIMESTAMP	string ("YYYY-mm-dd HH:MM:SS.NNNNNNN")
BOOLEAN	BOOL

2.14.5 Vora Ingestor

This operator allows you to dynamically ingest data into SAP Vora based on incoming records messages. The DB table and its column definitions are determined by the metadata included in the message under attribute "vora.record.definition".

The target table will be automatically created with the information provided in the metadata. If the corresponding table already exists, its columns must match those columns defined in the metadata. This metadata is typically generated by SAP Vora Avro Decoder, which is capable of extracting record fields based on the provided Avro schema from input data in various formats such as Avro, JSON and CSV.

Configuration Parameters

Name	ID	Type	Description
Connection Type	connectionType	string	The connection to SAP Vora can be configured directly using DSN or indirectly using connection. Default: "manual"

Name	ID	Type	Description
DSN	dsn	string	A valid data source name in the format <code>v2://host:port/?binary=true</code>. Make sure that you add <code>/?</code> <code>binary=true</code> to the end, because only binary transfer is available for the SAP Vora Transaction Coordinator. Default: "v2://localhost:2204/?binary=true"
User	user	string	The user name if the connection is configured using dsn. Default: ""
Password	password	string	The password if the connection is configured using dsn. Default: ""
Connection	connection	object	A valid connection configuration provided by Connection-Manager.
Aggregation	aggregation	bool	Enables the automatic aggregation of records to trigger a series of bulk inserts independently of the number of records contained in each record message. This automatic aggregation assumes a single processing sequence and does not support aggregating records from multiple processing sequences. Default: false
Aggregate - Max Bytes	aggregateMaxBytes	int	Limits the maximal size of the aggregated records in bytes under the auto-aggregation mode. Until the aggregated records reach this limit, the records are aggregated. Default: 4194304

Name	ID	Type	Description
Aggregate - Max Records	aggregateMaxRecs	int	Limits the maximal number of the aggregated records under the auto-aggregation mode. Until the aggregated records reach this limit, the records are aggregated. Default: 1000
Aggregate - Max Time	aggregateMaxTime	int	Limits the maximal time in milliseconds to wait until flushing the aggregated records that have not reached the size constraints. Default: 2000
Database Schema	databaseSchema	string	The database schema name. Default: ""
Partition Key Regex	partitionKeyRegex	string	A regular expression to select a sequence of Avro record field names to be bound to the arguments to the specified partition function. Details: When generating the partitioning scheme (the actual binding of arguments to parameters of partition keys) the system deduces the matching of actual columns to parameters of the partition scheme using this regular expression. Example: Suppose we have a record with fields <code>col1 VARCHAR(500)</code>, <code>col2 BIGINT</code>, <code>col3 DATE</code>, and a hash partition function <code>"pf(pa, pb)"</code>. Then, specifying <code>partitionKeyRegex, .*1].*3</code> will select <code>col1</code> and <code>col3</code> to be bound to the parameters <code>pa</code> and <code>pb</code>, respectively. Default: ""

Name	ID	Type	Description
Partition Criterion	partitionCriterion	string	A partition function. The use of this parameter is accompanied by the use of parameter <code>partitionKeyRegex</code> to define a partition scheme. Alternatively, a partition scheme can be specified in the provided Avro schema using extension property <code>partitioningCriteria</code>. Default: false
Table Type	tableType	string	The table type ("STREAMING"). Default: "STREAMING"
Ingestion Mode	ingestionMode	string	The ingestion mode INSERT or UPSERT Default: "INSERT"
Primary Key Regex	primaryKeyRegex	string	A regular expression to select some Avro record field names to be used as the primary keys. The primary keys may be specified in the Avro schema using extension property "primaryKey". This parameter is only useful when the primary keys are not specified in the Avro schema. Default: ""
Quote Identifiers	quoteIdentifiers	bool	Automatically quote each identifier such as column name. If the column name specified using schema property <code>colName</code> and it is already quoted, it has no effect. Default: false

Input

Input	Type	Description
in	message	Accepts record messages to be processed.

Output

Output	Type	Description
out	message	Messages with the header properties that indicate the commit progress. For each input message, there will be one output message. The output message may or may not contain the commit token included in the input message (for example, message header message.commit.token). The presence of a commit token will indicate that all messages up to the message associated with this token have been successfully stored.
error	message	If this port is not connected, any processing error will lead to the termination of the operator. If this port is connected, a message with the header property message.response.error with the original body payload is emitted from this port.

2.14.6 Vora Loader

This operator works as a client for the SAP Vora transaction coordinator and allows you to load data into SAP Vora.

Supported storage services are:

- Local File System (local) - Refers to the file system where the Vora instance is running.
- [System Management Files \(/files\)](#)
- [Microsoft Azure Data Lake \(ADL\)](#)
- [Google Cloud Storage \(GCS\)](#)
- [Hadoop Distributed File System \(HDFS\)](#)

- [Amazon Simple Storage Service \(S3\)](#)
- [Microsoft Azure Storage Blob \(WASB\)](#)
- [WebHDFS](#)

Configuration Parameters

Parameter	Type	Description
Vora Connection	object	The object containing the connection parameters to a Vora instance.
Vora Connection Timeout (ms)	int	The number of milliseconds to wait when connecting to Vora before timing out. Default: 60000
Init Statements	string	SQL code to be executed when the operator is initialized. Default: ""
Table Name	string	Mandatory. The name of the table where the loaded file is inserted. Default: "TEST.T1"

Parameter	Type	Description
File Format	string	The format of the file to be loaded.
		Avro Vora
		booleanBOOLEAN
		intINTEGER
		longBIGINT
		floatREAL/FLOAT
		doubleDOUBLE
		bytesVARCHAR(*)
		stringVARCHAR(*)
		decimal(p,s)DECIMAL(p,s)
		dateDATE
		time-millisTIME
		time-microsTIME
		timestamp-millisTIMESTAMP
		timestamp-mi-crosTIMESTAMP
		fixed(n)VARCHAR(2n)
Default: "csv"		

Parameter	Type	Description
Column Options	array	A JSON array containing attributes that can be added to a specific column for CSV files. It is composed by the following values: <i>Column Name</i> (type string) Mandatory. Name of the column to add attributes. <i>Column Index</i> (type int) Index that will represent the column. <i>Decimal Separator</i> (type string) Character that corresponds to the decimal separator. <i>Thousands Separator</i> (type string) Character that corresponds to thousands separator. <i>Format</i> (type string) Format of dates or datetimes. <i>Null Value</i> (type string) Value that corresponds to null. <i>Escape Character</i> (type string) Character that should be escaped.
Field Delimiter	string	Mandatory. The character used to separate column values in CSV files. Default: ","
Skip	int	The number of lines at the beginning of the CSV file to skip.
String Delimiter	string	The delimiter for strings in CSV files.
Decimal Separator	string	Character that corresponds to the decimal separator in CSV files.
Thousands Separator	string	Character that corresponds to thousands separator in CSV files.
Format	string	Format of dates or datetimes of the datasource in CSV files.
Null Value	string	Value that corresponds to null on the datasource in CSV files.
Escape Character	string	Character that should be escaped in CSV files.

Parameter	Type	Description
Storage Service	string	The service to load files from. Additional parameters may depend on the selected service. Local file system refers to the file system where the Vora instance is running. Default: "local"
Proxy	string	The proxy from storage. Currently only supported on WebHDFS, S3, WASB and GCS connections. Default: "noprox"

Input

Input	Type	Description
filename	string	The storage path of the file to be loaded to the transaction coordinator.

Output

Output	Type	Description
result	string	An object containing any data if valid and error, for example: <pre>{"Data":null,"ErrorMessage":"","Success":true}.</pre>

2.14.7 Vora Table Consumer

This operator allows to read from a Vora table or view. It uses Flowagent subengine to connect to a Vora table via SAP Data Services. To effectively read from it, it is necessary to connect this operator to a Flowagent-based

producer operator (for example, Flowagent CSV Producer). When reading from a Vora table, fetch size can be specified.

Configuration parameters

Parameter	Type	Description
Vora Connection	object	The object containing the connection parameters to a Vora database.
HANA Wired Port	string	The HANA wired port used to connect to Vora instance.
Source Table	string	Vora source Table.
Additional Session Parameters	string	Session parameters that will affect the connection with the source.
Fetch Size	integer	Indicates the number of rows that will be fetched from the source in each request. For wide tables, smaller values are better; while for tables with few columns, larger values are better. Default: 100

Input

Input	Type	Description
inTableName	string	Table name from which the data will be read, usually in <Schema>.<TableName> format. i Note This is optional. If <code>Source Table</code> is presented, this port is ignored.

Output

Output	Type	Description
outConfig	string.flowagent.config	Flowagent type port, which only Flowagent producers can consume (for example, Flowagent File Producer). To get the data as string, Flowagent CSV Producer can be used as producer.
outError	string	If mapped, the operator errors are sent to this port and the graph continues running. If not mapped, the graph terminates with the operator error.

2.15 Streaming Analytics

Related Information

[Streaming Analytics Projects \[page 483\]](#)

2.15.1 Streaming Analytics Projects

This operator represents a streaming analytics project. The operator is based on streaming lite, which is a compact version of SAP HANA streaming analytics. As such, each instance of the operator starts an instance of the streaming lite server.

This streaming lite instance does not depend on SAP HANA in any way. If you wish to read from or write data to an SAP HANA database you can do so by adding the Database Input or Output adapter into your streaming analytics project.

Streaming analytics is ideally suited for situations where data arrives as events happen, and where there is value in collecting, understanding, and acting on this data right away. Some examples of data sources that produce streams of events in real time include:

- Sensors
- Smart devices
- Web sites (click streams)
- IT systems (logs)

- Financial markets (prices)
- Social media

Data flows into streaming projects from various sources, such as adapters or operator ports that connect the sources to the streaming lite server. The streaming projects contain business logic, which they apply to the incoming data, typically in the form of continuous queries and rules. These streaming projects are entirely event-driven, turning the raw input streams into one or more derived streams that can be captured in a database, sent as alerts, posted to downstream applications, or streamed to live dashboards.

At its most basic level, a project consists of streams, windows, and adapters.

- Adapters connect a stream or window to a data source or destination.
- A stream processes incoming events without retaining and storing data, and produces output events according to an applied continuous query.
- A window receives data but can also retain and store data. Incoming events can add, update, or delete rows in the window's table.

You can also define one or more input and output operator ports in your project to receive and send data.

Configuration Parameters

Name	ID	Type	Description
script	script	string	Contains Continuous Computation Language (CCL), which specifies the functionality of the project. CCL is the primary event processing language of streaming analytics and is based on Structured Query Language (SQL). Default: "file://script.ccl"
ccx	ccx	string	Already compiled CCL (CCX) that is used to start the project. If this value is empty, the script property must have a value specified as that is then compiled into CCX and used to start the project. If a value is provided, the script property value is ignored and the compiled CCX is used to start the project. Default: ""

Name	ID	Type	Description
ccr	ccr	string	Optional runtime project configuration (CCR) file that governs specific runtime properties of a project. Default: ""
command port	command port	string	If specified, this port number is used by the project to listen for commands. If the value is 0 or left empty, a random free port number is used instead. Default: ""

Name	ID	Type	Description
input format	input format	string	The format used to parse rows received on input ports. CSV: Incoming records need to be in comma-separated values (CSV) format and follow the RFC4180 standard. Headers are not supported. Multiple records can be batched together. JSON: Input can be only a JSON array. Each of its elements can be either: • A JSON array representing a data row where its elements are in the same order as the schema of the target input stream or window in the project CCL. If the input has opcodes parameter is set to true, the first array element must contain an opcode character. • A JSON object whose keys are the column names in the schema of the target input stream or window in the project CCL. Spare keys are ignored. If the input has opcodes parameter is set to true, the JSON object must contain an <code>__opcode</code> field with an opcode character value specified. Default: "CSV"

Name	ID	Type	Description
output format	output format	string	The format used to build rows that are sent on output ports. CSV: Outgoing records are in CSV format and follow the RFC4180 standard. Headers are not included and multiple records are batched together according to the values specified in the output batch size and output batch timeout parameters. JSON: Output is a JSON array that forms a batch of records. The size is based on the values specified in the output batch size and output batch timeout parameters. Each array element is either a JSON object or an array depending on the value of the JSON record type parameter. Default: "CSV"

Name	ID	Type	Description
JSON record type	json record type	string	The type of JSON record to output for each outgoing data row. <i>Object</i>: Each outgoing record is a JSON object whose keys are the column names in the schema of the source output stream or window in the project CCL. If the add opcodes to output parameter is set to true, the JSON object will contain an __opcode field with an opcode character value specified. <i>Array</i>: Each outgoing record is a JSON array representing a data row. Its elements are in the same order as the schema of the source output stream or window in the project CCL. If the add opcodes to output parameter is set to true, the first array element contains an opcode character. Default: "Object"

Name	ID	Type	Description
input has opcodes	input has opcodes	boolean	Specifies whether incoming data rows contain an opcode character. If set to false, the insert opcode value is used. If set to true, each incoming data row will include one of the following opcodes: <i>i</i> (insert): Insert the record. Throw an error if a record with the same key already exists. <i>u</i> (update): Update an existing record. Throw an error if a record with the specified key does not exist. <i>d</i> (delete): Delete a record. Throw an error if a record with the specified key does not exist. <i>p</i> (upsert): Insert the record if the key does not exist, update the record otherwise. <i>s</i> (safedelelete): Delete the record if it exists, do nothing if it doesn't exist. Default: "true"
add opcodes to output	add opcodes to output	boolean	Specifies whether the outgoing data rows contain an opcode character. Default: "true"
output batch size	output batch size	string	Specifies how many outgoing rows, per output port, should be included in a batch. If the value is empty or set to less than 2 rows, each outgoing row is sent individually. Default: ""

Name	ID	Type	Description
output batch timeout	output batch timeout	string	Specifies how many milliseconds to collect output rows for a batch before sending out the current batch and starting a new one. If the value is left empty or is less than 10 milliseconds, the default timeout of 10000 milliseconds (10 seconds) is used. That is, the shortest duration you can specify is 10 milliseconds. Default: ""
stats frequency	stats frequency	string	Specifies how often, in seconds, to print out an INFO level message with message flow statistics. For example, if set to 10 seconds, an INFO level message will print out every 10 seconds. Default: ""

Name	ID	Type	Description
hana services	hana services	array	An array of objects containing connection parameters for SAP HANA databases that can be used as data services. To be able to access the data services from the project CCL, the name of each object needs to match the corresponding service property in the project CCL. All array elements can be either: • Manual: All required SAP HANA connection properties need to be specified in the properties editor. • Configuration Manager: Existing connections that are type HANA_DB and defined in Connection Management can be selected from a dropdown list. Default: "[]"

Name	ID	Type	Description
odbc services	odbc services	array	An array of objects containing connection parameters for generic ODBC data sources such as databases. They can be used as ODBC data services in the project CCL. To be able to access them from the project CCL, the name of each object needs to match the corresponding service property in the project CCL. Each object contains reference to an ODBC DSN that needs to exist in the runtime environment. You can use <code>com.sap.streaming Dockerfile</code> to generate such DSN entries, as shown in an example in the bottom section of the Dockerfile. Default: <code>[]</code>
services file	services file	string	The location and name of the file generated by the operator, which contains all the data service entries available for use in the project CCL. Specifying this value may be useful for debugging issues with data service connectivity. If the value is left empty, the temporary file is used instead. Default: <code>""</code>
log4j properties	log4j properties	string	Specifies the log4j tracing properties that are used by the operator. By default, it includes output to console as well as the <code>to logs/Flow-Agent.log</code> file at INFO trace level. You can change these settings individually. Default: <code>""</code>

Input

Any input port names must match an input stream/window in the project CCL. The operator publishes any incoming data rows to that input stream/window. Only string and message type ports are supported.

Output

Output	Type	Description
<code>outError</code>	string	If mapped, the operator errors are sent to this port and the graph continues running. If not mapped, the graph terminates with the operator error.
<code>committedMessage</code>	message	Notifies about commits of input messages. If mapped, for any input message that has a value specified for the <code>message.commit.token</code> header attribute, the operator will emit an output message with an empty body and all header attributes from the original input message preserved. Before emitting this output message, the operator will publish and commit all input rows contained in the corresponding input message.

Any other output port names must match an output stream or window in the project CCL. The operator subscribes to that output stream or window for any outgoing data rows. Only string and message type ports are supported.

Functional Restrictions

As mentioned earlier, the Streaming Analytics operator is based on streaming lite which is a compact version of SAP HANA streaming analytics. The operator supports the following:

Component	Name
Adapters	• All adapters built using the adapter toolkit. However, you can use them only in managed mode, which means the adapter starts and stops with the streaming analytics project. To learn which adapters are based on the toolkit framework, see Adapter Summary. • The Streaming Web Output adapter is also supported. See Streaming Web Output Adapter. • Database Input and Output adapters. See Database Adapter.

Do not use the binaries or SDKs that come with the streaming install image to communicate with and send data to and from the streaming project. Instead, use the input and output ports on the operator, as well as any of the supported adapters.

There is additional functionality and components that the operator does not support. These are based on the restrictions for streaming lite. For complete details, see the [Streaming Lite](#) section. Keep these in mind as you use the streaming analytics operator and graphs.

For more information on building a streaming analytics project, see the *SAP HANA Streaming Analytics: Developer Guide* and *SAP HANA Streaming Analytics: CCL Reference Guide*.

Related Information

[SAP HANA Streaming Analytics: Developer Guide](#)
[SAP HANA Streaming Analytics: CCL Reference Guide](#)

2.16 Structured Data Operators

You can model a flow and build a transformation pipeline that will move data from the source(s) to the target(s) while transforming the data in the process without having to create custom operators.

Related Information

[Data Transform \[page 495\]](#)
[SAP Application Consumer \[page 496\]](#)
[SAP Application Producer \[page 497\]](#)
[SQL Consumer \[page 498\]](#)
[Structured File Consumer \[page 501\]](#)
[Structured File Producer \[page 503\]](#)

[Table Consumer \[page 507\]](#)
[Table Producer \[page 511\]](#)

2.16.1 Data Transform

The Data Transform operator in Modeler provides wide variety of options available to meet your data transformation needs.

This operator allows you to perform data transformations, such as projection, filter, column operations, and join. It supports to be connected with other operators from Structured Data Operators category as sources and targets.

Configuration Parameters

Parameter	Type	Description
definition	object	The definition of a transformation pipeline.

i Note

- The configuration parameters for the Data Transform operator is node-specific. For more information on how to configure individual nodes in the operator, see [Data Transform](#).
- Any unconnected output port results in a graph failure.

Input

None

Output

None

! Restriction

The Data Transform operator does not work if it is connected to another Flowagent Data Transform operator or any operator that is not from the Structured Data Operators category.

2.16.2 SAP Application Consumer

Use the SAP Application Consumer operator to consume data from SAP and non-SAP sources as modeled in a vFlow graph.

The supported connections include:

- ABAP Legacy
- CDI
- OData
- OpenConnectors

Configuration Parameters

Parameter	Type	Description
Application Type	String	The application from where to consume tabular data.
Source Connection	Object	Connection ID of the source system.
Source Object	Object	Remote dataset of the selected source object.
Trace level	String	Trace level of the extraction.
Default (N)VARCHAR Size	String	The default size of a varchar. The valid range is 0–5000.
Batch Query	Boolean	Determines whether the query is performed in batches. When true, the query to the OData source is performed batch by batch. Each batch is the size specified in Fetch Size .
Fetch Size	Integer	Indicates the number of rows fetched from the source in each request. For wide tables, smaller values are better. For tables with fewer columns, larger values are better.
Access Method	String	Open Connectors API used for retrieving data. In GET mode, Open Connectors' data query API is called directly once per page to retrieve data. In BULK method, an asynchronous query job is created at Open Connectors and then starts streaming data.

Input

Parameter	Type	Description
inTrigger	Message	Trigger to indicate that the operator should start its execution.

i Note
The input port is optional. If the input port does not have a connection, then the operator starts when the graph starts running.

Output

Parameter	Type	Description
outTable	Table	Table type port, which can be used to connect to other structured operators.

2.16.3 SAP Application Producer

Use the SAP Application Producer to write data to SAP and non-SAP sources as modeled in a graph.

This operator reads from any structured operators, and produces a table in the specified target like OData and BW.

Supported connections include:

- OData
- SAP BW

i Note

For BW Datastore Targets:

- BW only supports Advanced Datastore objects as a target under the /DATASTORE folder.
- For BW DataStore objects as target, the option [Write Interface-Enabled](#) should be checked in the BW system (under DataStore Object-> Modeling Properties -> Special Properties -> [] Write Interface-Enabled).
- To preview BW Datastore data within the operator, an ABAP connection needs to be created in the Connection Management application that is associated with the specified BW Connection. That ABAP connection then needs to be specified for the [ABAP Connection ID](#) field for the BW Connection.

Configuration Parameters

Parameter	Type	Description
Service	String	The application from where to consume tabular data.
Connection ID	Object	Connection ID of the target system.
Target Object	Object	Remote dataset of the selected target object.
Batch Size	Integer	Indicates the number of rows committed to the target in each request. For wide tables, smaller values are better. For tables with fewer columns, larger values are better.

Input

Parameter	Type	Description
inTable	Table	Table type port, which comes as output from other structured operators.

Output

Parameter	Type	Description
outMessage	Message	Outputs a message object with the following definition: • Body: (String) The fully qualified name of the target table. • Attributes: ◦ producer.totalRows: (String) Number of rows written to the target.

BW Advanced Datastore Activation

When the SAP Application Producer finishes writing to the Advanced Datastore object, the data needs to be activated on the BW system before it can be used. This can be done on the BW system by creating a BW Process Chain. Configure the BW process chain to activate the data in the Datastore object referenced in the SAP Application Producer. After the Process Chain is configured, saved, and activated, use the BW Process Chain operator in a graph to invoke the process chain and activate the data via a pipeline graph.

To load the data to BW and then activate it via a Process Chain, create a graph that includes a pipeline operator that executes a graph that writes the data to the BW DataStore. Then chain that graph to a pipeline operator that executes the graph to call the Process Chain that activates the data.

2.16.4 SQL Consumer

SQL Consumer operator reads from a specified database using native SQL. This operator produces a structured output, and you need to connect it to other operators from Structured Data Operators category.

The supported databases are:

- Azure SQL DB (AZURE_SQL_DB)
- DB2 Database (DB2)
- Google BigQuery SQL (GBQ)
- HANA (HANA_DB)
- Microsoft SQL Server (MSSQL)
- MySQL Database (MYSQL)
- Oracle Database OSS (ORACLE)

- Redshift Database (REDSHIFT)
- SAP IQ Database (SAP_IQ)

i Note

This operator produces runtime metrics. See [Operator Metrics](#)

Configuration Parameters

Parameter	Type	Description
Database type	string	Database on which the SQL statements are executed.
Azure SQL Database Connection	object	Object containing the connection parameters to an Azure SQL Database.
DB2 Connection	object	Object containing the connection parameters to a DB2 database.
Google BigQuery Connection	object	Object containing the connection parameters to a Google BigQuery database.
HANA Connection	object	Object containing the connection parameters to a HANA database.
MSSQL Connection	object	Object containing the connection parameters to a MSSQL database.
MySQL Connection	object	Object containing the connection parameters to a MySQL database.
Oracle Connection	object	Object containing the connection parameters to an Oracle database.
Redshift Connection	object	Object containing the connection parameters to a Redshift database.
SAP IQ Connection	object	Object containing the connection parameters to a SAP IQ database.
Azure SQL Database Native SQL	string	SQL statement used to read from the source. The syntax depends on the underlying source.
DB2 Native SQL	string	SQL statement used to read from the source. The syntax depends on the underlying source.
Google BigQuery Native SQL	string	SQL statement used to read from the source. The syntax depends on the underlying source.
HANA Native SQL	string	SQL statement used to read from the source. The syntax depends on the underlying source.

Parameter	Type	Description
MSSQL Native SQL	string	SQL statement used to read from the source. The syntax depends on the underlying source.
MySQL Native SQL	string	SQL statement used to read from the source. The syntax depends on the underlying source.
Oracle Native SQL	string	SQL statement used to read from the source. The syntax depends on the underlying source.
Redshift Native SQL	string	SQL statement used to read from the source. The syntax depends on the underlying source.
SAP IQ Native SQL	string	SQL statement used to read from the source. The syntax depends on the underlying source.
Additional session parameters	string	Set session parameters that affect the connection with the source.
Fetch Size	integer	Indicates the number of rows fetched from the source in each request. For wide tables, smaller values are better. For tables with less columns, larger values are better.

Input

Input	Type	Description
inTrigger	message	Trigger to indicate that the operator should start the execution.

i Note

The input port is optional. If the input port does not have a connection, then the operator starts when the graph starts running.

Output

Output	Type	Description
outTable	table	Port of type Table, that can be used to connect to other Structured Data Operators.

2.16.5 Structured File Consumer

Structured File Consumer operator reads from any supported cloud storage. The operator produces structured output, and you need to connect it to other operators from Structured Data Operators category.

The supported databases are:

- Azure Data Lake (ADL)
- Azure Data Lake V2 (ADL_V2)
- Google Cloud Storage (GCS)
- HDFS
- Amazon S3 (S3)
- Alibaba OSS (OSS)
- SDL
- Windows Azure Storage Blob (WASB)

i Note

This operator produces runtime metrics. See [Operator Metrics](#)

Configuration Parameters

Parameter	Type	Description
Storage type (Mandatory)	string	The storage from which the file is read.

Enter the required connection information based on the selected storage type.

Storage Type	Type	Description
ADL Connection	Object	Object containing the connection parameters to an ADL remote.
ADL V2 Connection	Object	Object containing the connection parameters to an ADL V2 remote.
GCS Connection	Object	Object containing the connection parameters to a GCS remote.
HDFS Connection	Object	Object containing the connection parameters to an HDFS remote.

Storage Type	Type	Description
OSS Connection	Object	Object containing the connection parameters to an Alibaba OSS remote.
S3 Connection	Object	Object containing the connection parameters to an S3 remote.
SDL Connection	Object	Object containing the connection parameters to an SDL remote.
WASB Connection	Object	Object containing the connection parameters to a WASB remote.

Enter the required source file information based on the selected storage type.

Parameter	Type	Description
ADLsource file	Object	File from where the content is read. The file must be selected via the browsing action.
ADL V2 source file	Object	File from where the content is read. The file must be selected via the browsing action.
GCS source file	Object	File from where the content is read. The file must be selected via the browsing action.
HDFS source file	Object	File from where the content is read. The file must be selected via the browsing action.
OSS source file	Object	Full path to the file that is read in the storage.
S3 source file	Object	File from where the content is read. The file must be selected via the browsing action.
SDL source file	String	File from where the content is read. The file must be selected via the browsing action.
WASB source file	Object	File from where the content is read. The file must be selected via the browsing action.
Fail on string truncation	Boolean	Indicates whether the reader should exit and fail if a string truncation has happened. This behavior happens when the length of a string in the metadata is smaller than the actual data in the source file.
Fetch size	Integer	Indicates the number of rows fetched from the source in each request. For large files, smaller values are better. For files with less columns, larger values are better.

Remote Source Description Limitation

When you select a file to be read via the browsing UI, the UI obtains the metadata definition for the file. There are three major limitations with this approach:

1. CSV files do not have a metadata definition, so the application infers the metadata by looking at the file.
2. Parquet and ORC files have a metadata definition, but they do not specify the length of strings, which default to 5000.
3. For 'local' sources, browsing and selection is not supported and the metadata definition is not generated.

Given these facts, the metadata can be wrong, and the job may fail. For strings, the default is to fail the job on strings truncation. However, the operator can be configured to tolerate string truncations by setting the **Fail on**

string truncation property to **false**. For 'local' storage type, the behavior is always false and the string data is truncated.

Input

Input	Type	Description
inTrigger	message	Trigger to indicate that the operator should start the execution.

i Note
The input port is optional. If the input port does not have a connection, then the operator starts when the graph starts running.

Output

Output	Type	Description
outTable	table	Port of type Table, that can be used to connect to other Structured Data Operators.

2.16.6 Structured File Producer

Structured File Producer operator reads from any structured data operators and produces a file (CSV, ORC, or PARQUET) in the specified storage.

The supported cloud storages are:

- Azure Data Lake (ADL)
- Azure Data Lake V2 (ADL_V2)
- Google Cloud Storage (GCS)
- HDFS
- Amazon S3 (S3)
- Alibaba OSS (OSS)
- SDL
- Windows Azure Storage Blob (WASB)

i Note

This operator produces runtime metrics. See [Operator Metrics](#)

Configuration Parameters

Parameter	Type	Description
Storage type (Optional)	string	The storage in which the file is written.

Enter the required connection information based on the selected storage type.

Storage Type	Type	Description
ADL Connection	Object	Object containing the connection parameters to an ADL remote.
ADL V2 Connection	Object	Object containing the connection parameters to an ADL V2 remote.
GCS Connection	Object	Object containing the connection parameters to a GCS remote.
HDFS Connection	Object	Object containing the connection parameters to an HDFS remote.
OSS Connection	Object	Object containing the connection parameters to an Alibaba OSS remote.
S3 Connection	Object	Object containing the connection parameters to an S3 remote.
SDL Connection	Object	Object containing the connection parameters to an SDL remote.
WASB Connection	Object	Object containing the connection parameters to a WASB remote.

Enter the required source file information based on the selected storage type.

Parameter	Type	Description
Local file name	String	A formatted string describing the output path and file name. If the root directory of the file is not provided, /vrep/ is used as default root directory of the file.
ADL file name	Object	File to be saved on ADL.
ADL V2 file name	Object	File to be saved on ADL V2.
GCS file name	Object	File to be saved on GCS.
HDFS file name	Object	File to be saved on HDFS.
OSS file name	Object	Location where the file is saved on OSS.
S3 file name	Object	File to be saved on S3.
SDL file name	Object	File to be saved on SDL.
WASB file name	Object	File to be saved on WASB.
Mode	String	Defines the write mode for CSV files. Options are Overwrite and Append. Overwrite replaces the existing content, while Append adds data to an existing file. In both cases, if the file does not exist yet, it's

Parameter	Type	Description
		created during the writing process. Important: GCS, S3, and WASB do not allow append.
Format	String	Output file format: CSV, PARQUET, and ORC are supported for the cloud storage. The CSV file conforms to the RFC-4180 specification, where the row delimiter is always CRLF and the escape character is always double-quotes.
CSV Properties	Object	Properties of the CSV output. - Column delimiter: (string) The field separator. These characters are supported: , ; : TAB. - Header included: (boolean) Indicates whether each CSV output has a header as the first line.
Additional Properties	Object	Additional properties related to the upload options to cloud storage. It's only applicable when a cloud storage service is selected. Compression type: (string) Compression type used when uploading file into cloud service.
Batch size	Integer	Indicates the number of rows that are inserted in a single commit.
Write part files	Boolean	Defines whether the writer produces a single file or a directory of files.
Limit part file records	Boolean	Defines whether to limit individual partition files with a maximum number of records.
Max part file records	Integer	Indicates the maximum number of records in each partition file. This setting guarantees that partition files do not exceed this number.
Part file write mode	String	Defines the write mode for partition files. Options are Overwrite and Append. Overwrite deletes the existing directory if it was a partition file directory and re-create it. Append adds new files to an existing partition file directory. Important: If appending, the producer must match the existing format of the partition file directory, including the format and column layout.

Placeholders

For file names, we support the following placeholders:

- <sourceTableName>: When connected to a table consumer, this placeholder is replaced by the table name of the source. For SQL consumers, this placeholder is replaced by SQL_Table.

Note

Unsupported characters (anything besides alphanumeric characters, a hyphen, and a period) are replaced with an underscore.

- <counter>: An incremental integer for each file written, starting with 1.
- <date>: The date at the moment of writing, in the format yyyyMMdd.
- <time>: The time at the moment of the writing, in the format HHmmss.
- <timestamp>: The timestamp at the moment of the writing, in the format yyyyMMdd_HHmmss.

- <partitionName>: The partition number if the consumer has partitions. For example, 0, 1, and so on. Important: If the consumer has partitions, and the number is not specified, the partition number is always appended to the file name to avoid concurrency issues during writing.

Remote File Name Conventions

Each cloud provider has its own naming convention for defining bucket and object names. Follow these conventions to avoid any source-related issue.

Even though operators accept most special characters, we recommend that only alphanumeric characters are used. Using alphanumeric characters avoids any undesired issues because all sources work well with such characters. Furthermore, some special characters such as ", +, and, are not allowed by the Flowagent File Producer operator and should not be used.

Unknown Data Types

When a column has an unknown data type, it is skipped during processing and is not reflected in the file name. To avoid this behavior, it's necessary to override the unknown data type in the adapted dataset of the consumer. See the example, where \${user-defined-length} needs to be replaced with the maximum length of the COL0 in the dataset:

Original metadata:

Sample Code

```
"adapted_dataset": {
  "schema": {
    "tableBasedRepresentation": {
      "attributes": [
        {
          "name": "COL0",
          "datatype": "UNKNOWN",
          "nativeDatatype": "XML"
        }
      ]
    }
  }
}
```

Modified metadata:

Sample Code

```
"adapted_dataset": {
  "schema": {
    "tableBasedRepresentation": {
      "attributes": [
        {
          "name": "COL0",
          "datatype": "STRING",
          "length": "${user-defined-length}",
          "nativeDatatype": "XML"
        }
      ]
    }
  }
}
```

```

    }
  ]
}
}

```

Input

Input	Type	Description
inTable	table	Table type port, which comes as output from other structured operators.

Output

Output	Type	Description
outMessage	message	Outputs a message object with the following definition: - Body: (string) The fully qualified name of the target table. - Attributes: - producer.totalRows: (string) Number of rows written to the target. - producer.type: (string) Type of the producer that generated the message, which can be one of the following depending on which operator generated the message: CSV for Flowagent CSV Producer, File for Flowagent File Producer, or Table for Flowagent Table Producer. - producer.subType: (string) Subtype of the object that was generated in the target, which can be one of the following: TABLE, CSV, PARQUET, or ORC depending on the producer settings. - producer.partition.count: (integer) Number of partitions from the source. - producer.partition.index: (integer) The partition index that was processed. The index ranges from 0 to partition.count. - Encoding: (string)

2.16.7 Table Consumer

Table Consumer operator reads from an specified database table or view. The operator produces a structured output, and you need to connect it to other operators from Structured Data Operators category.

The supported databases are:

- Azure SQL DB (AZURE_SQL_DB)

- DB2 Database (DB2)
- Google BigQuery (GCP_BIGQUERY)
- SAP HANA Database (HANA)
- Microsoft SQL Server (MSSQL)
- MySQL Database (MYSQL)
- Oracle Database OSS (ORACLE)
- Redshift Database (REDSHIFT)
- SAP VORA Database (VORA)
- SAP IQ Database (SAP_IQ)

i Note

This operator produces runtime metrics. See [Operator Metrics](#)

Configuration Parameters

Parameter	Type	Description
Database type	string	Database where the table is read.
Azure SQL Database Connection	object	Object containing the connection parameters to an Azure SQL Database.
DB2 Connection	object	Object containing the connection parameters to a DB2 database.
Google BigQuery Connection	object	Object containing the connection parameters to a Google BigQuery database. This feature requires configuring the Google BigQuery ODBC driver. For more information, see Connection Management documentation.
HANA Connection	object	Object containing the connection parameters to a HANA database.
MSSQL Connection	object	Object containing the connection parameters to a MSSQL database.
MySQL Connection	object	Object containing the connection parameters to a MySQL database.
Oracle Connection	object	Object containing the connection parameters to an Oracle database.
Redshift Connection	object	Object containing the connection parameters to a Redshift database.
SAP IQ Connection	object	Object containing the connection parameters to a SAP IQ database.
Vora Connection	object	Object containing the connection parameters to a SAP IQ database.

Parameter	Type	Description
Azure SQL Database Source Table	object	Table from where the content is read. The table must be selected via the browsing action.
DB2 Source Table	object	Table from where the content is read. The table must be selected via the browsing action.
Google BigQuery Source Table	object	Table from where the content is read. The table must be selected via the browsing action.
HANA Source Table	object	Table from where the content is read. The table must be selected via the browsing action.
MSSQL Source Table	object	Table from where the content is read. The table must be selected via the browsing action.
MySQL Source Table	object	Table from where the content is read. The table must be selected via the browsing action.
Oracle Source Table	object	Table from where the content is read. The table must be selected via the browsing action.
Redshift Source Table	object	Table from where the content is read. The table must be selected via the browsing action.
SAP IQ Source Table	object	Table from where the content is read. The table must be selected via the browsing action.
Vora Source Table	object	Table from where the content is read. The table must be selected via the browsing action.
Additional session parameters	string	Set session parameters that affect the connection with the source.
Fetch Size	integer	Indicates the number of rows fetched from the source in each request. For wide tables, smaller values are better. For tables with less columns, larger values are better.

If Vora is selected, the following property is available:

Parameter	Type	Description
HANA Wire port	string	HANA Wire port of the transaction coordinator, which is determined as 3XX15, where XX is the instance number of the Vora cluster as configured in the Kubernetes cluster manager. For example, when the instance number of the Vora cluster is configured as 01, the HANA Wire port is 30115. A system administrator with the kube-config permissions for the cluster can obtain the HANA Wire port by using the following command: <pre>kubectl describe deployment vora-tx-coordinator -n=YourNamespace grep tc_hana_wire_port</pre> Replace <code>YourNamespace</code> with your namespace, which you can get by running the following command: <pre>kubectl get namespaces</pre>

Input

Input	Type	Description
inTrigger	message	Trigger to indicate that the operator should start the execution. Note The input port is optional. If the input port does not have a connection, then the operator starts when the graph starts running.

Output

Output	Type	Description
outTable	table	Port of type Table, that can be used to connect to other Structured Data Operators.

2.16.8 Table Producer

Table Producer operator reads from any structured operators, and produces a table in the specified database.

The supported databases are:

- HANA
- SAP IQ
- Vora

i Note

This operator produces runtime metrics. See [Operator Metrics](#)

Configuration Parameters

Parameter	Type	Description
Database type (Optional)	string	The target database to operate.

Input

Input	Type	Description
inTable	table	Port of type Table, that comes as output from other structured data operators.

Output

Output	Type	Description
outMessage	message	Outputs a message object.

2.17 Tables

Related Information

[Decode Table \[page 512\]](#)

[Generate Table \[page 516\]](#)

2.17.1 Decode Table

Decodes tabular data from one of the supported formats into table messages. Currently, only CSV input is supported.

Configuration Parameters

Name	ID	Type	Description
Input Format	format	string	Mandatory. The input format. Currently, only CSV is supported. Default: ""

Name	ID	Type	Description
CSV Mode	csvMode	string	When using CSV format, this specifies how the operator should interpret the incoming data. Accepted values: • Batch: each chunk from the input will be parsed as a self-contained CSV string. The records will be inserted before reading the next input from data. • Stream: each chunk is considered as the next piece of a single stream, so each individual input does not need to be complete. Once CSV stream bulk size records are buffered (or CSV stream bulk timeout goes by without any new records), they are output in a single message. When the graph stops, accumulated records are discarded. Default: "Batch"
CSV Stream Bulk Size	csvStreamBulkSize	int	How many CSV records should be accumulated to be output in a single message. The value zero makes the bulk unlimited in size, so that records are output only every time the timeout is reached. Default: 100

Name	ID	Type	Description
CSV Stream Bulk Timeout	csvStreamBulkTimeout	string	When in CSV stream mode, this option forces the operator to output all accumulated records, even if there are less than the bulk size (CSV stream bulk size). The value zero disables the timeout, so records are flushed only when the bulk size is reached. Default: "10s"
CSV Field Delimiter	csvFieldDelimiter	string	The character used as field delimiter in CSV input. Default: ","
CSV Comment Character	csvComment	string	The character used as comment indicator. When this configuration is non-empty, lines starting with this character will be ignored. Default: ""
CSV Lazy Quotes	csvLazyQuotes	bool	If true, allows fields to contain unescaped quotes. With this option, only quotes at the start of a field need to be escaped. Default: true

Name	ID	Type	Description
CSV Header	csvHeader	string	Determines whether CSV will contain a header and how to use it. The header will be regarded as being the first record of each input (in batch mode) or of the entire lifetime (in stream mode). Accepted values: • None: the first record is regular data, not a header. • Ignore: the first record is a header that should be discarded. • Column names: the first record is a header from which to extract column names. These names will be stored in Table Messages's standard <code>table.columns</code> attribute. Default: "None"

Input

Input	Type	Description
input	message	A message whose body will be decode according to the input format configuration.

Output

Output	Type	Description
output	message.table	A table message holding the data decoded from the input. The original input message attributes will be preserved in the output, except when in CSV stream mode, because output is decoupled from input.

For more information, see [Table Messages](#).

2.17.2 Generate Table

This operator can generate Table Messages with a user-defined schema and random data.

Configuration Parameters

Name	ID	Type	Description
Configuration Mode	configMode	string	Specifies how the operator will obtain metadata to generate tables. Accepted values: • Static (from configuration): the name, columns and primary key(s) of the generated table(s) will be determined by the respective configuration values. • Dynamic (from input): the name, columns and primary key(s) of the generated table(s) will be determined by the input Table Message. Default: ""

Name	ID	Type	Description
Table Name	tableName	string	The name of the generated table in Static mode. Default: ""
Table Columns	tableColumns	array	The column definition for the table(s) generated in Static mode. Default: []
Primary Key	primaryKey	array	A list of strings holding the name(s) of the column(s) that compose the table's primary key. Default: []
Output Mode	outputMode	string	In Static mode, it determines when the operator should generate tables. Accepted values: - Once: the operator will output a single table, then become idle. - Periodic: the operator periodically outputs a table. - Trigger: the operator outputs a table upon receiving the set number of inputs (see Trigger count). Default: "Once"
Period	period	string	When in Periodic mode, this property determines the interval between consecutive outputs. Default: ""
Trigger Count	triggerCount	integer	When in Trigger mode, this property sets the number of inputs that should be received before triggering each output. Default: 1

Name	ID	Type	Description
Rows per Output	rowsPerOutput	integer	The number of rows of random data to be generated for each output table. May be set to zero for empty tables. Default: 5

Input

Input	Type	Description
trigger	any.*	In Static mode, this input is consumed only if Output mode is set to Trigger. In this case, the value itself is ignored and may be of any type. In Dynamic mode, the input must be a Table Message describing the table to be generated.

Output

Output	Type	Description
output	message.table	A Table Message generated according to the requested schema. If Rows per output is greater than zero, the message body will have randomly-generated data of the appropriate type for each column.

2.18 Utilities

Related Information

[Avro Serializer \[page 519\]](#)
[Avro Deserializer \[page 520\]](#)
[Flowgraph Compiler \[page 521\]](#)
[Format Converter \[page 522\]](#)
[Graph Terminator \[page 525\]](#)
[Header Remover \[page 526\]](#)
[Histogram \[page 527\]](#)
[Histogram Plotter \[page 529\]](#)
[HTML Viewer \[page 529\]](#)
[Message Aggregator \[page 530\]](#)
[Message Key Store \[page 533\]](#)
[Py3 Data Generator \[page 537\]](#)
[Py3 Type to Blob \[page 537\]](#)
[Py3 Type to String \[page 538\]](#)
[Terminal \[page 539\]](#)
[Wiretap \[page 541\]](#)

2.18.1 Avro Serializer

The Avro Serializer operator converts CSV encoded lines and produces the contents as Avro encoded messages. It does not support nested records.

Configuration Parameters

Name	ID	Type	Description
Avro Schema	avroSchema	string	Avro schema used to read fields names and types. Default: ""
Delimiter	delimiter	rune	Ascii code for delimiter character, the default is "," code. Default: 44
Header Included	headerIncluded	bool	Used to indicate if there is a header on CSV first line. Default: true

Name	ID	Type	Description
Include Schema	includeSchema	bool	The schema should be included on avro message. Default: false

Input

Input	Type	Description
in	message	Message with CSV in the body.

Output

Output	Type	Description
out	message	Avro representation of the input message.

2.18.2 Avro Deserializer

The Avro Deserializer operator converts Avro-encoded messages and produces the contents as CSV-encoded lines.

Configuration Parameters

Name	ID	Type	Description
codeSchema	codecSchema	string	The Avro schema used for input message encoding. Default: ""

Input

Input	Type	Description
in	message	The Avro-encoded message.

Output

Output	Type	Description
out	message	A CSV representation of the input message.

2.18.3 Flowgraph Compiler

Does a flowgraph conversion.

Configuration Parameters

Name	ID	Type	Description
Flowgraph Type	Flowgraph type	string	Mandatory. Flowgraph Type. Default: "basic"
Reference Directory Location	Reference directory location	string	Reference directory location. Default: "/tmp"
Side Effect Location	Side effect location	string	Mandatory. Side effect location. Default: "/tmp"

Input

Input	Type	Description
in	string	The flowgraph description.

Output

Output	Type	Description
out	blob	The jar file.
outmessage	message	The body of the message containing the jar file and its attributes containing the Spark configuration file.

2.18.4 Format Converter

The Format Converter operator converts blobs between CSV, JSON and XML. The output format is set as a configuration value. The input format is deduced using the following rules:

The input format is deduced using the following rules:

- If it starts with { or [, it must be JSON.
- If it starts with <, it must be XML.
- Otherwise, it must be CSV.

If the first field in CSV input starts with any of the three special characters ({, [and <), it must be quoted.

CSV

The CSV format conforms to [RFC 4180](#) , with the following observations:

- Any valid UTF-8 character or escape sequence may be used as field separator ("comma").
- Output CSV will use CRLF as a record separator, placing it after every record including the last one.
- Input CSV can use either LF or CRLF as a record separator, and the last record may or may not be followed by a separator.
- Because CSV lacks any type information, when converting to JSON, all fields will be made into strings.

XML

Converting to XML requires one of the following to be true:

- `xmlOutputRootTag` is not empty, in which case all input content will be placed within this tag. For example, input

```
[{"a": 1}, {"b": "hello"}]
```

becomes

```
<root>
  <item>
    <a>1</a>
  </item>
  <item>
    <b>hello</b>
  </item>
</root>
```

where `xmlOutputRootTag` is `root` and `xmlOutputItemTag` is `item`.

- Input is a JSON object containing a single property, whose key will be used as root tag. For example:

```
{"countries": [{"id": "US", "currency": "USD"}, {"id": "DE", "currency": "EUR"}]}
```

becomes

```
<countries>
  <country>
    <id>US</id>
    <currency>USD</currency>
  </country>
  <country>
    <id>DE</id>
    <currency>EUR</currency>
  </country>
</countries>
```

assuming `xmlOutputRootTag` is empty and `xmlOutputItemTag` is `country`.

Converting from XML requires no XML-specific configuration. Sibling tags with the same name will be placed in an array named after them. For example:

```
<people>
  <person>
    <name>Bonnie</name>
    <age>23</age>
  </person>
  <person>
    <name>Clyde</name>
    <age>25</age>
  </person>
</people>
```

will be turned into

```
{
  "people": {
    "person": [
      {"name": "Bonnie", "age": "23"},

```

```

    { "name": "Clyde", "age": "25" }
  ]
}

```

Notice that all textual values in XML become JSON strings.

Configuration Parameters

Parameter	Type	Description
csvComma	string	The field separator. Default: ","
targetFormat	string	The file type to be converted to. Accepted values are JSON, CSV or XML. Default: "JSON"
csvHeaderIncluded	bool	Indicates whether or not each CSV input will have a header as first line. If the header will not be present, the fields parameter must be supplied. Default: false
fields	string	A comma-separated list of field names. This value is required only when: - converting from headerless CSV - converting to CSV Default: " "
xmlOutputRootTag	string	The name for the root tag to be used when converting to XML. If this value is empty, the operator will be able only to convert JSON input, which must be an object with a single property that will serve as root tag. Default: "items"
xmlOutputItemTag	string	The tag name to be used for each element in an array when converting to XML. If this value is empty, xmlOutputRootTag will be used. Default: "item"

Input

Input	Type	Description
input	blob	The source data to be converted.

Output

Output	Type	Description
output	blob	The resulting data, in <code>targetFormat</code> format.

2.18.5 Graph Terminator

The Graph Terminator operator terminates the current graph execution.

Configuration Parameters

None

Input

Input	Type	Description
stop	any.*	A string that triggers the termination. The content of the string is ignored.

Output

None

2.18.6 Header Remover

The Header Remover operator removes specified headers from an SAP Vora message. The operator has two output ports: one that produces the SAP Vora message and another that produces only the message body.

Configuration Parameters

Name	ID	Type	Description
Pattern	pattern	string	A regex pattern to match the names of the headers to be removed. Default: ""
Exclude Pattern	excludePattern	string	A regex pattern to match the names of the headers that are excluded from being removed. Default: ""

Input

Input	Type	Description
in	message	An SAP Vora message.

Output

Output	Type	Description
out	message	An SAP Vora message.
outbody	any.*	The body of an SAP Vora message.

2.18.7 Histogram

The Histogram operator receives a numeric value as input and increments the counter of the bin that corresponds to the range in which the value lies. The array of counts is then sent in the output port.

Configuration Parameters

Parameter	ID	Type	Description
Number of Bins	nBins	int	The number of bins of the histogram. It must be greater than zero. Default: 10
Range Min	rangeMin	float64	The minimum value on the x axis. Input values lower than Range Min are ignored. Default: 0.0
Range Max	rangeMax	float64	The maximum value on the x axis. Input values higher than Range Max are ignored. Default: 1.0
Output on nth Input	outputEveryNInput	uint64	Sends an accumulated histogram on every Output on nth Input values received in the input port. It is sent only if the histogram has changed since the last send (it may not have changed if the values received were out of range). Default: 1

Parameter	ID	Type	Description
Output Period (s)	outputPeriodInSec	float64	Sends an accumulated histogram every <code>Output Period (s)</code> seconds. It is sent only if the histogram has changed since the last send (it may not have changed if the values received were out of range). Default: 0.0

Note

Either *Output on nth input* or *Output Period (s)* must be set (greater to zero), but not both.

Input

Input	Type	Description
histIn	any.*	A numeric value that is converted to float64. It can be a string or blob that represents a numeric value or a golang primitive numeric type.

Output

Output	Type	Description
histOut	message	A Data Pipeline message with the histogram in the body as an array of uint64, and the message header has the <code>Range Min</code> and <code>Range Max</code> from the configuration.

2.18.8 Histogram Plotter

The Histogram Plotter operator receives a string-converted Data Pipeline message as input with an array of integers in the body and rangeMin and rangeMax fields in the header.

It then generates a vertical bar plot with one bar corresponding to each position in the received array. The rangeMin and rangeMax fields are used to make the tick marks in the x axis. There are two y axis. The left y axis represents values of the array as percentages (position i: $100 \cdot \text{array}[i] / \text{sum}(\text{array})$). The right y axis shows the actual value in the array. Both y axis have 10 tick marks.

Configuration Parameters

None

Input

Input	Type	Description
in1	string	A string-converted Data Pipeline message with an array of integers in the body. The array should be formatted as a JSON array. The header should contain the fields rangeMin and rangeMax. Use the toStringConverter operator to convert the output of the Histogram operator to give the correct format for this operator input.

Output

None

2.18.9 HTML Viewer

The HTML Viewer operator renders HTML code on the browser as it arrives at the input port.

This operator renders HTML code on the browser as it arrives at the input port.

`script` tags are ignored by the operator, but CSS in `style` tags are accepted.

Configuration Parameters

None

Input

Input	Type	Description
in1	string.*	A string containing HTML code to be rendered.

Output

Output	Type	Description
out1	string.*	Relays the same string that is received.

2.18.10 Message Aggregator

The Message Aggregator operator aggregates input messages in a custom manner specified by JavaScript functions that the user can enter.

Configuration Parameters

Name	ID	Type	Description
Script	script	script	Contains either the script to be executed or the path to the script. For a script, the string must start with "file:///". Default: "file:///default.js"
Partial Aggregation	partialAggregation	bool	Specify true for partial aggregation. Specify false for global aggregation. Default: false

Name	ID	Type	Description
Convert Input Body to Type	convertInputBodyToType	string	Specify a type name for the input message body to be converted into before being passed as an argument to the JavaScript function. Accepted type names: <code>blob</code> (for an array of bytes), <code>string</code>, <code>bool</code> and all primitive numeric types from go-lang. Maps and arrays of types other than byte are currently not supported. Be aware that not all conversions are possible and the Golang conversion  rules apply.
Convert Output Body to Type	convertOutputBodyToType	string	Similar to <code>convertInBodyToType</code>, but it converts the output message body right after the JavaScript function is returned and before the message is sent to the output channel. In addition to the same supported type names from <code>convertInBodyToType</code>, the string "input" is also supported, and uses the input to the JavaScript function for the conversion of its output. Default: ""
Ignore Group Key	ignoreGroupKey	any	If the key returned by the <code>getGroupKey(msg)</code> function is equal to the <code>ignoreGroupKey</code>, then the message received is not passed to the aggregate function. Default: false

Input

Input	Type	Description
in	message	The input message to be aggregated.

Output

Output	Type	Description
out	message	The aggregated message result.

Basic Example 1: Summing with Global Aggregation

When the `Partial Aggregation` parameter is set to false, then the operator expects the user to specify three JavaScript functions:

- `getGroupKey`: Specifies in which group the received message is aggregated.
- `canRelease`: Determines if a given group can already be aggregated and emitted to the out channel.
- `aggregate`: Performs the actual aggregation logic upon a group list of messages.

For example:

```
function getGroupKey(msg) {
    return msg.Body%2;
}
function canRelease(list) {
    return list.length >= 5;
}
function aggregate(list) {
    var sum = 0;
    for (var i = 0; i < list.length; i++) {
        sum += list[i].Body;
    }
    var msg = {};
    msg.Body = sum;
    msg.Attributes = {};
    return msg;
}
```

Basic Example 2: Summing with Partial Aggregation

When the `Partial Aggregation` option is set to true, then the operator expects the user to specify three JavaScript functions:

- `getGroupKey`: Specifies in which group the received message is aggregated.
- `partialAggregate`: Performs a partial aggregation and holds its state inside the JavaScript environment. The key is also passed in case the user wants to use it for grouping. The `partialAggregate` function also should return a bool indicating if the aggregate for the key should be released.
- `getAggregation`: Returns the aggregation for the key passed as argument.

For example:

```
var msgGroupState = {};  
function getGroupKey(msg) {  
    return msg.Body%2;  
}  
function partialAggregate(key, m) {  
    var doRelease = false;  
    if (!(key in msgGroupState)) {  
        msgGroupState[key] = [m.Body, 1];  
        return doRelease;  
    }  
    var state = msgGroupState[key];  
    state[0] += m.Body ;  
    state[1] += 1;  
    if (state[1] === 5) {  
        doRelease = true;  
    }  
    msgGroupState[key] = state;  
    return doRelease;  
}  
function getAggregation(key) {  
    var aggr = msgGroupState[key][0];  
    delete msgGroupState[key];  
    var msg = {};  
    msg.Body = aggr;  
    msg.Attributes = {"key": key};  
    return msg;  
}
```

2.18.11 Message Key Store

Filters input messages. Messages are identified with a unique key

The operator replicates messages received on the input ports to the output ports based on message history. SAP Vora database is used in the background to keep track of the message history.

A simple use case scenario is using Message Key Store operator to load `.csv` files given in a specific path into a database and avoid loading the same file multiple times even after the graph is restarted. In this case, a unique identifier for messages would be file names.

After each file is successfully loaded into the database, an acknowledgement is sent back to the operator.

i Note

The message key store operator does not make decisions about some specific situations for example if a message is lost. Let's say the output of the message key store goes to a consumer operator; then, one can not distinguish about the following two cases:

- successful consumption in the consumer operator but the acknowledge signal is lost due to crash
- unsuccessful consumption in the consumer operator and the message gets lost due to crash

This means there is a need for a decision making mechanism in the graph containing message key store operator that decides what to do when messages are lost.

Configuration Parameters

Title	ID	Type	Description
Connection	connection	object	The object containing the connection parameters to a database instance.
Connection Timeout (ms)	connectionTimeoutInMs	int	The number of milliseconds to wait for when connecting to Vora before timing out. Default: 60000
Message Key	messageKey	string	A unique identifier for the message. The identifier is expected to exist as an attribute for input messages. Default: "message.request.id"
Use Previous Message History	usePreviousMessageHistory	bool	This field determines whether to use a previous history of messages. This is helpful in restart scenarios when scenarios have already processed part of messages and it would like to avoid processing them again. Default: false

Title	ID	Type	Description
Group ID	groupID	string	When Use Previous Message History is set, the field specifies the name of the database table that contains information about the previous message history. Default: ""
Previous Content Initialization	contentInitialization	string	When using previous message history, the field defines what should be done with this history. Possible actions are Resume that resume using the previous message history and Reset that clears all previous message history and starts from scratch. Default: "Resume"
Terminate on Error	terminateOnError	bool	A flag that indicates whether or not the graph should stop if this operator encounters an error. If set to false, an output may represent an error, in which case its attribute message.error will be true and its body will contain a string describing the error. Default: true

Input

Input	Type	Description
input	message.*	The input message to be processed.

Input	Type	Description
ack	message.*	In most of use cases, input message that enters the operator for the first time will be sent for further processing. When the processing is done, the message is pushed back into the ack port. In the example given above about loading files into the database, an acknowledgement is sent back to the operator when file successfully loaded into the database.

Output

Output	Type	Description
output	message.*	A message received on the input port is forwarded to this port if the message hasn't been acknowledged yet.
acked	message.*	a message received on the input port is forwarded to this port if the message is already acknowledged before. Any input received in the ack port is forwarded to this port.

Forwarding on the output port is done according to the following table. `Input` and `Ack` on the first two columns shows whether the message is received on `input` or `ack` ports. The `Acked` column shows whether this message has been acknowledged. The last two columns show `output` and `ack` output ports and for each case the message will be forwarded the desired port if the item in table is true.

Input	Ack	Acked	Output	Ack
False	True	False	False	True
False	True	True	False	True
True	False	False	True	False
True	False	True	False	True

2.18.12 Py3 Data Generator

This operator is an extension of the base operator Python 3. It generates random sample data every 500ms.

Configuration Parameters

None

Input

None

Output

Output	Type	Description
output	string	The generated data.

2.18.13 Py3 Type to Blob

This operator will serialize any Python3 data received in its input port and send the result to the output port. First, the data will try to be serialized by calling the method `to_blob()` on the data. If that doesn't work then the data will be serialized with protocol 2 of the pickle library.

Configuration Parameters

None

Input

Input	Type	Description
input	python36.*	Any Python3 data.

Output

Output	Type	Description
output	blob	The input serialized to bytes.

2.18.14 Py3 Type to String

This operator will convert any Python3 data received in its input to string by feeding it into the standard Python `str(.)` function.

Configuration Parameters

None

Input

Input	Type	Description
input	python36.*	Any Python3 data.

Output

Output	Type	Description
output	string	A string representation of the input received.

2.18.15 Terminal

The Terminal operator represents a terminal window in the browser.

It consists of an output area on top and an input area. The output area is bound to the input port in1. The input area is bound to the output port out1.

The areas can be resized by dragging the frame separator.

Display Areas

- **Output:** Displays in real time the data received from another operator through the bound port `<in1>`. May add a prefix or a suffix to each message as set in layout. The maximum number of lines to be kept on screen can be set at the Line Limit form. Multi-line strings received will count as only one message and display on multiple lines, receiving only one layout formatting.
- **Input:** Displays in real time the data received from another operator through the bound port in1. May add a prefix or a suffix to each message as set in layout. Received multi-line strings will count as only one message and display on multiple lines, receiving only one layout formatting. The number of lines to be kept on screen is limited by the **Line Limit** field, and its default value is `<500>` (setting it to `<0>` removes the limit). When opening the terminal or reconnecting after a period of disconnection, an automatic discontinuity message may be shown, indicating how many messages were lost in the time it was not connected. If all the messages were still present in the back end spool, they will all be sent and no discontinuity message will be shown. The discontinuity marker is not available when spool size is set to `<0>`.

Menu Options

- **Clear Input or Clear Output:** Clears the content of the corresponding area.
- **Line Limit:** Limits the number of lines that will be shown at the Output area. Press Enter after inserting the number to refresh. Setting this to 0 will remove line limit and display as many lines as the machine can receive or process.
- **Keyboard Shortcuts:** Displays a pop-on dialog with the Input are shortcuts for editing commands.

Configuration Parameters

Parameter	Type	Description
disable	bool	Setting this to true disables the output processing in the back end. Default: false
maxSize	int	The maximum size of each message to be displayed in the Terminal. Any message larger than this size in bytes will be truncated when displayed. Default: 8192
spoolSize	int	The number of preceding messages that are spooled and sent to the display that has started at some later time. Default: 100
layout	string	The layout pattern to display each inputted message, consisting of literal characters and the following pieces: • %m: The received string message. • %d: Timestamp. • %n : Newline feed. • %r : Time (milliseconds) between operator creation and message reception at back end component. Default: "[%d] %m%n"

Input

Input	Type	Description
in1	string.*	Input string to be displayed on terminal output area.

Output

Output	Type	Description
out1	string.*	Output string from terminal input area.

2.18.16 Wiretap

The Wiretap operator can wiretap a connection between two operators in a Modeler graph and display the traffic to the browser window or to an external websocket client that connects to this operator.

To view the traffic, open the UI window using the context menu of this operator after the graph has started. If the operator is started in a group with multiplicity, the traffic from all the instances are displayed.

Configuring the output format

The layout parameter can be customized to display the traffic data in some specific format or to hide some parts of data such as unimportant or sensitive data. The default format produces a line with the date and the message.

Layout format specifiers and their usage

Specifier	Description	Usage Samples
%i	The index number of the entry.	%i
%d	The customizable formatted date.	%d
%g	The group node name.	%g
%m	The entire message or payload.	%m
%h	The specified attribute.	%h{openapi.operation_id}
%a	The specified pairs.	%a{message.request.id,openapi.operation_id}
%A	The attributes excluding those specified.	%A{openapi.password}
%b	The message body or payload.	%b
%r	The time in milliseconds since the wiretap has started.	%r
%n	The newline.	%n

Sample parameter samples

Sample Layout	Sample Output
(%g)[%d]%m	(graph-1)[2019-03-04 16:51:10,000]{{"message.request.id":"89efe633"}}hello world
[%d{HH:mm:ss}]%h{message.request.id}	[16:51:10]89efe633
[%i]%a{message.request.id,openapi.remote_addr}	[27]{{"message.request.id":"89efe635","openapi.remote_addr":"80.245.152.130"}}

Configuration Parameters

Parameter	Type	Description
disable	bool	Disables the output when this parameter is set to true. Default: false
maxSize	int	The maximal size of the message to be displayed in the wiretap. Any message larger than this size will be truncated when displayed. Default: 4092
spoolSize	int	The number of preceding messages that are spooled at the operator and can be sent to the display that has started or reconnected. This spool size is truncated by its maximal size of 256. Default: 32
layout	string	The layout pattern to format each line using the conversion characters. Default: "[%d] %m"

Input

Input	Type	Description
in	any.*	Input to this wiretap.

Output

Output	Type	Description
out	any.*	Output from this wiretap. This port may be left unconnected.

2.19 Others

2.20 Deprecated Operators

Related Information

[CDC Graph Generator \(Deprecated\) \[page 543\]](#)
[Copy File \(Deprecated\) \[page 548\]](#)
[Create TA Request \(Deprecated\) \[page 553\]](#)
[Functional Services \(Deprecated\) \[page 556\]](#)
[Move File \(Deprecated\) \[page 574\]](#)
[PA Automated \(Deprecated\) \[page 579\]](#)
[Read File \(Deprecated\) \[page 580\]](#)
[Remove File \(Deprecated\) \[page 590\]](#)
[SAP ABAP ODP Object Consumer \(Deprecated\) \[page 594\]](#)
[Spark Data Transform \(Deprecated\) \[page 598\]](#)
[Text Analysis Connector \(Deprecated\) \[page 599\]](#)
[Write File \(Deprecated\) \[page 603\]](#)

2.20.1 CDC Graph Generator (Deprecated)

Change Data Capture is a Delta Capture Technique using database-specific triggers for Insert, Update and Delete. Since manually creating the SQL per table is time consuming, this operator simplifies this.

This operator will require the source Database connection and source table, and the target connection and target specific properties. The operator does static analysis of the metadata of the source table and creates the SQL script for the given database to create shadow table, triggers, view, and few procedures, and so on. Based on these, the operator generates three graphs.

i Note

Because this operator does only static analysis, the generated SQL Script needs to be evaluated by your DBA before running the graphs.

Port	Description	Path
outTriggersGraph	Graph with DDL to create triggers on source system	/vrep/vflow/ graphs/com/sap/cdc/ initializeTrigger/ graph.json
outInitialLoadGraph	Graph to do initial load from source to target	/vrep/vflow/ graphs/com/sap/cdc/ initialLoad/graph.json
outDeltaGraph	Graph with delta capture logic	/vrep/vflow/ graphs/com/sap/cdc/delta/ graph.json

The graphs will be generated based on the [Path](#) configuration of Write File operator. Modify that if you want the graph to be generated under different location.

Supported Sources

- DB2
- HANA
- MySQL
- Microsoft SQL Server
- Oracle

Note

The following source-specific SQL Privileges are required. Ensure the connection user has the following privileges.

- CREATE TABLE
- CREATE VIEW
- CREATE SEQUENCE
- CREATE PROCEDURE
- CREATE TRIGGER

Supported Targets

- Databases (HANA, VORA)
- Cloud Storage (ADL, WASB, S3, Alibaba Cloud OSS, GCS, HDFS)
- Local File system (local)

Modes

For HANA as target we support two modes:

- Track Change History
- Apply Change Data

Limitations

- Schema changes (for example, changes through DDLs) are not supported.
- Truncate Table operation is not supported.

Delta Graph Logic

Several methods exist to capture the changes in a source table: timestamp based, native CDC, log reading, and so on. Native CDC methods with transactional consistency are difficult to scale due to their atomicity nature. For SAP Data Intelligence purposes, where the data volume can be significantly higher for certain tables, we propose to use triggers. Note that `trigger` is the only method that is supported for all databases.

To achieve this, we created the following objects in the source schema.

Object	Type	Description
srcTable_CLEANUP	PROCEDURE	Initial procedure that will drop existing triggers and other database objects created for this srcTable.
srcTable_SEQUENCE	SEQUENCE	Some databases will use a SEQUENCE to generate an increasing ID.
srcTable_SHADOW	TABLE	Shadow table that will be used for storing the delta records along with transaction type.
srcTable_TRIGGER_QUEUE	TABLE	Queue table that will be used for marking the records from shadow table for replication.
srcTable_SHADOW_VIEW	VIEW	View that inner joins the marked records from queue table with shadow table.
srcTable_CHECKPOINT	TABLE	Checkpoint table is used to indicate when we began replication and when we completed the delta.

Object	Type	Description
srcTable_INSERT_TRIGGER	TRIGGER	Insert trigger.
srcTable_DELETE_TRIGGER	TRIGGER	Delete trigger.
srcTable_UPDATE_TRIGGER	TRIGGER	Insert trigger.
srcTable_INITIAL_SHADOW_SCAN	PROCEDURE	Procedure to determine when the graph should stop running.
srcTable_SHADOW_SCAN	PROCEDURE	Procedure to mark shadow table records into queue table.
srcTable_SHADOW_CHECKPOINT	PROCEDURE	Procedure to update timestamp on checkpoint table.
srcTable_SHADOW_CLNEAUP	PROCEDURE	Procedure to cleanup shadow and queue table both.

When you create the trigger on the source, any changes happening to the source table are replicated in the shadow table.

When you trigger the delta graph, the following actions happen:

- srcTable_SHADOW_SCAN procedure is called on the source, which will copy the current records from the shadow table to the queue table.
- The records in the queue table have a load status (indicating whether the records were loaded to target or not).
- Table Consumer will read the data from the view only the records that are in queue table and will load to target.
- srcTable_SHADOW_CHECKPOINT procedure is called to update the load status of the records in the queue table.
- srcTable_SHADOW_CLEANUP is called to delete the records that were successfully loaded to target from queue and shadow tables.

Configuration parameters

The following parameters can or must (*) be provided to execute a data transfer.

Parameter	Description
source*	Source system
target*	Target system
Delta Graph Run Mode*	Polling Interval vs Manual

Parameter	Description
Delta Batch Size*	How many rows will be fetched per polling period
Tablespace	Tablespace where the SHADOW and TRIGGER_QUEUE tables should be created

Based on the source or target system the configuration is based on. See the Table Consumer operators for your source, [Flowagent File Producer \[page 143\]](#), and [Flowagent Table Producer \[page 150\]](#).

Delta Graph Run Mode

- **Polling Interval:** In this mode, delta graph runs every N seconds. Use this mode if the source table changes frequently. In this mode, the delta graph pod runs forever. It will first run the first delta job, then wait N seconds and trigger the delta job again.
- **Manual:** The graph terminates after running once. Use this mode if the source is does not change frequently. Running once only can save pod resources when not running. This is also useful when you have multiple tables to replicate. For example, say that you have 100 tables in the sources. Ten of those are changing frequently, while the remaining 90 are updated less frequently. To utilize the pod resources efficiently, one can have the frequent tables' graph running indefinitely, and the remaining 90 tables delta graphs can be chained in another workflow graph. That way, only one pod, is used and it will run the delta graph first, and then the second, and so on.

Input

Input	Type	Description
input	string	Mandatory. The input signal received from an another operator that can be used as a trigger. You can always use Constant Generator to trigger this operator.

Output

Output	Type	Description
outTriggersGraph	string	The graph for triggers creation.

Output	Type	Description
outInitialLoadGraph	string	The graph for doing the initial load of data from the source to target.
outDeltaGraph	string	The graph for delta capture logic to capture changes from the source to target.

Additional Information

- Connect this operator to the Trigger/Constant Generator or Terminator operator if it is the first or last operator that you want to be executed.
- During the execution, it will create multiple graphs but not execute them.
- Validate and optimize the generated graphs before running things to consider are.
- Changing the fetchSize for the consumer (wide tables use a lower number like 100, narrow tables use a higher number).
- Adding a logical partition in the consumer, and grouping the producer to run the initial load in parallel.
- Because this operator will do only static analysis, the generated SQL Script needs to be evaluated by your DBA before running the graph on a production database.

Related Information

[Improving CDC Graph Generator Operator Performance](#)
[Changing Data Capture \(CDC\)](#)
[CDC Graph Generator Sample Graph \[page 659\]](#)

2.20.2 Copy File (Deprecated)

The Copy File operator is used to copy files in a storage service. The expected input message is specified in the Input section in this topic.

Note

This operator is deprecated. Use the operators [Read File \[page 303\]](#) and [Write File \[page 309\]](#) to perform the same action.

Additionally, [List Files \[page 296\]](#) can be used to move directories.

This operation is recursive, meaning it will copy any file or directory under the given source path.

If destination is a file, it will be overwritten; if it is a non-empty directory, it will fail.

Supported services are:

- [Google Cloud Storage \(GCS\)](#)
- [Amazon Simple Storage Service \(S3\)](#)
- [Microsoft Azure Storage Blob \(WASB\)](#)
- [Alibaba Cloud Object Storage Service \(Alibaba Cloud OSS\)](#)

Use `Connection Management` for defining connections over the manual option.

Note

This operator supports only services that have such operations available. On other services, [Copy](#) can be achieved through [Read](#) + [Write](#).

Behavior Example

In the given file structure:

```
.
|
+-- a
|   +-- file1.txt
|   +-- file2.txt
+-- b
    +-- f1.txt
    +-- f2.txt
```

- Copying source `a/file1.txt` to destination `newfile.txt` would succeed because the destination does not exist.
- Copying source `a/file1.txt` to destination `b/f1.txt` would succeed and overwrite `b/f1.txt` because the destination is an existing file.
- Copying source `a/file1.txt` to destination: `b/` would fail because `b/` already exists and is not empty.
- Copying source `a/` to destination `b/` would fail because `b/` already exists and is not empty.
- Copying source `a/` to destination `b/dir/` would succeed because `b/dir/` does not exist.

Configuration Parameters

Name	ID	Type	Description
Service	service	string	The file service to operate. Additional parameters may depend on the selected service. Default: "GCS"

Name	ID	Type	Description
Connection	connection	object	Holds information about the service's connections. Only use manual connections when using a connection ID is not possible. Each service connection parameters is documented separately: • Google Cloud Storage (GCS) • Amazon Simple Storage Service (S3) • Microsoft Azure Storage Blob (WASB) • Alibaba Cloud Object Storage Service (Alibaba Cloud OSS)
Configuration Type	configurationType	string	<i>connection parameter</i> Which type of connection information will be used: Manual (user input), or retrieved by the Connection Management Service. i Note It is recommended to choose <i>Configuration Manager</i> as the configuration type and select an existing <i>Connection ID</i>. The <i>Manual</i> configuration type must be used only for testing purposes. Default: ""
Connection ID	connectionID	string	<i>connection parameter</i> The ID of the connection information to retrieve from the Connection Management Service. Default: ""
Connection Properties	connectionProperties	object	<i>connection parameter</i> All the connection properties for manual input.

Name	ID	Type	Description
Timeout (ms)	timeoutInMs	int	Sets the time limit to execute the operation. If 0, no time-out is used. Default: 0
Retry Period (ms)	retryPeriodInMs	int	The time interval in milliseconds between connection trials. Default: 0
Retry Attempts	numRetryAttempts	int	The number of times to retry a connection. Default: 0
Simultaneous Requests	simultaneousRequests	int	The number of simultaneous requests generated on recursive calls (only available for GCS, S3, and WASB). Default: 1
Stop Requests on Error	stopRequestOnError	boolean	Controls whether simultaneous requests from recursive calls should stop at first error (only available for GCS, S3, and WASB). Default: false
Terminate on Error	terminateOnError	boolean	A flag that indicates whether the graph should stop if this operator encounters an error. If set to <code>false</code>, an output may represent an error, in which case its attribute <code>message.error</code> will be <code>true</code> and its body will contain a string describing the error. Default: true

Input

Input	Type	Description
in	message	A message whose body has the source file path and destination. Expected message attributes: <i>storage.src</i> Mandatory. Path to source file or directory. <i>storage.dst</i> Mandatory. Path to destination file or directory (should match types with source). The destination path may contain <code><dirname></code> and <code><basename></code>, referring to the given source path's <code>dirname</code> and <code>basename</code>. For example, <code>move src:data/A/user.csv dst:<dirname>/processedFiles/<basename></code> would map to the destination <code>data/A/processedFiles/user.csv</code>. <i>storage.srcBucket</i> Overrides <code>bucket</code> from configuration on service GCS and S3 for the source file or directory. <i>storage.dstBucket</i> Overrides <code>bucket</code> from configuration on service GCS and S3 for the destination file or directory.

Output

Output	Type	Description
out	message	A message whose header <code>message.error</code> (bool) reports whether the operation was successful. The body and any other header from the input message are copied to this one.

2.20.3 Create TA Request (Deprecated)

Creates a request for the Text Analysis Connector operator in JSON format.

i Note

This operator is deprecated. Please check SAP Note [2739161](#) for more information.

See the Text Analysis section of the Developer Guide for SAP Vora for more information about the text analysis configurations and output format.

Configuration Parameters

Name	ID	Type	Description
Server endpoints	Text Analysis server endpoint	string	Mandatory. Host name and port number of the vora-text-analysis service. For example: 'vora-textanalysis:10002'.
Storage service	Storage service	string	The file service where files to be analyzed are located. Additional parameters may depend on the selected service.
Connection	Connection	object	Holds information about connection information for the services. Each service connection parameters is documented separately: • Microsoft Azure Data Lake (ADL) • Google Cloud Storage (GCS) • Hadoop Distributed File System (HDFS) • Amazon Simple Storage Service (S3) • Microsoft Azure Storage Blob (WASB) • WebHDFS

Name	ID	Type	Description
Text Analysis configuration	Text Analysis configuration	string	Standard configuration for text analysis service. • LINGANALYSIS_BASIC • LINGANALYSIS_STEMS • LINGANALYSIS_FULL • EXTRACTION_CORE • EXTRACTION_CORE_ENTERPRISE • EXTRACTION_CORE_PUBLIC_SECTOR • EXTRACTION_CORE_VOICEOFCUSTOMER
Languages	Languages	string	A list of languages used for language detection specified in ISO 639-1 codes. For example: 'EN, DE, ES'. If no language is specified, then automatic detection is attempted.

Name	ID	Type	Description
Mime type	Mime type	string	Optional. The type of input documents. Allowed values are: • 'text/plain' • 'text/html' • 'text/xml' • 'text' The value 'text' indicates that the input is one of plain text, HTML or XML. If not set, or if value is 'text', document identification and conversion are performed. i Note The analysis of documents in binary formats is also supported, but automatic format detection is used in this case.
Text encoding	Text encoding	string	If the document contains text, this parameter indicates the encoding. For example: 'UTF-8'. If not set and the MIME type indicates text, encoding detection and conversion are performed.

Input

Input	Type	Description
input	string	A string containing the following values, separated by comma: • <code>folderpath</code> [mandatory]: The location of the files to be analyzed, relative to the root path included in the connection configuration. • <code>recursive_flag</code> (true or false) [mandatory]: If true, the analysis is done recursively in the subfolders in the specified location. • <code>document_id</code> (default:0): The next id to be given to a document in the result. If provided, and equal to the current value, it is increased by one. • <code>root_request</code> (true or false, default:true): Indicates if the request is the initial request in the graph. Examples: <pre>'/path/to/folder,true,0,false','/path/to/folder,false'</pre>

Output

Output	Type	Description
output	string	A request for text analysis in JSON format. See the documentation of the Text Analysis Connector (Deprecated) [page 599] operator (<code>com.sap.textanalysis.taconnector</code>) for more details.

2.20.4 Functional Services (Deprecated)

This operator gives you access to pretrained functional services based on Machine Learning.

i Note

This operator is deprecated. Please check SAP Note [2739161](#)  for more information.

Supported services are:

- [Image Classification \(Deprecated\) \[page 558\]](#)
- [Image Feature Extraction \(Deprecated\) \[page 560\]](#)
- [Image OCR \(Deprecated\) \[page 562\]](#)
- [Similarity Scoring \(Deprecated\) \[page 567\]](#)
- [Text Classifier \(Deprecated\) \[page 569\]](#)
- [Topic Detection \(Deprecated\) \[page 571\]](#)

Configuration Parameters

Choose the desired functional service with the configuration option `Functional Service`. Depending on which service is chosen, certain configuration options are displayed.

These service-specific configuration parameters can be either set in the configuration dialog or as part of the input data to the service operator. The description of the configuration options as well as the input and output data can be found in the documentation for the respective service by selecting the name of the functional service below.

All services require a connection to ML Foundation. There are four options for setting the connection in the configuration parameter `ML Connection`:

Name	ID	Type	Description
ML Connection	mlAPIConnection	object	Information about connection to ML Foundation.
ML Default			The URLs and credentials. If you do not know what connection to choose this is probably the right choice for you.
HTTP Connection			Connections of this type need to have URLs and credentials set in the Connection Management app.
HTTP Connection ID	httpConnectionID	string	The ID of the connection information to retrieve from the Connection Management Service. Default: ""

Name	ID	Type	Description
ML Cluster Connection			Connections of this type will already have all information defined in the Connection Management app.
Connection ID	connectionID	string	The ID of the connection information to retrieve from the Connection Management Service. Default: ""
Manual			A custom connection requiring all specific fields to be filled.

Related Information

[Image Classification \(Deprecated\) \[page 558\]](#)

[Image Feature Extraction \(Deprecated\) \[page 560\]](#)

[Image OCR \(Deprecated\) \[page 562\]](#)

[Similarity Scoring \(Deprecated\) \[page 567\]](#)

[Text Classifier \(Deprecated\) \[page 569\]](#)

[Topic Detection \(Deprecated\) \[page 571\]](#)

2.20.4.1 Image Classification (Deprecated)

This is a pre-trained image classification service. The model for the classifier was obtained by training on the ILSVRC2014 Task 2 dataset. The service takes as input one or several images and classifies each of them into a set of 1000 categories such as trees, animals, food, vehicles, people, and more.

i Note

This operator is deprecated. Please check SAP Note [2739161](#)  for more information.

Input

Input	Type	Description
input	message	A message whose body is the path to the file to be classified or the path to an archive with one or several files to be classified. In this case the header is not required as there are no configuration parameters. Accepted file extensions: zip, tar, gz, tgz, jpg, jpe, jpeg, png, gif, bmp, tif, or tiff.

Example of input message:

header	body
{}	"/path/to/file/sample.jpg"

Or with archive:

header	body
{}	"/path/to/file/sample.zip"

For example, you could use a JS operator, enter the following script to define the input text, and wire this operator to the Functional Service operator:

```
$.addGenerator(gen)
function gen(ctx) {
  var msg = {};
  msg.Body = "/path/to/file/sample.jpg"
  $.output(msg)
}
```

Example of input message with input file content as blob in message body

header	body
{"storage.filename": "/path/to/file/sample.jpg"}	Serialized file as blob.

In this case it is required to send the file name as an attribute with name `storage.filename`:

You can directly wire the `outFile` port from the `Read File` operator to the input port of this operator.

Output

Output	Type	Description
output	message	The classification result as a JSON string. The top five categories or labels are listed together with their predicted probability, ordered by descending probability.

Example:

```
{
  "id": "f09a0cfd-e1bc-4740-5e56-919a94a56d5a",
  "predictions": [
    {
      "name": "image1.jpg",
      "results": [
        {
          "label": "chainlink fence",
          "score": 0.71404
        }, ...
      ]
    },
    {
      "name": "image2.jpg",
      "results": [
        {
          "label": "spider web, spider's web",
          "score": 0.44043
        }, ...
      ]
    }
  ],
  "processedTime": "2017-09-29T13:38:27.729971",
  "status": "DONE"
}
```

2.20.4.2 Image Feature Extraction (Deprecated)

A functional service for extracting image features, one vector for each input image. The vectors can be used for comparison, information retrieval, clustering, or further processing.

i Note

This operator is deprecated. Please check SAP Note [2739161](#) for more information.

Input

Input	Type	Description
input	message	A message that contains the input data to the service.

The body of the message can be one of the following:

Message Body	Header	Body
A path to an input image (supported are: jpg, jpe, jpeg, png, gif, bmp, tif, or tiff). The header is ignored in this case.	{}	/home/test/sample.jpg
A path to an archive containing several image files (supported are: zip, tar, gz, tgz). The header is ignored in this case.	{}	/home/test/sample.zip
A blob containing the input image or input archive (for supported formats, see above). In this case it is required to include the file name as an attribute with name storage.filename in the header.	{"storage.filename": "/home/test/sample.jpg"}	Serialized file as blob.

You can directly wire the outFile port from the Read File operator to the input port of this operator.

Output

Output	Type	Description
output	message	The inference result as a JSON string, the format varies according to the service.

Example:

```
{
  "id": "f09a0cfd-e1bc-4740-5e56-919a94a56d5a",
  "predictions": [
    {
      "name": "image1.jpg",
      "featureVectors": [
        1.5959638357162476, ...
      ]
    },
    {
      "name": "image2.jpg",
      "featureVectors": [
        0.20161056518554688, ...
      ]
    }
  ]
}
```

```
"processedTime": "2017-10-06T10:21:33.575961",  
"status": "DONE"  
}
```

Error: If an error happens due to an input, this port outputs a message. If this port is not connected, a failure will stop the graph with an error.

2.20.4.3 Image OCR (Deprecated)

The inference service for Optical Character Recognition (OCR) takes an image or PDF file as input and outputs the recognized text from the image (or PDF).

i Note

This operator is deprecated. Please check SAP Note [2739161](#)  for more information.

Configuration Parameters

Name	ID	Type	Description
options	optionsOCR	object	Object that must be configured with the following fields: • lang • outputType • pageSegMode • modeType Default: according to parameter.

Name	ID	Type	Description
Languages	lang	string	Mandatory. The language of the text submitted separated by commas. A maximum of 3 languages can be set: • en: English (Default) • de: German • fr: French • es: Spanish • ru: Russian • ar: Arabic • zh-Hans: Chinese Simplified • zh-Hant: Chinese Traditional • ja: Japanese • ja-vert: Japanese (vertical text) • ko: Korean • ko-vert: Korean (vertical text) • pt: Portuguese Default: "en"
Output Type	outputType	string	Mandatory. The output type of the result. • txt: plain text (Default) • xml: text with markup and additional attributes Default: "txt"

Name	ID	Type	Description
Page Segmentation Mode	pageSegMode	string	Mandatory. The page segmentation mode. • 0: Orientation and script detection (OSD) only • 1: Automatic page segmentation with OSD (Default) • 3: Fully automatic page segmentation, but no OSD • 4: Assume a single column of text of variable sizes • 5: Assume a single uniform block of vertically aligned text • 6: Assume a single uniform block of text • 7: Treat the image as a single text line • 8: Treat the image as a single word • 9: Treat the image as a single word in a circle • 10: Treat the image as a single character • 11: Sparse text. Find as much text as possible in no particular order • 12: Sparse text with OSD • 13: Raw line. Treat the image as a single text line, bypassing hacks that are Tesseract-specific Default: "1"

Name	ID	Type	Description
Model Type	modelType	string	Mandatory. Type of the machine learning model for OCR. • IstmPrecise: precise model with LSTM cells • IstmFast: fast model with LSTM cells • IstmStandard: standard model with Istm cells (Default) • noLstm: model without LSTM cells • all: model with LSTM cells and standard processing algorithms Default: "IstmStandard"

Input

Input	Type	Description
input	message	A message whose body is the input information sent to the service. The header can optionally contain configuration parameters. Configuration parameters can be either set from this message header or from the configuration dialog of the operator. If at least one configuration parameter is set in the message header then the parameters from the configuration dialog are completely ignored. The input file can be either specified with a file path or as serialized blob in the message body. Accepted file formats or extensions: jpg, jpeg, png, pdf, or PDF.

Example of input message with configuration parameters and path to input file

header	body
<code>{"options": {"lang": "en,de", "outputType": "txt", "pageSegMode": "1", "modelType": "lstmStandard"}}</code>	<code>"/path/to/file/sample.jpg"</code>

For example, you could use a JS operator, enter the following script to define the input parameters and input file path, and wire this operator to the Functional Service operator:

```
$.addGenerator(gen)
function gen(ctx) {
  var msg = {};
  msg.Attributes = {};
  msg.Attributes["options"] = '{"lang": ["en","de"], "outputType": "xml",
"pageSegMode": "1", "modelType": "lstmStandard"}';
  msg.Body = "/path/to/file/sample.jpg"
  $.output(msg)
}
```

Example for input message with input file content as blob in message body

header	body
<code>{"storage.filename": "/path/to/file/sample.jpg", "options": {"lang": ["en","de"], "outputType": "txt", "pageSegMode": "1", "modelType": "lstmStandard"}}</code>	Serialized file as blob.

In this case, it is required to send the file name as an attribute with name `storage.filename`.

header	body
<code>{"storage.filename": "/home/test/sample.jpg"}</code>	Serialized file as blob.

You can directly wire the `outFile` port from the Read File operator to the input port of this operator.

Output

Output	Type	Description
output	message	The inference result as a JSON string, the format varies according to the service.

Example:

```
{
  "id": "string",
  "predictions": [
    "This is the recognized text from the input image"
  ],
  "processedTime": "string",
}
```

```
"status": "DONE"
}
```

Error: If an error happens due to an input, this port outputs a message. If this port is not connected, a failure will stop the graph with an error.

2.20.4.4 Similarity Scoring (Deprecated)

A functional service for comparing vectors using a similarity score ranging from -1 to 1 (larger values mean a high similarity).

Note

This operator is deprecated. Please check SAP Note [2739161](#) for more information.

The similarity score is calculated based on the cosine similarity of the vectors. The vectors are ranked by similarity and the IDs of the top n similar vectors are listed in the output together with their similarity score. The vectors retrieved from the “image feature extraction” service can be used as input to this service.

Configuration Parameters

Name	ID	Type	Description
Options	optionsSim	object	
Number of Similar Vectors	numSimilarVectors	int	Number of most similar vectors to return in response.

Input

Parameter	Type	Description
input	message	A message whose body is the input information sent to the service. The numSimilarVectors parameter needs to be set either in the configuration dialog of the operator or in the header of the input message.

The input vectors for the service can be provided in three different format options. These input formats cannot be mixed. By default, all vectors in the input will be compared to each other (all pairwise combinations). The third input format also allows to define a “query set”.

Format Options

- A path to an archive file containing the input vectors can be passed in the message body. Accepted file extensions or formats: zip, tar, gz or tgz. The archive needs to contain one file for each input vector. The file extension should be .txt with the following format [number, number, number, ..., number]. All values in one line.

Examples of input message type files:

header	body
<code>{"options": {"numSimilarVectors":1}}</code>	<code>/home/test/sample.zip</code>

- The archive (with the same format as described above) can be sent as blob in the message body. In this case it is required to include the file name as a header attribute `storage.filename`. For example:

header	body
<code>{"storage.filename": "/path/to/file/sample.zip", "options": {"numSimilarVectors":1}}</code>	<code>Serialized file as blob.</code>

- The body of the message can contain the input vectors encoded as a JSON format string, which holds the set of input vectors. Each vector is represented as an object that includes two key-value pairs: ID and vector. The ID is a string, and vector is an array of numbers. The set of vectors is represented as an array of vector objects. The input JSON object should include one or two key-value pairs. If one set will be provided, the object should include one key-value pair, where key is '0', and value is a set of vectors. In this case each vector in the set will be compared to the other vectors in this set. If two sets are provided, the object should include two key-value pairs, where the names are '0' and '1' and with both values being sets of vectors. In this case, for each vector from set '0' top numSimilarVectors similar vectors from set '1' will be found.

Example for one set of vectors

header	body
<code>{"options": {"numSimilarVectors":1}}</code>	<code>{"0": [{"id": "v0", "vector": [0.22, 0.93]}, {"id": "v1", "vector": [0.39, 0.69]}]}</code>

Example for two sets of vectors, the first set is the "query set"

header	body
<code>{"options": {"numSimilarVectors":1}}</code>	<code>{"0": [{"id": "v0", "vector": [0.22, 0.93]}, {"id": "v1", "vector": [0.39, 0.69]}], "1": [{"id": "u0", "vector": [0.45, 0.12]}, {"id": "u1", "vector": [0.32, 0.68]}]}</code>

Output

Output	Type	Description
output	message	The inference result as a JSON string.

Example:

```
{
  "id": "f09a0cfd-e1bc-4740-5e56-919a94a56d5a",
  "predictions": [
    {
      "id": "v1",
      "similarVectors": [
        {
          "id": "v5",
          "score": 0.7713963286130165
        }, ...
      ]
    },
    {
      "id": "v2",
      "similarVectors": [
        {
          "id": "v10",
          "score": 0.7770760429620751
        }, ...
      ]
    }
  ],
  "processed_time": "2017-10-06T10:21:33.575961",
  "status": "DONE"
}
```

Error: If an error happens due to an input, this port outputs a message. If this port is not connected, a failure will stop the graph with an error.

2.20.4.5 Text Classifier (Deprecated)

This is a pre-trained text classification service.

Note

This operator is deprecated. Please check SAP Note [2739161](#) for more information.

The model for the classifier was obtained by training on approximately 300,000 Iccat product texts. The classifier chooses from the following set of 50 categories: all-in-one PCs workstations, LED TVs, NAS and storage servers, PC workstations, USB cables, USB flash drives, antivirus security software, cable interface gender adapters, computer monitors, digital cameras, fiber optic cables, flat panel spare parts, graphics cards, ink cartridges, internal hard drives, keyboards, laser LED printers, laser toner and cartridges, memory modules, mice, mobile phone cases, motherboards, mounting kits, multifunctionals, network switches, networking cables, networking cards, notebook cases, notebook spare parts, notebooks, other, power adapters and inverters, power cables, printer kits, printer scanner spare parts, processors, projection screens, projector mounts, rechargeable batteries, screen protectors, servers, smartphones, software licenses upgrades, solid

state drives, tablet cases, tablets, thin clients, uninterruptible power supplies, warranty and support extensions, washing machines.

Input

Parameter	Type	Description
input	message	A message whose body is the string to be classified. The header is not required as there are no configuration parameters.

Example of input message

header	body
{}	"15 inch lenovo laptop"

For example, you could use a JS operator, enter the following script to define the input text, and wire this operator to the Functional Service operator:

```
$.addGenerator(gen)
function gen(ctx) {
  var msg = {};
  msg.Body = "15 inch lenovo laptop"
  $.output(msg)
}
```

Output

Output	Type	Description
output	message	The inference result as a JSON string. All 50 categories or labels are listed together with their predicted probability, ordered by descending probability.

Example:

```
{
  "id": "f09a0cfd-e1bc-4740-5e56-919a94a56d5a",
  "predictions": [
    {
      "results": [
        {
          "label": "notebooks",
          "score": 0.71404
        },
        {
```

```

        "label": "digital cameras",
        "score": 0.02382
      }, ...
    ]
  },
  "processedTime": "2017-09-29T13:38:27.729971",
  "status": "DONE"
}

```

Error: If an error happens due to an input, this port outputs a message. If this port is not connected, a failure will stop the graph with an error.

2.20.4.6 Topic Detection (Deprecated)

A functional service for detecting and ranking topics from documents. Extracted keywords from provided documents are based on Non-negative Matrix Factorization (NMF).

Note

This operator is deprecated. Please check SAP Note [2739161](#) for more information.

Configuration Parameters

Name	ID	Type	Description
options	optionsTopic	object	Mandatory. Object that must be configured with the parameter fields that follow.
Number of Topics	numTopics	int	Mandatory. Total number of topics to be detected Default: 20
Number of Topics per Document	numTopicsPerDoc	int	Mandatory. Number of most relevant topics to be listed per document Default: 1
Number of Keywords per Topic	numKeywordsPerTopic	int	Mandatory. Number of keywords to be listed per topic Default: 15

Name	ID	Type	Description
Number of Features	numFeatures	int	Maximum number of key-words to be extracted from documents in total

Input

Parameter	Type	Description
input	message	A message whose body is the input information sent to the service. The header can optionally set the configuration parameters. Configuration parameters can be either set from this message header or from the configuration dialog of the operator. If at least one configuration parameter is set in the message header then the parameters from the configuration dialog are ignored. The input files need to be raw text format. At least two documents must be submitted. The documents need to be wrapped in an archive file. Accepted file extensions: zip, tar, gz, or tgz.

The archive file path can be passed in the message body. For example:

header	body
<code>{}</code>	<code>"/path/to/file/sample.zip"</code>

And with the same configuration parameters:

header	body
<code>{"options": {"numTopics":20, "numTopicsPerDoc":1, "numKeywordsPerTopic":15}}</code>	<code>"/path/to/file/sample.zip"</code>

Alternatively, the file content of the input archive can be passed as blob in the message body. In this case, it is required to include the file name as an attribute with name `storage.filename`. For example, together with the optional configuration parameters:

header	body
<pre>{ "storage.filename": "/path/to/file/sample.zip", "options": { "numTopics": 20, "numTopicsPerDoc": 1, "numKeywordsPerTopic": 15 } }</pre>	Serialized file as blob.

Output

Output	Type	Description
output	message	The inference result as a JSON string. For each input document the detected topic IDs are listed. And for each detected topic its score and keywords are listed, per document.

Example:

```
{
  "_id": "402a582a-c18e-4466-b8c0-085f5c244da6",
  "predictions": [
    {
      "docName": "doc1",
      "topics": [
        0,
        6
      ],
      "scores": [
        0.14,
        0.15
      ],
      "keywords": [
        [
          "kw1",
          "kw2",
          "kw3"
        ],
        [
          "kw10",
          "kw21",
          "kw32"
        ]
      ]
    }, ...
  ],
  "request": {
    "files": [
      "sample.zip"
    ],
    "options": {
      "numTopics": 3,
      "numTopicsPerDoc": 2,
      "numKeywordsPerTopic": 4,
      "numFeatures": 100
    }
  },
  "status": "DONE"
}
```

```
}
```

Error: If an error happens due to an input, this port outputs a message. If this port is not connected, a failure will stop the graph with an error.

2.20.5 Move File (Deprecated)

The Move File operator is used to move (rename) files in a file service. The expected input message is specified in the Input section in this topic.

i Note

This operator is deprecated. Use the operators [Read File \[page 303\]](#), [Write File \[page 309\]](#) and [Remove File \[page 306\]](#) to perform the same action.

Additionally, [List Files \[page 296\]](#) can be used to move directories.

This operation is recursive, meaning it will move any files under the given source path. Interpretation of a path may differ on service as a directory (ADLS, file, HDFS, WebHDFS) or as a prefix (GCS, S3, WASB).

If destination is a file, it will be overwritten; if it is a non-empty directory, it will fail.

Supported services are:

- [System Management Files \(/files\)](#)
- [Microsoft Azure Data Lake \(ADL\)](#)
- [Google Cloud Storage \(GCS\)](#)
- [Hadoop Distributed File System \(HDFS\)](#)
- [Amazon Simple Storage Service \(S3\)](#)
- [Microsoft Azure Storage Blob \(WASB\)](#)
- [WebHDFS](#)
- [Alibaba Cloud Object Storage Service \(Alibaba Cloud OSS\)](#)

i Note

Use `Connection Management` for defining connections over the manual option.

In the given file structure:

```
.
|
+-- a
|   +-- file1.txt
|   +-- file2.txt
+-- b
    +-- f1.txt
    +-- f2.txt
```

- Moving source `a/file1.txt` to destination `newfile.txt` would succeed because the destination does not exist.
- Moving source `a/file1.txt` to destination `b/f1.txt` would succeed and overwrite `b/f1.txt` because the destination is an existing file.

- Moving source `a/file1.txt` to destination `b/` would fail because `b/` already exists and is not empty.
- Moving source `a/` to destination `b/` would fail because `b/` already exists and is not empty.
- Moving source `a/` to destination `b/dir/` would succeed because `b/dir/` does not exist.

Configuration Parameters

Name	ID	Type	Description
Service	service	string	The file service to operate. Additional parameters may depend on the selected service. Default: "file"
Connection	connection	object	Holds information about connection information for the services. Only use manual connections when using a connection ID is not possible. Each service connection parameter is documented separately: • Microsoft Azure Data Lake (ADL) • Google Cloud Storage (GCS) • Hadoop Distributed File System (HDFS) • Amazon Simple Storage Service (S3) • Microsoft Azure Storage Blob (WASB) • WebHDFS • SDL • Alibaba Cloud Object Storage Service (Alibaba Cloud OSS)

Name	ID	Type	Description
Configuration Type	configurationType	string	<i>connection parameter</i> Which type of connection information will be used: Manual (user input), or retrieved by the Connection Management Service. i Note It is recommended to choose <i>Configuration Manager</i> as the configuration type and select an existing <i>Connection ID</i>. The <i>Manual</i> configuration type must be used only for testing purposes. Default: ""
Connection ID	connectionID	string	<i>connection parameter</i> The ID of the connection information to retrieve from the Connection Management Service. Default: ""
Connection Properties	connectionProperties	object	<i>connection parameter</i> All the connection properties for manual input.
Timeout (ms)	timeoutInMs	int	Sets the time limit to execute the operation. If 0, no timeout is used. Default: 0
Retry Period (ms)	retryPeriodInMs	int	The time interval in milliseconds between connection trials. Default: 0
Retry Attempts	numRetryAttempts	int	The number of times to retry a connection. Default: 0

Name	ID	Type	Description
Simultaneous Requests	simultaneousRequests	int	The number of simultaneous requests generated on recursive calls (only available for GCS, S3, and WASB). Default: 1
Stop Request on Error	stopRequestOnError	boolean	Controls whether simultaneous requests from recursive calls should stop at first error (only available for GCS, S3, and WASB). Default: false
Terminate on Error	terminateOnError	boolean	A flag that indicates whether or not the graph should stop if this operator encounters an error. If set to <code>false</code>, an output may represent an error, in which case its attribute <code>message.error</code> will be <code>true</code> and its body will contain a string describing the error. Default: true

Input

Input	Type	Description
in	message	A message whose body has the source file path and destination. Expected message attributes: <i>storage.src</i> Mandatory. Path to source file or directory. <i>storage.dst</i> Mandatory. Path to destination file or directory (should match types with source). The destination path may contain <dirname> and <basename>, referring to the given source path's dirname and basename. For example, <code>move src:data/A/user.csv dst:<dirname>/processedFiles/<basename></code> would map to the destination <code>data/A/processedFiles/user.csv</code>. <i>storage.srcBucket</i> Overrides <code>bucket</code> from configuration on service GCS and S3 for the source file or directory. <i>storage.dstBucket</i> Overrides <code>bucket</code> from configuration on service GCS and S3 for the destination file or directory.

Output

Output	Type	Description
out	message	A message whose header <code>message.error</code> (bool) reports whether the operation was successful. The body and any other header from the input message are copied to this one.

2.20.6 PA Automated (Deprecated)

This operator applies the PA Automated Analytics product for prediction of values. Note that a pre-trained model for PA Automated Analytics must exist and reside in a folder relative to the <VFLOW_REPO>/blobs directory.

Note

This operator is deprecated. Please check SAP Note [2739161](#) for more information.

Configuration Parameters

Parameter	Type	Description
modelPath	string	Mandatory. Contains the path relative to <VFLOW_REPO>/blobs/ where the kxen models are stored, for example, the KxAdmin.txt file and the model files. Default: ""
modelName	string	Mandatory. The name of the model that should be deployed. Must be contained in the KxAdmin.txt file, otherwise the operator will fail. Default: ""
operatingMode	string	Mandatory. Operating mode that is either <code>stream</code> or <code>resultfile</code> . Default: "stream"
schemaDescFile	string	Kxen-schema description file for the incoming strings or CSV input file. Default: ""
streamCacheSize	integer	If in stream mode, this specifies the size of the internal stream chunk cache. Default: 10000

Input

Input	Type	Description
<code>inDataCSVPath</code>	string	Local path of the input data CSV file.
<code>inData</code>	string	Input port for streaming data.

Output

Output	Type	Description
<code>outResultContent</code>	string	If <code>operatingMode</code> is "stream", this port streams the result line by line. If the <code>operatingMode</code> is "resultfile", the output will be the filepath to the result file.

2.20.7 Read File (Deprecated)

The Read File operator is used to read a file or periodically poll a directory for its contents in a storage service.

i Note

This operator is deprecated.

Please use [Read File \[page 303\]](#) instead, possibly preceded by a [List Files \[page 296\]](#) if dealing with directories.

i Note

Limitation: The operator cannot browse at the root level of connections using a bucket or container, for example, S3, GCS, and WASB connections with root path / currently are not supported.

Supported services are:

- [System Management Files \(/files\)](#)
- [Microsoft Azure Data Lake \(ADL\)](#)
- [Google Cloud Storage \(GCS\)](#)
- [Hadoop Distributed File System \(HDFS\)](#)
- [Amazon Simple Storage Service \(S3\)](#)
- [Microsoft Azure Storage Blob \(WASB\)](#)
- [WebHDFS](#)
- [Alibaba Cloud Object Storage Service \(Alibaba Cloud OSS\)](#)

Note

Use Connection Management for defining connections over the manual option.

The operation takes only one input parameter: the path of the file. This is given as a string in the body of a message in the `inPath` port. If no input is connected to the port, the operator will periodically poll from the configured path.

The file content is output as the body of a message in the `outFile` port. Further details of the operation are reported as headers of the message, as listed in the port documentation.

An example of usage is given in the `com.sap.demo.file` graph.

Polling directories: When the given path points to a directory, this operator will poll all files inside that directory. It may be recursive if set to do so.

Caution

Reading large (GB) files in a single chunk may cause memory issues (OOMKilled). Use the `chunkSize` parameter in such cases.

Configuration Parameters

Name	ID	Type	Description
Service	service	string	The file service to operate. Additional parameters may depend on the selected service. Default: "file"

Name	ID	Type	Description
Connection	connection	object	Holds information about the service's connections. Only use Manual connections when using a Connection ID is not possible. Each service connection is documented separately: • Microsoft Azure Data Lake (ADL) • Google Cloud Storage (GCS) • Hadoop Distributed File System (HDFS) • Amazon Simple Storage Service (S3) • Microsoft Azure Storage Blob (WASB) • WebHDFS • SDL • Alibaba Cloud Object Storage Service (Alibaba Cloud OSS)
Connection ID	connectionID	string	<i>connection parameter</i>: The ID of the connection information to retrieve from the Connection Management Service. Default: ""
Connection Properties	connectionProperties	object	<i>connection parameter</i>: All the connection properties for the selected service for manual input.
Path	path	string	A directory to be polled (ends with /) or a file to be read. This only applies if inPath is not connected. Default: "/tmp/test.txt"

Name	ID	Type	Description
Delete After Send	deleteAfterSend	bool	A flag that indicates whether the file should be deleted after its contents have been sent. ⚠ Caution There is no guarantee that the operator connected to the output receives the filename or content of the file. i Note If <code>path</code> is a directory, the files in it will be removed after being read, but the directory itself will remain. When in recursive mode, subdirectories will also be kept. Default: false

Name	ID	Type	Description
Chunk Size	chunkSize	string	The maximum number of bytes that can be read from files at once. It reads the bytes in blocks until it reaches the end of the file. This can be used to reduce graph latency and memory usage. If chunkSize is zero, files are read in a single chunk. Otherwise, it will be broken in chunks with a maximum size chunkSize. It may be dynamically customized through the message header <code>storage.chunkSize</code>. This field allows metric prefixes to be used and an optional "i" to indicate binary bases. For example: • "0", "0B": unlimited • "1", "1B": 1 byte • "2kb", "2KB": 2*103 bytes • "2kib", "2KiB": 2*210 bytes • "3mb", "3MB": 3*106 bytes • "3mib", "3MiB": 3*220 bytes Prefixes for Giga (G), Tera (T), and Peta (P) are also supported. Default: "0"
Retry Attempts	numRetryAttempts	int	The number of times to retry a connection. Default: 0
Retry Period (ms)	retryPeriodInMs	int	The time interval in milliseconds between connection trials. Default: 0

Name	ID	Type	Description
Poll Period (ms)	pollPeriodInMs	int	The time interval in milliseconds between successive polls. If no interval is needed, the value 0 should be used. Default: 1000
Batch Read	batchRead	bool	A flag that controls whether all files should be read in batches. If set to True and a directory is a file, then it should be read in batches. If set to True and a directory is being polled, then outFilename is given a list with one filename per line polled, then outFilename is given a list with one filename per line (may be empty when using onlyReadOnChange); separated by CRLF (may be empty when using onlyReadOnChange). Default: false
Recursive	recursive	bool	A flag that controls whether a directory listing should recursively include all sub-directories. Default: false

Name	ID	Type	Description
Pattern	pattern	string	A regular expression used to filter file paths before reading them. If empty, all files are accepted. The expression is applied to a path after being converted to an absolute one only if one of the following is true: • It came from <code>inPath</code> and points to a regular file (not a directory). • It is a file found in the polled directory, regardless of whether the directory was specified by <code>inPath</code> or <code>in path</code>. For instance: the expression <code>^\/?(\w+\w/)*[A-Za-z]\w+\.txt\$</code> would match any path pointing to a file whose name starts with an alphabetic letter and ends with the extension <code>.txt</code>, such as <code>/foo/bar/file.txt</code> and <code>bar/file.txt</code>, but not <code>file.csv</code> nor <code>/foo/bar/file</code>. The expression <code>(.txt) (.csv)\$</code> would match any file with <code>.txt</code> or <code>.csv</code> extension. Default: ""

Name	ID	Type	Description
Only Read on Change	onlyReadOnChange	bool	If true, only outputs a file if it is new or changed, which avoids the repetitive reading of unchanged files. This uses the date and time given by the file system as the latest modification time as opposed to the actual file contents. If using <code>batchRead</code>, a message with an empty body is outputted to <code>outFilename</code>, if no files were read, representing the empty list of files. Default: false
Terminate on Error	terminateOnError	boolean	A flag that indicates whether or not the graph should stop if this operator encounters an error. If set to <code>false</code>, an output may represent an error, in which case its attribute <code>message.error</code> will be <code>true</code> and its body will contain a string describing the error. This applies to whichever output port is connected. Default: "true"

Name	ID	Type	Description
Configuration Type	configurationType	string	connection parameter: Which type of connection information will be used: Manual (user input) or retrieved by the Connection Management Service. i Note It is recommended to choose Configuration Manager as the configuration type and select an existing Connection ID. The Manual configuration type must be used only for testing purposes. Default: ""

Input

Input	Type	Description
inPath	message	A message whose body is the path (relative or absolute) of a file or directory (ends with /) to be read. When reading a single file, the message header <code>storage.offset</code> may be set to read a specific chunk from a file, which is also subject to the <code>chunkSize</code> configuration.

Output

Output	Type	Description
outFilename	message	A message whose body is the path of the file. This will be equal to the path that prompted the reading (either <code>inPath</code> or <code>path</code>).

Output	Type	Description
outFile	message	A message whose headers describe the file read and whose body contains the file's contents as a blob. The message contains the following headers: • <i>storage.chunkIndex</i> (type int). The current chunk's index, starting from zero. • <i>storage.chunkCount</i> (type int). The total number of chunks for this file. • <i>storage.fileSize</i> (type int64). The size of the file, in bytes. • <i>storage.endOfFile</i> (type bool). A flag that indicates whether this chunk is the last (this is simply a convenience: <code>endOfFile == (chunkIndex == chunkCount-1)</code>). • <i>storage.filename</i> (type string). The name of the file, including extension (if any), but excluding its directory. • <i>storage.directory</i> (type string). The absolute path of the directory where the file resides. • <i>storage.path</i> (type string). The absolute path of the file (equivalent to <code><directory>/<filename></code>). • <i>storage.polledDirectory</i> (type string). The absolute path to the directory being polled, if applicable. • <i>storage.pathInPolledDirectory</i> (type string). The file path relative to <code>polledDirectory</code>. This is the result of subtracting <code>polledDirectory</code> from <code>directory</code>. • <i>storage.polledDirectorySize</i> (type int64). The total size of the directory, in bytes, if applicable. • <i>storage.fileIndex</i> (type int64). The index of the current file relative to a

Output	Type	Description
		single request. Starts at 0 and is set to 0 if reading a single file. • <code>storage.fileCount</code> (type int64). The total number of files being read relative to a single request. Is set to 1 if reading a single file. • <code>storage.endOfSequence</code> (type bool). A flag that indicates if that is the last file of a given request (this is simply a convenience: <code>endOfSequence == (fileIndex == fileCount-1)</code>). • <code>message.error</code> (type bool). A flag that indicates whether the operation failed.

2.20.8 Remove File (Deprecated)

The Remove File operator is used to remove files in a storage service.

i Note

This operator is deprecated.

Please use [Remove File \[page 306\]](#) instead.

This operation is recursive, meaning it will remove any files under the given path. If the path points to a directory, the directory itself is removed as well.

Supported services are:

- [System Management Files \(/files\)](#)
- [Microsoft Azure Data Lake \(ADL\)](#)
- [Google Cloud Storage \(GCS\)](#)
- [Hadoop Distributed File System \(HDFS\)](#)
- [Amazon Simple Storage Service \(S3\)](#)
- [Microsoft Azure Storage Blob \(WASB\)](#)
- [WebHDFS](#)
- [Alibaba Cloud Object Storage Service \(Alibaba Cloud OSS\)](#)

i Note

Use `Connection Management` for defining connections over the manual option.

Configuration Parameters

Name	ID	Type	Description
Service	service	string	The file service to operate. Additional parameters may depend on the selected service. Default: "file"
Connection	connection	object	Holds information about connection information for the services. Only use manual connections when using a connection ID is not possible. Each service connection parameter is documented separately: • Microsoft Azure Data Lake (ADL) • Google Cloud Storage (GCS) • Hadoop Distributed File System (HDFS) • Amazon Simple Storage Service (S3) • Microsoft Azure Storage Blob (WASB) • WebHDFS • SDL • Alibaba Cloud Object Storage Service (Alibaba Cloud OSS)

Name	ID	Type	Description
Configuration Type	configurationType	string	<i>connection parameter</i>: Which type of connection information will be used: Manual (user input) or retrieved by the Connection Management Service. i Note It is recommended to choose <i>Configuration Manager</i> as the configuration type and select an existing <i>Connection ID</i>. The <i>Manual</i> configuration type must be used only for testing purposes. Default: ""
Connection ID	connectionID	string	<i>connection parameter</i>: The ID of the connection information to retrieve from the Connection Management Service. Default: ""
Connection Properties	connectionProperties	object	<i>connection parameter</i>: All the connection properties for the selected service for manual input.
Terminate on Error	terminateOnError	boolean	A flag that indicates whether or not the graph should stop if this operator encounters an error. If set to false, an output may represent an error, in which case its attribute message.error will be true and its body will contain a string describing the error. Default: "true"

Name	ID	Type	Description
Timeout (s)	timeoutInMs	int	Sets the time limit to execute the operation. If 0, no time-out is used. Default: 0
Retry Period (s)	retryPeriodInMs	int	The time interval in milliseconds between connection trials. Default: 0
Retry Attempts	numRetryAttempts	int	The number of times to retry a connection. Default: 0
Simultaneous Requests	simultaneousRequests	int	The number of simultaneous requests generated on recursive calls (only available for GCS, S3, and WASB). Default: 1
Stop Request on Error	stopRequestOnError	boolean	Controls whether simultaneous requests from recursive calls should stop at first error (only available for GCS, S3, and WASB). Default: false

Input

Input	Type	Description
in	message	A message whose body is the path (relative or absolute) of a file or directory to be removed.

Output

Output	Type	Description
out	message	A message whose header <code>message.error</code> (bool) reports whether the operation was successful. The body and any other header from the input message are copied to this one.

2.20.9 SAP ABAP ODP Object Consumer (Deprecated)

Reads from an ABAP ODP object.

It uses Flowagent subengine for execution, and you need to connect it to a Flowagent-based producer operator (for example, [Flowagent CSV Producer \[page 140\]](#)).

! Restriction

This operator is deprecated and will be removed soon.

Please use its successor [ABAP ODP Reader \[page 46\]](#).

Refer to SAP Note [2739161](#)  for more information.

Configuration Parameters

Parameter	Type	Description
ABAP Connection (protocol RFC only)		
ODP Object	string	ODP Object (for example, / EXTRACTORS/<ODP Context>/ <Extractor Name>).
Trace Level	string	
Fetch Size	integer	Indicates the number of rows that will be fetched from the source in each request. For wide tables, smaller values are better. For tables with fewer columns, larger values are better. Default: 100

ODP Filters

After importing the ODP source, you can also edit the filters in the UI. During metadata import, the source will provide the type of filters available or possible operators for each column.

User will need to select the column, choose the operator (= or BETWEEN) then provide the values.

The filters will be applied during the graph run. Filters is available in both mode, FULL or DELTA.

Delta Processing

ODP object can provide delta records in different ways. The delta record will provide a row type like Insert (I), Update (U), Delete (D), Before Image (B), Unknown (X) etc. User can either decide to keep a history of changed records (Track Change History) mode, or decide to apply the changes directly to the target (Apply Change Data) mode. These options are exposed on the operators that support such feature.

By default, Flowagent CSV and File producer support only Track Change History mode, while Flowagent Table Producer with HANA as selected service, will support both modes. For HANA Table Producer, user can select Apply Change Data in the applier type, this will cause Flowagent to apply the changes to the target table, so I,U,D will properly be applied to the target.

Based on your extractor source type, you can either use Track Change History mode to keep a history or changed records or Apply Change Data mode to apply these delta changes to the target system. Below lists some of the ODP Object types and their support in Delta + Apply Change Data support.

Delta Type	Description	Apply Change Data Support
ADD	Additive Extraction (for example, LIS Info Structures)	Not supported with HANA Producer in Apply Change Data Mode. All delta rows will be ignored
AIE	After-Images (FI-GL/AP/AR)	Delete is not send by the source, hence target data won't get deleted.
AIED	After-Images with Deletion Flag (FI-GL/AP/AR)	Supported.
NEWE	Only New Records (Inserts) (Cube-Compatible)	Supported.
ODS	ODS Extraction	Supported.

Delta Columns

ABAP ODP consumer add the following columns to the original table.

Column Name	Description	Source
DI_SEQUENCE_NUMBER	Contains an integer for sequencing.	ODP Consumer Operator
DI_OPERATION_TYPE	Contains the row operation type: I (insert), D (delete), X (unknown), and the sort.	ODP Consumer Operator
ODQ_CHANGEMODE	For CDC, identifies the row type. Works in conjunction with ODQ_ENTITYCNTR to determine how to handle the row. C (create), U (update), D (delete).	SAP
ODQ_ENTITYCNTR	For CDC, an integer that works in conjunction with ODQ_CHANGEMODE to determine how to handle row.	SAP

The following table describes the relationships between ODQ_CHANGEMODE and ODQ_ENTITYCNTR.

CHANGEMODE	ENTITYCNTR	Description
C	+1	The record is new and inserted.
U	+1	The record is inserted.If there is a record with same key already available, the record is updated. In this case, there should be a corresponding U- (short notation for U and -1) record. The record contains the new information.
U	-1	The record is deleted and replaced with the corresponding record with same key and U+ (short notation for U and +1) if available. The record contains the previous (old or outdated) information.
U	0	The record is updated, but the key figures must be added to the already available key figures. So the record returns the difference between the old record and the new record, not the new record itself.
D	-1	The record is deleted. It contains the same key as an already existing record, but the key figures are inverted. So the record might be "deleted" by adding the key figures to the already available values. They should sum up to 0.

CHANGEMODE	ENTITYCNTR	Description
D	+1	The record is deleted. It contains the actual data in both key and key figures.

Delta Recovery

In case of graph failures or recovery, user can get the composite request id from ODQ Monitor (/odqmon) and choosing the subscription id they have provided during the graph creation. The composite request id is a timestamp of job last ran, user can then edit the graph and provide the value in the "Extraction from Timestamp" field to signal the timestamp where Flowagent should read the records from.

i Note

Only in case of `Apply Change Data` mode, the data will be consistent with source system. In case of `Track Change History` the target can contain duplicate records.

Input

Input	Type	Description
inTableName	string	ODP object name (for example, /EXTRACTORS/<ODP Context>/<Extractor Name>).

i Note

This is optional. If ODP Object Name is presented, this port is ignored.

Output

Output	Type	Description
<code>outConfig</code>	<code>string.flowagent.config</code>	Flowagent type port, which only <code>Flowagent</code> producers can consume (for example, <code>Flowagent File Producer</code>). In order to get the data as <code>string</code> , <code>Flowagent CSV Producer</code> can be used as producer.
<code>outError</code>	<code>string</code>	Operator error. If mapped, the operator error is sent to the operator this port is mapped to and the graph remains running. If not mapped, the graph terminates with the operator error.

2.20.10 Spark Data Transform (Deprecated)

The Spark Data Transform operator provides a wide variety of options to meet your data transformation needs.

i Note

This operator is deprecated. Use the [Data Transform \[page 495\]](#) operator instead.

There are nodes available in the operator that provides capabilities to meet your data transformation requirement. For example, you can use these nodes to create aggregations, projections, joins, unions, and more. Configure each node to meet your individual data specifications.

This operator has one input port (`input`) and two output ports (`output` and `error`). Use this operator in a graph that uses only other data workflow operators.

Connect the input port to another data workflow operator to trigger its start. On the `error` output port, the engine writes a message in case of errors when trying to execute the transformation operation. On success, the engine writes a message on the `output` port. If you want the execution of the graph to proceed, connect the respective output ports to other data workflow operators.

i Note

Any message written to an unconnected output port results in a graph failure.

Configuration Parameters

The configuration parameters for the Spark Data Transform operator is node-specific. For more information on how to configure individual nodes in the operator, see [Transform Data](#).

Input

Input	Type	Description
input	string	Accepts message from a connected data workflow operator.

Output

Output	Type	Description
output	string	If the operator executed successfully, this port carries the success message.
error	string	Operator error. If the port is connected, then the graph will keep running. If it is not connected, the graph will terminate with the operator error.

For more information, see [Transform Data](#).

2.20.11 Text Analysis Connector (Deprecated)

Allows you to run various natural language processing tasks on unstructured textual data by using the Vora text analysis server.

Note

This operator is deprecated. Please check SAP Note [2739161](#) for more information.

The operator receives a request for text analysis in JSON format. There are two types of requests:

1. **Data:** The content of the document to be analyzed is included in the request. The operator outputs the result of the analysis as a string in CSV format.
2. **Folder:** The request includes a folder name with the location of the documents to be analyzed. The result of the analysis is written directly in the specified location in two files `[folder_name]_TA.csv` and `[folder_name]_TADOC.csv`.

A sample graph called **TextAnalysis Example** (`com.sap.textanalysis.example`) and **TextAnalysis Files** (`com.sap.textanalysis.files`) using this operator can be found in the graph library.

See the Text Analysis section of the Developer Guide for SAP Vora for more information about the text analysis configurations and output format.

Request Parameters

Parameter	Description
endpoint	Host name and port number of the vora-textanalysis service. For example: 'vora-textanalysis:10002'.
taconfig	Text Analysis default configuration. One of: • LINGANALYSIS_BASIC • LINGANALYSIS_STEMS • LINGANALYSIS_FULL • EXTRACTION_CORE • EXTRACTION_CORE_ENTERPRISE • EXTRACTION_CORE_PUBLIC_SECTOR • EXTRACTION_CORE_VOICEOFCUSTOMER If empty, LINGANALYSIS_BASIC is used.
languages	Optional A list of languages used for language detection specified in ISO 639-1 codes. For example: 'EN, DE, ES'. If no language is specified, then automatic detection is attempted.
mime_type	Optional. The type of input documents. Allowed values are: • 'text/plain' • 'text/html' • 'text/xml' • 'text' The value 'text' indicates that the input is one of plain text, HTML or XML. If not set, or if value is 'text', document identification and conversion are performed. Note that the analysis of documents in binary formats is also supported, but automatic format detection is used in this case.
text_encoding	Optional. If the document contains text, this parameter indicates the encoding. For example: 'UTF-8'. If not set and the MIME type indicates text, encoding detection and conversion are performed.

Parameter	Description
document	The text of the document to be analyzed. i Note Relevant for 'data' type of request.
service	A string with the connection parameters to the remote file storage service (See <code>resource_string</code> in the Connection Types for DATASOURCE and EXPORT without Connection Management page of the SQL Reference for SAP Vora in SAP Data Intelligence for formatting information).. i Note Relevant for 'folder' type of request.
recursive_flag	true/false. If true the analysis is done recursively in the sub-folders in the specified location. Output files are written locally in each subfolder. i Note Relevant for 'folder' type of request.
document_id	true/false. The starting ID for identifying documents in the result. i Note Relevant for 'folder' type of request.
folderpath	The location of the files to be analyzed. Currently supported file systems are local and hdfs. For example: ' /tmp/folder ' 'hdfs://vora-hdfs:8020/user/hdfs/folder ' i Note Relevant for 'folder' type of request.

Configuration Parameters

Name	ID	Type	Description
CmdLine	CmdLine	string	The command line of the process to be forked (should not be modified). Default: "/project/textanalysis/bin/v2textanalysis_vflow"
Env	Env	object	Additional environment variables to be provided to the process. Default: "[]"
InheritEnv	InheritEnv	boolean	If set to true, the forked process inherits the environment. Default: "false"
TraceSderr	TraceSderr	boolean	If set to true, the standard error of the forked process is traced to the parent process with severity. Default: "false"

Input

Input	Type	Description
inFileData	string	A request including the document to be analyzed.
inFolderPath	string	A request including a folder to be processed.

Output

Output	Type	Description
outFileData	string	Result of the analysis of the document received through the inFileData port, in CSV format.
outFolderPath	string	A string containing the input request specification and the number of successfully analyzed documents.

2.20.12 Write File (Deprecated)

The Write File operator writes files to a storage service.

i Note

This operator is deprecated.

Please use [Write File \[page 309\]](#) instead.

Supported services are:

- [System Management Files \(/files\)](#)
- [Microsoft Azure Data Lake \(ADL\)](#)
- [Google Cloud Storage \(GCS\)](#)
- [Hadoop Distributed File System \(HDFS\)](#)
- [Amazon Simple Storage Service \(S3\)](#)
- [Microsoft Azure Storage Blob \(WASB\)](#)
- [WebHDFS](#)
- [Alibaba Cloud Object Storage Service \(Alibaba Cloud OSS\)](#)

i Note

Use `Connection Management` for defining connections over the manual option.

The operation takes only one input parameter: the content of the file. The target file is configured in the operator's `path`. To dynamically change the target, see *Path formatting* below.

Configuration Parameters

Name	ID	Type	Description
Mode	mode	string	Controls whether the target file should be appended to, created (avoiding overwrites, truncated if it already exists), or overwritten (created if it does not exist). It may be dynamically set through the message header <code>storage.writeMode</code>. Default: "append"
Connection	connection	object	Holds information about connection information for the services. Only use manual connections when using a Connection ID is not possible. Each service connection parameters is documented separately: • Microsoft Azure Data Lake (ADL) • Google Cloud Storage (GCS) • Hadoop Distributed File System (HDFS) • Amazon Simple Storage Service (S3) • Microsoft Azure Storage Blob (WASB) • WebHDFS • SDL • Alibaba Cloud Object Storage Service (Alibaba Cloud OSS)

Name	ID	Type	Description
Configuration Type	configurationType	string	<i>connection parameter</i> i Note It is recommended to choose : Which type of connection information will be used: Manual (user input) or retrieved by the Connection Management Service. <i>Configuration Manager</i> as the configuration type and select an existing <i>Connection ID</i>. The <i>Manual</i>: Which type of connection information will be configuration type must be used only for testing purposes. Default: ""
Connection ID	connectionID	string	<i>connection parameter</i>: The ID of the connection information to retrieve from the Connection Management Service. Default: ""
Connection Properties	connectionProperties	object	<i>connection parameter</i>: All the connection properties for the selected service for manual input.
Path	path	string	A formatted string describing the output path for files. See Path formatting below for details and examples. Default: "/tmp/file_\<counter\>.txt"< p=""> </counter\>.txt"<>
Retry Attempts	numRetryAttempts	int	The number of times to retry a connection. Default: 0

Name	ID	Type	Description
Retry Period (ms)	retryPeriodInMs	int	The time interval in milliseconds between connection trials. Default: 0
Terminate on Error	terminateOnError	boolean	A flag that indicates whether the graph should stop if this operator encounters an error. If set to False, an output may represent an error, in which case its attribute message.error will be True and its body will contain a string describing the error. Default: "true"
Service	service	string	The file service to operate. Additional parameters may depend on the selected service. Default: "file"

Input

Input	Type	Description
inFile	message	A message whose body (blob) will be written to a file. There are no requirements on the message's headers other than those referred to in the path and mode configuration parameters. Default:

Output

Output	Type	Description
outFilename	message	A message whose body is the path to the file to which content is written or appended. Whether this path is relative or absolute depends on how it was given to the <code>path</code> configuration. The header <code>message.error</code> (bool) reports whether the operation was successful. Any other header from input is copied to this message.

Path Formatting

Strings in the `path` configuration are subject to the following rules:

- Schemes can be invoked using angle brackets: The string `<foo>` will be replaced by the result of the scheme named "foo". Available schemes are:
 - `counter`: an incremental integer
 - `date`: the current local date in the format `YYYYMMDD`
 - `time`: the current local time in the format `HHMMSS`Any other (unrecognized) scheme names will cause an error.
- Message headers can be queried using `\${bar}`. For example, `\${bar}` would be replaced by the value of header "bar" in the message given to `inFile`. Note that the dollar sign must always be escaped with a backslash, otherwise it will be seen as a substitution parameter.
 - A default value can be set using an equal sign: `\${bar=lorem}` will be replaced by the value "lorem" whenever the input message lacks the "bar" header. If no default value is set and the message is missing the header, an error will be thrown.
 - It is necessary to escape the dollar sign with a backslash (for example, `\${bar}`) to prevent the Modeler from interpreting it as a substitution parameter.
 - Anything else (that is not between `<` and `>` or `\${` and `}`) will be left untouched.

Limitations

- The following characters cannot appear in scheme or message header names: `<>${}`.
- Empty scheme (`<>`) or header (`\${}`) names will be left untouched.

Example: Basic Usage

Suppose you have messages coming from a Kafka Consumer whose topic describes the type of sensor that sent the message. Then you can use a path like this:

```
mydir_<date>_<time>/\${kafka.topic}.csv
```

It produces output files like this:

```
mydir_20170131_234550/noise.csv
```

```
mydir_20170201_080010/temperature.csv  
mydir_20170201_080010/humidity.csv
```

Example: Copying Directories

If we want to reproduce an entire directory structure, we can set a File, HDFS, or S3 Consumer to poll this directory and use the `storage.pathInPolledDirectory` header to refer to each file's location in it:

```
outputDir/\${storage.pathInPolledDirectory}
```

can be expanded to, for example:

```
outputDir/YearReport.docx  
outputDir/January/sales.pdf  
outputDir/January/finance.xlsx  
outputDir/February/sales.pdf
```


3 SAP Data Intelligence Graphs

A graph is a network of operators connected to each other using typed input ports and output ports for data transfer. Users can define and configure the operators in a graph.

For information about using SAP Data Intelligence graphs and creating your own custom graphs, see [Creating Graphs](#) in the *Modeling Guide for SAP Data Intelligence*.

Category	Graph	Technical Name	Primary Use Case	Status
Connectivity	Kafka [page 624]	com.sap.demo.kafka	Generate arbitrary data and pass it to the Kafka producer. The Kafka consumer reads the data and writes it to a terminal.	
Connectivity	OpenAPI GreeterClient Demo [page 624]	com.sap.demo.open-api.client.greeter	Call arbitrary REST services from a graph.	
Connectivity	OpenAPI PlainClient Demo [page 625]	com.sap.demo.open-api.client.plain_greeter	Call arbitrary REST services from a graph.	
Connectivity	OpenAPI Greeter Demo [page 625]	com.sap.demo.open-api.server.greeter	Expose REST services from a graph.	
Connectivity	OpenAPI Plain Greeter [page 626]	com.sap.demo.open-api.server.plain_greeter	Expose REST services from a graph.	
Connectivity		com.sap.demo.open-api.server.odata_people	Use an openapi.server to expose an OData service endpoint.	
Connectivity	S3 [page 626]	com.sap.demo.s3	Produce arbitrary data from the data generator.	
Connectivity	Google Pub/Sub [page 627]	com.sap.demo.google-pubsub	Publish messages using the Google Pub/Sub Producer, and consume them via the Google Pub/Sub Consumer.	
Examples	Data Generator [page 631]	com.sap.demo.data-generator	Generate arbitrary values from the data generator simulating a set of sensors.	
Examples	Go Data Generator [page 638]	com.sap.demo.goDataGenerator	Generate arbitrary values from the data generator simulating a set of sensors.	

Category	Graph	Technical Name	Primary Use Case	Status
Examples	Message Generator [page 645]	com.sap.demo.messageGenerator	Generate arbitrary values packed into a message, simulating a set of sensors.	
Examples	Go Message Generator [page 638]	com.sap.demo.goMessageGenerator	Generate arbitrary values from the data generator packed into a message, simulating a set of sensors.	
Examples	File System [page 636]	com.sap.demo.file	Generate arbitrary data from the data generator.	
Examples	HDFS [page 640]	com.sap.demo.hdfs	Generate arbitrary data from the data generator.	
Examples	S3 [page 650]	com.sap.demo.s3	Generate arbitrary data from the data generator.	
Examples	GCS [page 637]	com.sap.demo.gcs	Generate arbitrary data from the data generator.	
Examples	Storage [page 649]	com.sap.demo.storage	Generate arbitrary data from the data generator.	
Examples	Kafka [page 643]	com.sap.demo.kafka	Generate arbitrary data from the data generator and pass it to the Kafka producer.	
Examples	Histogram Simple Example [page 641]	com.sap.demo.util.histogramSimple	Generate random numbers that are sent to the Histogram operator, which produces the histogram of its inputs as a Message, with an integer array on its body, and the range of the interval on its attributes.	
Examples	HTML Viewer [page 641]	com.sap.demo.htmlViewer	Simulate a Sales Campaign generating pseudo-random data for a large number of stores.	
Examples	Load into SAP HANA [page 644]	com.sap.demo.hana	Produce sensor data as a CSV string.	

Category	Graph	Technical Name	Primary Use Case	Status
Examples	Vora SQL Console [page 651]	com.sap.demo.vora.client	Use a Vora Client operator to execute SQL commands on SAP Vora.	
Examples	Load into SAP Vora [page 644]	com.sap.demo.vora.hdfs	Produce sensor data that is aggregated by the JavaScript operator.	
Examples	Go Operator and File [page 639]	com.sap.demo.system.goOperator	Read content from a file and uses the Go operator to deal with the file content and name using different go functions and message type in the Go operator.	
Examples	Google Dataproc [page 640]	com.sap.demo.hadoop.dataproc	Submit a Spark job to an existing Google Dataproc cluster, and see the results of the job.	
Examples	Example Counter [page 634]	com.sap.demo.counter	Run tests with this graph, which runs for five seconds and then shuts down with status "completed".	
Examples	Data Workflow [page 632]	com.sap.demo.data-workflow	Start an internal example graph. It serves as an example on how to use the operators from the Data Workflow category.	
Examples	Example Notification [page 635]	com.sap.demo.notification	Use SAP Data Intelligence Notification and Pipeline operators from the Data Workflows category.	
Examples	Data Mask [page 632]	com.sap.demo.data-Mask	Protect personally identifiable or sensitive information by covering all or a portion of the data.	
Examples	DQMm Address [page 633]	com.sap.demo.dqmm.addrCleanse	Create a request from data that can be passed to the DQMm Client operator to be sent to the service.	

Category	Graph	Technical Name	Primary Use Case	Status
Examples	DQMm Reverse [page 634]	com.sap.demo.dqmm.reverseGeo	Create a request from data that can be passed to the DQMm Client operator to be sent to the service.	
Examples	Basic Validation [page 630]	com.sap.demo.validationRule.basicValidationRule	Evaluate input data against a rule.	
Examples	IoT Validation [page 642]	com.sap.demo.validationRule.IoTValidationRule	Model a sensor that tracks the temperature and surface illuminance.	
Examples	RClient Example [page 646]	com.sap.demo.rClientExample	See the different features of the RClient operator.	
Graph Snippets [page 651]	Copy Files (Incremental Load) [page 652]	To be configured in the wizard.	Ingest data from CSV or JSON files stored in a (cloud) storage into a VORA streaming table by first processing all existing files and then monitoring the directory for any new/modified files.	
Graph Snippets [page 651]	Copy Files (Full Load) [page 652]	To be configured in the wizard.	Load data from CSV or JSON files stored in a (cloud) storage into a HANA table and terminate when all files are loaded.	
Graph Snippets [page 651]	Simple Javascript DB-to-File Data Manipulation [page 654]	To be configured in the wizard.	Manipulate data from an SAP HANA database table using plain Javascript (ECMA-Script 2009 / ES5) and writing the manipulated data to a file.	
Graph Snippets [page 651]	Simple Javascript File-to-DB Data Manipulation [page 655]	To be configured in the wizard.	Manipulate data using plain Javascript (without any libraries) and write the manipulated data to an SAP HANA database table.	
Graph Snippets [page 651]	Simple Python File-to-DB Data Manipulation [page 657]	To be configured in the wizard.	Manipulate data using plain Python and write the manipulated data to an SAP HANA database table.	

Category	Graph	Technical Name	Primary Use Case	Status
Graph Snippets [page 651]	Simple R File-to-DB Data Manipulation [page 658]	To be configured in the wizard.	Manipulate data using plain R (using only built-in libraries) and write the manipulated data to an SAP HANA database table.	
Graph Snippets [page 651]	Simple Javascript File Data Manipulation [page 655]	To be configured in the wizard.	Manipulate data using plain Javascript (ECMAScript 2009 / ES5) and writing the manipulated data back to another file.	
Graph Snippets [page 651]	Simple Node.js File Data Manipulation [page 656]	To be configured in the wizard.	Manipulate data using Node.js and write the manipulated data back to another file.	
Graph Snippets [page 651]	Simple Python File Data Manipulation [page 657]	To be configured in the wizard.	Manipulate data using plain Python with only built-in modules and write the manipulated data back to another file.	
Graph Snippets [page 651]	Simple R File Data Manipulation [page 658]	To be configured in the wizard.	Manipulate data using plain R (using only built-in libraries) and write the manipulated data back to another file.	
Graph Snippets [page 651]	Load Files into HANA (Full Load) [page 653]	To be configured in the wizard.	Load data from CSV or JSON files stored in a (cloud) storage into a HANA table and terminate once all files are loaded.	
Graph Snippets [page 651]	Load Files into VORA (Incremental Load) [page 653]	To be configured in the wizard.	Ingest data from CSV or JSON files stored in a (cloud) storage into a VORA streaming table by first processing all existing files and then monitoring the directory for any new or modified files.	
Ingestion via Flow-agent	Ingest OData Query [page 662]	com.sap.dh.ingestion.odata.queryconsumer	Read data from OData sources using the OData Query Consumer to read the result set of the native OData Query URL.	

Category	Graph	Technical Name	Primary Use Case	Status
Ingestion via Flow-agent	Flowagent Consumer [page 661]	com.sap.dh.ingestion.oracle.sqlconsumer	See two methods provided by Data Services for reading data from Oracle sources using the third party OCI client.	
Ingestion via Flow-agent	CSV Producer [page 660]	com.sap.dh.ingestion.csvproducer	Use the CSV Producer operator mainly to detail the usability of the outMessage port.	
Ingestion via Flow-agent	CDC Graph Generator Sample Graph [page 659]	com.sap.dh.cdc.graph-Generator	See how to use CDC Graph Generator operator.	
Ingestion via Flow-agent	Open Connectors SQL Consumer [page 670]	com.sap.dh.ingestion.openconnectors.sqlconsumer	See how to use Open Connectors SQL Consumer, which is a Flow-agent operator and uses Flowagent subengine for execution.	
Ingestion via Flow-agent	Open Connectors Table Consumer [page 671]	com.sap.dh.ingestion.openconnectors.tableconsumer	Use Open Connectors Table Consumer, which is a Flowagent operator that uses Flow-agent sub engine for execution.	
Ingestion via Flow-agent	MySQL Consumer [page 664]	com.sap.dh.ingestion.sqlconsumerparallel	Use MySQL SQL Consumer to perform a SQL query and read it in parallel.	
Ingestion via Flow-agent	Table Consumer via Flowagent [page 672]	com.sap.dh.ingestion.tableconsumer	Use Oracle Table Consumer, which is a Flow-agent operator that uses Flowagent subengine for execution.	
Ingestion via Flow-agent	Ingest to Table via Flowagent [page 663]	com.sap.dh.ingestion.tableproducer	Use Flowagent Table Producer to write to a HANA Table.	
ML Scenario Manager Templates and Examples [page 677]	Auto ML Inference Template [page 678]	com.sap.dsp.templates.automl_inference	Check later for information on this graph.	
ML Scenario Manager Templates and Examples [page 677]	Auto ML Training Template [page 678]	graph com.sap.dsp.templates.automl_training	Check later for information on this graph.	

Category	Graph	Technical Name	Primary Use Case	Status
ML Scenario Manager Templates and Examples [page 677]		com.sap.ml.functional-Services	Get familiar with ML Functional Services. You can run the graph and use the terminal to interactively create request messages.	
ML Scenario Manager Templates and Examples [page 677]		com.sap.dsp.templates.functional_services_files_template_client	See an example client for the functional services Image Classification, Image Feature Extraction or Image OCR.	
ML Scenario Manager Templates and Examples [page 677]		com.sap.dsp.templates.functional_services_files_template_server	See an example template for the functional services Image Classification, Image Feature Extraction or Image OCR.	
ML Scenario Manager Templates and Examples [page 677]		com.sap.dsp.templates.functional_services_text_template_client	See an example template for the functional services Text Classifier and Similarity Scoring.	
ML Scenario Manager Templates and Examples [page 677]		com.sap.dsp.templates.functional_services_text_template_server	See a an example template for the Functional Services Similarity Scoring and Text Classifier.	
ML Scenario Manager Templates and Examples [page 677]	HANA ML Inference on API Requests [page 680]	com.sap.dsp.templates.hana_ml_inference_template	Expose a previously trained model as a REST endpoint.	
ML Scenario Manager Templates and Examples [page 677]	HANA ML Training Template [page 681]	com.sap.dsp.templates.hana_ml_training_template	Create a model using HANA PAL functionality and store the artifact in SAP DI Data Lake.	
ML Scenario Manager Templates and Examples [page 677]		com.sap.ml.functional_services_image_classification	Documentation for this graph is yet to be released.	
ML Scenario Manager Templates and Examples [page 677]	Python Consumer Template [page 687]	com.sap.dsp.templates.python_consumer_template	Consume a trained ML model and expose it via API endpoint. The Python36 - Inference operator loads a model binary from Artifact Producer and receives input data from the OpenAPI Servlow operator.	

Category	Graph	Technical Name	Primary Use Case	Status
ML Scenario Manager Templates and Examples [page 677]	Python Data Preprocessing Example [page 689]	com.sap.dsp.templates.python_data_preprocessing_example	Undertake data preprocessing with Python, register a dataset, and submit metrics in an ML scenario.	
ML Scenario Manager Templates and Examples [page 677]	Python Producer Template [page 691]	com.sap.dsp.templates.python_producer_template	Process data or train models using Python language in four steps.	
ML Scenario Manager Templates and Examples [page 677]	R Consumer - Consuming an ML Model via API using R Template [page 694]	com.sap.dsp.templates.r_consumer_template	Consume a trained ML model and expose it via API endpoint. The RClient - Inference operator loads a model binary from Artifact Producer and receives input data from the OpenAPI Servlow operator.	
ML Scenario Manager Templates and Examples [page 677]	R Consumer - Exposing a trained ML model via API Example [page 696]	com.sap.dsp.templates.r_consumer_example	Do inferences with a trained model from your ML scenario.	
ML Scenario Manager Templates and Examples [page 677]	R Producer - Training an ML Model Using R Template [page 697]	com.sap.dsp.templates.r_producer_template	Train an ML model using R language in four steps.	
ML Scenario Manager Templates and Examples [page 677]	R Producer - Training a classification model using R Example [page 698]	com.sap.dsp.templates.r_producer_classification_example	Consume data from a CSV file stored in the Local File System (vrep) to train a classification model.	
ML Scenario Manager Templates and Examples [page 677]	R Producer - Training a regression model using R [page 700]	com.sap.dsp.templates.r_producer_regression_example	Consume data from a CSV file stored in the Local File System (vrep) to create a linear regression model.	
ML Scenario Manager Templates and Examples [page 677]				
ML Scenario Manager Templates and Examples [page 677]	TensorFlow MNIST Training Example [page 701]	com.sap.dsp.templates.tensorflow_mnist_training_example	Documentation for this graph is yet to be released.	
ML Scenario Manager Templates and Examples [page 677]	Tensorflow Serving Template [page 702]	com.sap.dsp.templates.tensorflow_serving_template	Documentation for this graph is yet to be released.	
ML Scenario Manager Templates and Examples [page 677]	Basic Training (TensorFlow Training Template) [page 678]	com.sap.dsp.templates.tensorflow_training_template	Documentation for this graph is yet to be released.	

Category	Graph	Technical Name	Primary Use Case	Status
SAP Analytics Cloud	SAP Analytics Cloud Push Demo [page 704] (Push to SAP)	com.sap.demo.sac.push	Push data to a dataset in SAP Analytics Cloud.	
SAP Vora	Direct Avro Ingestion [page 709]	com.sap.demo.vora.ingestion.avro_ingestion_direct_example	Read a series of messages, each consisting of Avro records. Each message is passed to the Avro ingestor, which directly stores the records in the vora disk engine and emits the commit token associated with each message.	
SAP Vora	Avro Ingest via Disk [page 709]	com.sap.demo.vora.ingestion.avro_ingestion_example_disk	Read a series of messages, each consisting of Avro records. Each message is passed to the preingestor, which creates a message with record objects. Finally, this message is passed to the ingestor, which stores the records in the vora disk engine and emits the commit token associated with each message.	
SAP Vora		com.sap.demo.vora.ingestion.avro_ingestion_example_series	Read a series of messages, each consisting of Avro records. Each message is passed to the preingestor, which creates a message with record objects. Finally, this message is passed to the ingestor, which stores the records in the vora timeseries engine and emits the commit token associated with each message.	
SAP Vora	Direct CSV Ingest2 [page 713]	com.sap.demo.vora.ingestion.csv_ingestion_direct_example2	Generate a series of messages, each consisting of one or more CSV records.	

Category	Graph	Technical Name	Primary Use Case	Status
SAP Vora	CSV Ingest2 via Disk [page 711]	com.sap.demo.vora.ingestion.csv_ingestion_example2_disk	Generate a series of messages, each consisting of one or more CSV records.	
SAP Vora		com.sap.demo.vora.ingestion.csv_ingestion_example2_series	Generate a series of messages, each consisting of one or more CSV records.	
SAP Vora	CSV Ingest3 via Disk [page 712]	com.sap.demo.vora.ingestion.csv_ingestion_example3_disk	Use an Avro schema for ingestion, with additional extension attributes to customize the table definition.	
SAP Vora	JSON Ingest2 via Disk [page 714]	com.sap.demo.vora.ingestion.json_ingestion_example2_disk	Generate a series of messages, each consisting of one or more JSON records.	
SAP Vora	Record Ingest via Disk [page 715]	com.sap.demo.vora.ingestion.rec_ingestion_example2_disk	Generate a series of messages, each consisting of one or more record objects.	
SAP Vora	Avro Ingest Mod via Disk [page 710]	com.sap.demo.vora.ingestion.avro_ingestion_example_disk_go	Read a series of messages, each consisting of Avro records. Each message is passed to the preingestor, which creates a message with record objects. Finally, this message is passed to the ingestor, which stores the records in the vora disk engine and emits the commit token associated with each message.	
SAP Vora	CSV Ingest2 via Disk [page 712]	com.sap.demo.vora.ingestion.csv_ingestion_example2_disk_go	Read a series of messages, each consisting of one or more CSV records.	
Scenario Templates [page 722]	BW ODP to Vora (Initial + Delta Load) [page 723]	com.sap.scenarioTemplates.BW.bwData-TransferODP	Fetch data from a BW DataStore object and write it into a VORA target via ODP.	
Scenario Templates [page 722]	BW HANA View to File [page 724]	com.sap.scenarioTemplates.BW.bwData-TransferHANAView	Fetch data from a BW DataStore object and write it into a file (S3) target via HANA View.	

Category	Graph	Technical Name	Primary Use Case	Status
Scenario Templates [page 722]	Data Extraction from SAP S/4HANA CDS View to a File Store [page 725]	com.sap.scenarioTemplates.ABAP.CDStoFile	Extract data from an ABAP CDS View to a file store and create related files in the file store	
Scenario Templates [page 722]	Data Extraction from SAP S/4HANA CDS View to KAFKA [page 726]	com.sap.scenarioTemplates.ABAP.CDStoKafka	Extract data from an ABAP CDS View to the KAFKA and create related messages.	
Scenario Templates [page 722]	Data Extraction using SLT to a File Store [page 728]	com.sap.scenarioTemplates.ABAP.SLTtoFile	Extract data from an ABAP table using SLT to a file store and create related files in the file store.	
Scenario Templates [page 722]	Data Extraction using SLT to KAFKA [page 729]	com.sap.scenarioTemplates.ABAP.SLTtoKafka	Extract data from an ABAP table using SLT to the KAFKA and create related messages.	
Scenario Templates [page 722]	Initial Load + Delta Extraction from AnyDB [page 733]	com.sap.scenarioTemplates.ETL-FromDB.cdcGraph-Generator	Use the Table Replicator operator for replication of relational databases.	
Scenario Templates [page 722]	Ingest Files into HANA [page 730]	com.sap.scenarioTemplates.datalakeToDatabase.ingestToHana	Ingest product data from CSV files into a HANA table.	
Scenario Templates [page 722]	Initial Load from Oracle (parallel) [page 733]	com.sap.scenarioTemplates.ETL-FromDB.cdcInitialLoad	Read contents from an Oracle table and load it into a file.	
Scenario Templates [page 722]	Ingest Files into Vora [page 731]	com.sap.scenarioTemplates.datalakeToDatabase.ingestToVora	Ingest product data from CSV files into a Vora table.	
Scenario Templates [page 722]	Javascript File Data Manipulation [page 734]	com.sap.scenarioTemplates.customDataProcessing.F2FJavascript	Manipulate data using a Javascript custom operator and writing the manipulated data back to another file.	
Scenario Templates [page 722]	Load Files into HANA [page 735]	com.sap.scenarioTemplates.datalakeToDatabase.loadToHana	Load product data from CSV files into a HANA table.	
Scenario Templates [page 722]	Load Files into Vora [page 736]	com.sap.scenarioTemplates.datalakeToDatabase.loadToVora	Load product data from CSV files into a Vora table.	
Scenario Templates [page 722]	Python File Data Manipulation with Pandas [page 738]	com.sap.scenarioTemplates.customDataProcessing.F2FPython	Manipulate file data using Python with the Pandas library and writing the manipulated data back to another file.	

Category	Graph	Technical Name	Primary Use Case	Status
Scenario Templates [page 722]	Node.js File Data Manipulation [page 737]	com.sap.scenarioTemplates.customDataProcessing.F2FNode	Manipulate file data using Node.js and writing the manipulated data back to another file.	
Scenario Templates [page 722]	R File Data Manipulation [page 739]	com.sap.scenarioTemplates.customDataProcessing.F2FR	Manipulate file data with an R custom operator and writing the manipulated data back to another file.	
Scenario Templates [page 722]	Simple Javascript File Data Manipulation [page 740]	com.sap.scenarioTemplates.customDataProcessing.simpleF2FJavascript	Manipulate file data using plain Javascript and writing the manipulated data back to another file.	
Scenario Templates [page 722]	Simple Javascript File-to-DB Data Manipulation [page 741]	com.sap.scenarioTemplates.customDataProcessing.simpleF2DBJavascript	Manipulate file data using plain Javascript and writing the manipulated data to an SAP HANA database table.	
Scenario Templates [page 722]	Simple Javascript DB-to-File Data Manipulation [page 742]	com.sap.scenarioTemplates.customDataProcessing.simpleDB2FJavascript	Manipulate data from an SAP HANA database table using plain Javascript and writing the manipulated data to a file.	
Scenario Templates [page 722]	Simple Node.js File Data Manipulation [page 743]	com.sap.scenarioTemplates.customDataProcessing.simpleF2FNode	Manipulate data using Node.js (without any libraries) and writing the manipulated data back to another file.	
Scenario Templates [page 722]	Simple Python File Data Manipulation [page 744]	com.sap.scenarioTemplates.customDataProcessing.simpleF2FPython	Manipulate data using plain Python with only built-in modules and writing the manipulated data back to another file.	
Scenario Templates [page 722]	Simple Python File-to-DB Data Manipulation [page 745]	com.sap.scenarioTemplates.customDataProcessing.simpleF2DBPython	Manipulate data using plain Python with only built-in modules and writing the manipulated data to an SAP Data Hub built-in Vora database table.	

Category	Graph	Technical Name	Primary Use Case	Status
Scenario Templates [page 722]	Simple R File Data Manipulation [page 746]	com.sap.scenarioTemplates.customDataProcessing.simpleF2FR	Manipulate data using plain R (using only built-in libraries) and writing the manipulated data back to another file.	
Scenario Templates [page 722]	Simple R File-to-DB Data Manipulation [page 747]	com.sap.scenarioTemplates.customDataProcessing.simpleF2DBR	Manipulate data using plain R (using only built-in libraries) and writing the manipulated data to an SAP HANA database table.	
Streaming Analytics	Using Adapters in Streaming Analytics Project CCL [page 748]	com.sap.streaming.adapters	This example graph shows how you can send and receive data to and from a streaming analytics project using adapters.	
Streaming Analytics	Streaming Analytics Freezer Monitor Demo [page 749]	com.sap.streaming.demo	See a more complex graph involving streaming analytics operators.	
Streaming Analytics	Connecting to Kafka in Streaming Analytics Project CCL [page 750]	com.sap.streaming.kafka	See how a streaming analytics project can use adapters to consume and produce data from and to a Kafka topic.	
Streaming Analytics	Sending Datatypes to Streaming Analytics Projects [page 751]	com.sap.streaming.types	Send data rows containing various streaming analytics column datatypes to a streaming analytics project.	
Text Analysis		com.sap.textanalysis.example	Use text analysis to show the use of the Text Analysis Connector operator.	
Utilities		com.sap.demo.util.responseinterceptor	Use util.responseinterceptor to intercept a response message and deliver it to the initiator of a request response invocation.	

Category	Graph	Technical Name	Primary Use Case	Status
Utilities		com.sap.demo.util.re-spintercept2	Use util.responseinterceptor to inject a response callback to an incoming message so that when its callback is invoked, the corresponding response message is emitted from this operator and this message can be sent to the responseinterceptor operator.	
Utilities		com.sap.demo.util.re-spinvoke	Use util.responseinterceptor to invoke the response callback to deliver a response message to the initiator of a request response invocation.	
Utilities	Wiretap [page 753]	com.sap.demo.util.wiretap	Inspect messages that are transferred from one operator to another.	
Utilities	Wiretap 2 [page 754]	com.sap.demo.util.wiretap2	Inspect messages that are transferred from one operator to another.	
Utilities	Wiretap 3 [page 754]	com.sap.demo.util.wiretap3	Inspect messages that are transferred from one operator to another.	

Related Information

[Connectivity \[page 623\]](#)

[Examples \[page 628\]](#)

[Graph Snippets \[page 651\]](#)

[Ingestion via Flowagent \[page 659\]](#)

[ML Scenario Manager Templates and Examples \[page 677\]](#)

[Others \[page 703\]](#)

[SAP Integration \[page 704\]](#)

[SAP Vora \[page 708\]](#)

[Scenario Templates \[page 722\]](#)

[Streaming Analytics \[page 747\]](#)

[Utilities \[page 752\]](#)

3.1 Connectivity

Related Information

[Example for OpenAPI Greeter Demo \(Go\) \[page 623\]](#)
[Example for OpenAPI Plain Greeter Demo \(Go\) \[page 623\]](#)
[Kafka \[page 624\]](#)
[OpenAPI GreeterClient Demo \[page 624\]](#)
[OpenAPI PlainClient Demo \[page 625\]](#)
[OpenAPI Greeter Demo \[page 625\]](#)
[OpenAPI Plain Greeter \[page 626\]](#)
[S3 \[page 626\]](#)
[Google Pub/Sub \[page 627\]](#)

3.1.1 Example for OpenAPI Greeter Demo (Go)

This is a Golang version of demo graph *OpenAPI Greeter Demo*.

Related Information

[OpenAPI Greeter Demo \[page 625\]](#)

3.1.2 Example for OpenAPI Plain Greeter Demo (Go)

This is a Golang version of demo graph *OpenAPI Plain Greeter*.

Related Information

[OpenAPI Plain Greeter \[page 626\]](#)

3.1.3 Kafka

The data generator generates arbitrary data and passes it to the Kafka producer. The Kafka consumer reads the data and writes it to a terminal. You can see the generated data by opening the UI of the terminal operator.

Prerequisites

- A running Kafka broker and Zookeeper .

Configure and Run the Graph

1. In the left panel, select the [Graphs](#) tab and navigate to `com/sap/demo/kafka`.
2. Check the configuration of the [producer \(20fb\) node](#): brokers
3. Check the configuration of the [consumer \(1h7j\) node](#): zookeepers
4. In the tool bar, select [Run](#) (play button).
5. The [Status](#) panel indicates if the graph is running.
6. Use the context menu [Open UI](#) of the [terminal \(z8d\) node](#) to open the terminal.
7. The terminal opens and you see the generated data.

3.1.4 OpenAPI GreeterClient Demo

The `openapi.client` operator can call arbitrary REST services from a graph.

Configure and Run the Graph

Follow the steps below to run the example from the SAP Data Intelligence Modeler UI:

1. Start the `openapi.server.greeter` demo graph (`com.sap.demo.openapi.server.greeter`), which exposes its service at path `/openapi/service/samples/greeter` relative to the Modeler instance URL.
2. Verify the host and port configured for this demo graph matches the actual Modeler instance.
3. Start this demo graph, which will automatically start invoking a series of operations.
4. Open the UI from the wiretap operator so that the response messages from the invoked operations are displayed on the browser.

3.1.5 OpenAPI PlainClient Demo

The `openapi.client` operator can call arbitrary REST services from a graph.

Configure and Run the Graph

Follow the steps below to run the example from the SAP Data Intelligence Modeler UI:

1. Start the `openapi.server plain_greeter` demo graph (`com.sap.demo.openapi.server.plain_greeter`), which exposes its service at path `/openapi/service/samples/plain_greeter` relative to the Modeler instance URL.
2. Verify the host and port configured for this demo graph matches the actual Modeler instance.
3. Start this demo graph, which will automatically start invoking a series of operations.
4. Open the UI from the wiretap operator so that the response messages from the invoked operations are displayed on the browser.

3.1.6 OpenAPI Greeter Demo

The `openapi.server` operator allows REST services to be exposed from a graph. It can expose the endpoint as a Swagger-driven service endpoint or a plain REST service endpoint. This example demonstrates its Swagger-driven mode. The exposed service is described by a Swagger document and the request messages are handled based on this document.

Configure and Run the Graph

Follow the steps below to run the example from the SAP Data Intelligence Modeler UI:

1. Start the graph. This demo service will be started at path `/openapi/service/samples/greeter` relative to the Modeler instance URL.
2. Open the UI from the wiretap operator so that the request messages are displayed on the browser.
3. Open the URL `/openapi/service/samples/greeter/swagger.json` to retrieve its Swagger document. (If you are using Chrome Swagger-plugin, you can directly view this document and invoke the operations. Otherwise, use another tool to invoke these REST operations described in the document.)

For example, when using `curl`, these correspond to:

```
curl http://{host}:{port}/{path}/openapi/service/samples/greeter/v1/ping
curl -X -d 'some text' http://{host}:{port}/{path}/openapi/service/samples/greeter/v1/echo
```

The implementation of the operations can be found in the script operator.

3.1.7 OpenAPI Plain Greeter

The `openapi.server` operator allows REST services to be exposed from a graph. It can expose the endpoint as a Swagger-driven service endpoint or a plain REST service endpoint. This example demonstrates its plain service mode. There is no Swagger document associated with this service.

Configure and Run the Graph

Follow the steps below to run the example from the SAP Data Intelligence Modeler UI:

1. Start the graph. This demo service will be started at path `/openapi/service/samples/plain_greeter` relative to the Modeler instance URL.
2. Open the UI from the wiretap operator so that the request messages are displayed on the browser.
3. From some REST client, send a GET request to `/openapi/service/samples/plain_greeter/v1/ping` or a POST request to `/openapi/service/samples/plain_greeter/v1/echo` with some text content, as in the greeter sample. In addition, there is another operation hosted at `/openapi/service/samples/plain/_greeter/v1/happy`, which returns an HTTP response with either status 200 or 400 randomly. When using curl, these correspond to:

```
curl http://{host}:{port}{mountpath}/openapi/service/samples/plain_greeter/v1/ping
curl -X POST -d 'some text' http://{host}:{port}{mountpath}/openapi/service/samples/plain_greeter/v1/echo
curl -i http://{host}:{port}{mountpath}/openapi/service/samples/plain_greeter/v1/happy
```

For example, for a local Modeler instance:

```
$ curl http://localhost:8090/openapi/service/samples/plain_greeter/v1/ping
{"pong":5}
$ curl http://localhost:8090/openapi/service/samples/plain_greeter/v1/ping
{"pong":6}
$ curl -X POST -d 'Buenos dias' http://localhost:8090/openapi/service/samples/plain_greeter/v1/echo
{"echo":"Buenos dias"}
$ curl -i http://localhost:8090/openapi/service/samples/plain_greeter/v1/happy
HTTP/1.1 200 OK
...
{"status":"happy"}
$ curl -i http://localhost:8090/openapi/service/samples/plain_greeter/v1/happy
HTTP/1.1 400 Bad Request
...
{"status":"unhappy"}
```

The implementation of the operations can be found in the script operator.

3.1.8 S3

The data generator produces arbitrary data.

The data is written to files on S3 by Write File. The files are stored in the directory `/tmp/demo/` (of the specified bucket). The file names follow the scheme `file_<counter>.txt`.

The Read File reads these files and the content is written to a terminal.

Prerequisites

- You need your AWS credentials, the bucket, and the region to use.

Configure and Run the Graph

Follow the steps below to run the example from the SAP Data Intelligence Modeler UI:

1. In the left panel, select the [Graphs](#) tab and navigate to `com/sap/demo/s3`.
2. Check the connection configuration of the [Write File](#) node: `awsAccessKeyId`, `awsSecretAccessKey`, `bucket`, `region`.
3. Check the configuration of the [Read File](#) node: `awsAccessKeyId`, `awsSecretAccessKey`, `bucket`, `region`.
4. In the tool bar, select [Run](#) (play button).
5. The [Status](#) panel indicates if the graph is running.
6. Use the context menu [Open UI](#) of the [Terminal](#) node to open the terminal.
7. The terminal opens and you see the produced files and their content.

Optional Step: Check the files on S3

1. Connect to S3 and browse the directory `/tmp/demo/` (of the specified bucket) to see the produced files.

3.1.9 Google Pub/Sub

This example shows how to publish messages using the Google Pub/Sub Producer, and consume them via the Google Pub/Sub Consumer.

Prerequisites

- A project on GCP with access to the Pub/Sub service and the corresponding access key file as JSON.

Configure and Run the Graph

Follow the steps below to run the example from the SAP Data Intelligence Modeler UI:

1. In the left panel, select the [Graphs](#) tab and open the Google Pub/Sub graph.

2. Check the configuration of the [Google Pub/Sub Producer](#) operator.
3. Create a connection containing a valid GCP project ID and corresponding key file using the manual configuration type. Alternatively, you can reuse an existing connection by selecting configuration manager as configuration type. Enter the name of the topic that the producer publishes to.
4. Check the configuration of the "Google Pub/Sub consumer" node. Similar to the producer, consumer also requires a valid connection. Next, enter the subscription name and if this subscription does not exist on the Pub/Sub service, enter the same topic name as in the producer.
5. In the tool bar, select [Run](#) (play button).
6. The [Status](#) panel indicates if the graph is running.
7. Use the context menu [Open UI](#) of the [publish/output](#) node to open the terminal. Here, any message you type in, is sent as a publication to the configured topic in the Google Pub/Sub service.
8. The bottom half shows the output and error messages of the consumer and producer.
9. Using the provided message ID, you can acknowledge receiving the message to the Google Pub/Sub service.
10. Open the context menu [Open UI](#) of the [ack/output](#) node to open the terminal.
11. Enter a message with the attribute: `{ "msg.MessageID": "<MESSAGE_ID>" }` with `<MESSAGE_ID>` replaced with the message ID of the message to be acknowledged. If you do not explicitly acknowledge the message, after stopping the graph and restarting it, the message will be delivered again. In this case, the redelivered messages can be seen in the `wiretap` node. Alternatively, you can disable explicit acknowledgment by removing the input connection to the `messageToAck` port of the `Google Pub/Sub Consumer`. In this case, any message received by the consumer is acknowledged automatically.

3.2 Examples

Related Information

[Anonymization \[page 629\]](#)
[Basic Validation \[page 630\]](#)
[Data Generator \[page 631\]](#)
[Data Mask \[page 632\]](#)
[Data Workflow \[page 632\]](#)
[DQMm Address \[page 633\]](#)
[DQMm Reverse \[page 634\]](#)
[Example Counter \[page 634\]](#)
[Example Notification \[page 635\]](#)
[File System \[page 636\]](#)
[Functional Services \[page 636\]](#)
[GCS \[page 637\]](#)
[Go Data Generator \[page 638\]](#)
[Go Message Generator \[page 638\]](#)

[Go Operator and File \[page 639\]](#)
[Google Dataproc \[page 640\]](#)
[HDFS \[page 640\]](#)
[HTML Viewer \[page 641\]](#)
[Histogram Simple Example \[page 641\]](#)
[IoT Validation \[page 642\]](#)
[Kafka \[page 643\]](#)
[Load into SAP HANA \[page 644\]](#)
[Load into SAP Vora \[page 644\]](#)
[Message Generator \[page 645\]](#)
[Person and Firm \[page 646\]](#)
[RClient Example \[page 646\]](#)
[Simple Wiring \[page 649\]](#)
[Storage \[page 649\]](#)
[S3 \[page 650\]](#)
[Vora SQL Console \[page 651\]](#)

3.2.1 Anonymization

The anonymization operator helps protect the privacy of individuals by creating groups of similar records so you can discover statistically valid insights from your data without risking re-identification of individuals. The data is written to a terminal. While the graph is running, you can see the generated input data by opening the UI of the input terminal operator. Likewise, you can see the anonymized data by opening the UI of the output terminal operator while the graph is running.

Configure and Run the Graph

Follow the steps below to run the example from the SAP Data Intelligence UI:

1. Open the [Anonymization](#) example graph.
2. In the tool bar, select [Run](#) (play button). The [Status](#) panel indicates whether the graph is running.
3. To view the input data, right click the [Input Terminal](#) node and select [Open UI](#). The terminal opens and you see the generated data.
4. To view the output data, right click the [Output Terminal](#) node and select [Open UI](#). The terminal opens and you see the anonymized data.

3.2.2 Basic Validation

The Validation Rule operator evaluates input data against a rule. That is, a logical expression checking for particular values, null, not null, and so on. The user may define one or more rules and give each rule a meaningful name.

Input data that evaluates to true are sent to the 'pass' port, and failing records are similarly sent to the 'fail' port. There is a 'failInformation' port that shows the rules that failed and a unique ID for those records. The 'outError' port receives messages to help diagnose problems, such as corrupt input stream characters.

Configure and Run the Graph

Follow the steps below to run the example from the SAP Data Intelligence Modeler UI:

1. Click the [Validation Rule Demo](#) icon to load the sample graph. Notice the Validation Rule operator has four output ports (described above) attached to four Terminal operators.
2. In the toolbar, select [Run](#) (play button). The [Status](#) panel indicates whether the graph is running.
3. To view the passing records, click the [Pass Output](#) terminal, and then select [Open UI](#).
4. To view the failing records, click the [Fail Output](#) terminal, and then select [Open UI](#).
5. To view the failure information, click the [Fail Info Output](#) terminal, and then select [Open UI](#).
6. To view any error messages, click the [Error Output](#) terminal, and then select [Open UI](#). For this example graph, there will be no errors.

Rules and Substitution Values Explained

The Validation Rule operator in this example is configured not only to evaluate the rule, but also to update a column with a substitution value in the event a rule failure. This section explains how to use a rule and an associated substitution value in more detail.

1. The input data generated by the Data Generator has four columns: ID, Name, Title, and Salary. The Name, Title, and Salary values are randomly selected, and the Title field has the possibility of having a null value.
2. Click the [Validation Rule](#) operator and select [Open Configuration](#). The configuration values are shown in the right pane.
3. Click on the [Rules *](#) option. The rule "Title is NOT NULL" is shown.
 - By default, this means that records with the Title value of NOT NULL are sent to the Pass port, and records with a Title value of null, are sent to the Fail port. This behavior is controlled by the [Fail Action](#) setting, which has a FAIL default value.
4. But for this example, Fail Action is set to BOTH. This instructs the Validation Rule operator to send records that fail the rule to both the Pass port and ALSO the Fail port. Why would you do this?
 - You can provide a substitution value to update or correct the failed records.
 - The substitution value "fixes" the aspects of a record that failed the rule and all the records are sent to one port. The records are output in the same order from the input source, even for the records that have the substitution value included.
5. Click the [Substitutions](#) option. The substitution "Title - 'Intern'" is shown.

- Because the substitution is defined as Intern, any records that fail AND are sent to the Pass port have the Title column updated with the value of Intern. Those records that go the Fail port do not have the Title column updated, and remain null.

Input data (ID, Name, Title, Salary), 3 records

- Notice that record 2 has a null Title value.

```
1,Ben,Software Engineer,60543.45
2,Drew,,0.0
3,Glenn,CEO,123054.34
```

Pass port:

- Receives all input records (passing and failing records).
- Notice the failed record (#2) received the substitution value of Intern for the Title column.
- Substitution action only occurs to records that fail a rule AND are sent to the Pass port.

```
1,Ben,Software Engineer,60543.45
2,Drew,Intern,0.0
3,Glenn,CEO,123054.34
```

'fail' port:

- Receives the one record that failed the rule.
- Notice the B indicating record was sent to BOTH the Pass and Fail ports.
- Failed rules are shown with a unique identifier.

```
2,Drew,,0.0,B,Failed rules(s): Title,1.0
```

'failInformation' port:

- Receives information on failed records: unique ID, rule name, and column name.

```
1.0,Rule_1,Title
```

3.2.3 Data Generator

The data generator generates arbitrary values simulating a set of sensors. The data is written to a terminal. You can see the generated data by opening the UI of the terminal operator.

Configure and Run the Graph

Follow the steps below to run the example from the SAP Data Intelligence Modeler UI:

- In the left panel, select the *Graphs* tab and navigate to `com/sap/demo/datagenerator`.

2. In the tool bar, select [Run](#) (play button).
3. The [Status](#) panel indicates if the graph is running.
4. Use the context menu [Open UI](#) of the [terminal \(1mnu\)](#) node to open the terminal.
5. The terminal opens and you see the generated data.

Optional Step: See the Code for Data Generation

1. If you'd like to see the code used for the data generation, use the context menu [Open Script](#) of the [datagenerator \(16d1\)](#) node.

3.2.4 Data Mask

The Data Mask operator protects the personally identifiable or sensitive information by covering all or a portion of the data. The data is written to a terminal. While the graph is running, you can see the generated input data by opening the UI of the input terminal operator. Likewise, you can see the masked data by opening the UI of the output terminal operator while the graph is running.

Configure and Run the Graph

1. In the left panel, select the [Graphs](#) tab and navigate to [Examples > Data Mask](#).
2. In the tool bar, select [Run](#) (play button).
3. The [Status](#) panel indicates whether the graph is running.
4. To view the input data, right click the [Input Terminal](#) node and select [Open UI](#). The terminal opens and you see the generated data.
5. To view the output data, right click the [Output Terminal](#) node and select [Open UI](#). The terminal opens and you see the masked data.

3.2.5 Data Workflow

This graph starts an internal example graph (com.sap.demo.counter). It serves as an example on how to use the operators from the Data Workflow category.

Configure and Run the Graph

Follow the steps below to run the graph from SAP Data Intelligence Modeler:

1. In the left panel, select the [Graphs](#) tab and navigate to [Examples > Data Workflow](#).
2. In the tool bar, select [Run](#) (play button).
3. The [Status](#) panel indicates if the graph is running.
4. The example graph will appear in the [Status](#) panel and the status is "running".

Configuration (Optional)

By default, the example graph is configured to terminate itself. In this context, the Data Workflow waits for the example graph to finish and will turn into "completed" after the example graph finishes. If you want to run another graph without waiting for the graph to finish, follow the steps below:

1. Open the Data Workflow, select [Pipeline](#) operator and open the configuration (the 3rd button).
2. Go to [Graph Name](#) and select a graph from the list.
3. Go to [Running Permanently](#) and choose [true](#) from the list.

If you want to run a graph on a remote system, follow the documentation of [Pipeline \[page 287\]](#).

3.2.6 DQMm Address

The DQMm Address Cleanse operator creates a request from data that can be passed to the DQMm Client operator to be sent to the service.

Prerequisites

- Subscription to the Data Quality Management, microservices for location data service.

Configure and Run the Graph

Follow the steps below to run the example from the SAP Data Intelligence Modeler UI:

1. Start the DQMm Address Cleanse demo graph (com.sap.demo.dqmm.addressCleanse).
2. Set the host of the DQMm application URL in the DQMm Client operator.
3. Choose either to authenticate using basic or oauth. You can use basic if you are accessing your subscription from your account on Trial. Use oauth if you are accessing your subscription from your account on a Factory landscape.
4. If you are using ouath, log into your SAP Cloud Platform account and create an OAuth client for your subscription.
5. In the DQMm Client operator, set the OAuth Token URL, Client ID, and the Client Secret.
6. By default, the graph writes the output to the `/tmp` folder. Change this if you want the result written to another folder.
7. Start the demo graph, which will automatically start invoking a series of operations.

3.2.7 DQMm Reverse

Start this demo graph, which will automatically start invoking a series of operations.

Prerequisites

- Subscription to the Data Quality Management, microservices for location data service.

Configure and Run the Graph

Follow the steps below to run the example from the SAP Data Intelligence Modeler UI:

1. Start the DQMm Reverse Geo demo graph (com.sap.demo.dqmm.reverseGeo).
2. Set the host of the DQMm application URL in the DQMm Client operator.
3. Choose either to authenticate using basic or oauth. You can use basic if you are accessing your subscription from your account on Trial. Use oauth if you are accessing your subscription from your account on a Factory landscape.
4. If you are using oauth, log into your SAP Cloud Platform account and create an OAuth client for your subscription.
5. In the DQMm Client operator, set the OAuth Token URL, Client ID, and the Client Secret.
6. By default, the graph writes the output to the `/tmp` folder. Change this if you want the result written to another folder.
7. Start the demo graph, which will automatically start invoking a series of operations.

3.2.8 Example Counter

The Example Counter graph runs for five seconds and then shuts down with status "completed." It is intended for testing purposes.

Configure and Run the Graph

Follow the steps below to run the graph from SAP Data Intelligence Modeler:

1. In the left panel, select the [Graphs](#) tab and navigate to [Examples > Example Counter](#).
2. In the tool bar, select [Run](#) (play button).
3. The [Status](#) panel indicates if the graph is running.
4. The example graph will appear in the [Status](#) panel and the status is "running".
5. After five seconds, the status will become "completed" unless an error occurred.

3.2.9 Example Notification

This graph serves as an example on how to use SAP Data Intelligence and Pipeline operators from the Data Workflows category.

In the graph, the Pipeline operator starts an internal example graph (`com.sap.demo.counter`) as a subgraph. If the example graph terminates with the status “dead”, an email notification will be sent to the configured email addresses.

Configure and Run the Graph

Follow the steps below to run the graph from SAP Data Intelligence Modeler:

1. In the left panel, select the [Graphs](#) tab and open the [Example Notification](#) graph.
2. In the tool bar, select [Run](#) (play button).
3. The [Status](#) panel indicates if the graph is running.
4. The example graph will appear in the [Status](#) panel and the status is "running".

To show the status of the example graph, toggle on [Show Subgraphs](#).

Configuring the Pipeline Operator (Optional)

By default, the example graph is configured to terminate itself. In this context, the Data Workflow waits for the example graph to finish and will turn into “completed” after the example graph finishes. If you want to run another graph without waiting for the graph to finish, follow the steps below:

1. Select [Pipeline](#) operator and open the configuration.
2. Go to [Graph Name](#) and select a graph from the list.
3. Go to [Running Permanently](#) and choose [true](#) from the list.

If you want to run a graph on a remote system, follow the documentation of [Pipeline \[page 287\]](#).

Configuring the Notification Operator

1. Select [Notification](#) operator and open the configuration.
2. Go to [Connection](#), choose either [Configuration Manager](#) or [Manual](#). When using [Configuration Manager](#), create a connection via [Connection Management](#) first.

i Note

It is recommended to choose [Configuration Manager](#) as the configuration type and select an existing [Connection ID](#). The [Manual](#) configuration type must be used only for testing purposes.

3. Go to [Default From Value](#) and input an email address as sender.

4. Go to [Default To Value](#) and input one or more receiver email addresses.
5. Go to [Default Subject Value](#) to give a subject to the email.

3.2.10 File System

The Data Generator produces arbitrary data.

The data is written to files on File System by Write File. The files are stored in the directory `./tmp/demo/` (where `./` is the current working directory, for example, where the Data Pipeline Engine was started). The file names follow the scheme `file_<counter>.txt`.

The Read File reads this file and the content is written to a terminal.

Configure and Run the Graph

Follow the steps below to run the example from the SAP Data Intelligence Modeler UI:

1. In the left panel, select the [Graphs](#) tab and navigate to `com/sap/demo/file`.
2. In the tool bar, select [Run](#) (play button).
3. The [Status](#) panel indicates if the graph is running.
4. Use the context menu [Open UI](#) of the [Terminal](#) node to open the terminal.
5. The terminal opens and you see the produced files and their content.

3.2.11 Functional Services

This is an example graph for using ML functional services. To get familiar with the services, you can run the graph and use the terminal to interactively create request messages.

The graph is composed of the following main components:

- Functional Services: This operator interacts with its assigned service (configured in the operator). It simplifies the input, which should be just the desired operation and the content to be sent.
- Terminal: The Terminal can be used in this graph to interactively create requests to the functional services.

Configuration parameters

Selects the functional service and selects whether to use the default connection or a custom one

- If manual is selected
Fill with the credentials and services URLs:
 - Authentication - should be OAuth2
 - Host - should be the service URL

- OAuth 2 Grant Type - should be client_credentials
 - OAuth 2 Token Endpoint, OAuth 2 Client ID and OAuth2 Client Secret
- Input message for the selected functional service.

Configure and Run the Graph

Follow the steps below to run the training example from the Data Pipeline UI:

1. Select the "Functional Services" operator.
2. In the configuration panel, select the Functional Service and a list of possible configurations will appear.
3. Fill the new configurations if desired (version, modelName, type or options). This depends on the selected service. For the options, in the message attributes, the JSON passed should be a string. Therefore, the quotes have to be escaped. Some input examples:

- Topic Detection

```
{"options": "{\"numTopics\":2, \"numTopicsPerDoc\":1, \"numKeywordsPerTopic\":15}\"}/vrep/test_images/test.zip
```

- Image OCR

```
{}/vrep/test_images/test.jpg
```

4. Run the graph, right-click on the [Terminal](#) operator and select [Open UI](#).
5. When in the terminal, press `Enter` to send the current line to the client.

3.2.12 GCS

The Data Generator produces arbitrary data.

The data is written to files on GCS by Write File. The files are stored in the directory `./tmp/demo/` (where `./` is the current working directory, for example, where the Data Pipeline Engine was started). The file names follow the scheme `file_<counter>.txt`.

The Read File reads this file and the content is written to a terminal.

Prerequisites

- You need your GCS credentials (the access key), the project ID, and the bucket.

Configure and Run the Graph

Follow the steps below to run the example from the SAP Data Intelligence Modeler UI:

1. In the left panel, select the [Graphs](#) tab and navigate to `com/sap/demo/s3`.
2. Check the connection configuration of the [Write File](#) node: `gcsJsonKey`, `gcsProjectID`, `gcsBucket`.
3. Check the configuration of the [Read File](#) node: `gcsJsonKey`, `gcsProjectID`, `gcsBucket`.
4. In the tool bar, select [Run](#) (play button).
5. The Status panel indicates if the graph is running.
6. Use the context menu [Open UI](#) of the [Terminal](#) node to open the terminal.
7. The terminal opens and you see the produced files and their content.

3.2.13 Go Data Generator

The data generator generates arbitrary values simulating a set of sensors. The data is written to a terminal. You can see the generated data by opening the UI of the terminal operator.

Configure and Run the Graph

Follow the steps below to run the example from the SAP Data Intelligence Modeler UI:

1. In the left panel, select the [Graphs](#) tab and navigate to `com/sap/demo/goDataGenerator`.
2. In the tool bar, select [Run](#) (play button).
3. The [Status](#) panel indicates if the graph is running.
4. Use the context menu [Open UI](#) of the [terminal \(1mnu\)](#) node to open the terminal.
5. The terminal opens and you see the generated data.

Optional Step: See the Code for Data Generation

1. If you would like to see the code used for the data generation, use the context menu [Open Script](#) of the [Go Data Generator](#) node.

3.2.14 Go Message Generator

The message generator generates arbitrary values packed into a message, simulating a set of sensors. The data is written to a terminal. You can see the generated data by opening the UI of the terminal operator.

Configure and Run the Graph

Follow the steps below to run the example from the SAP Data Intelligence Modeler UI:

1. In the left panel, select the [Graphs](#) tab and navigate to `com/sap/demo/goMessageGenerator`.
2. In the tool bar, select [Run](#) (play button).
3. The [Status](#) panel indicates if the graph is running.

4. Use the context menu [Open UI](#) of the [terminal \(1mnu\)](#) node to open the terminal.
5. The terminal opens and you see the generated data.

Optional Step: See the Code for Message Generation

1. If you would like to see the code used for the data generation, use the context menu [Open Script](#) of the [Go Message Generator](#) node.

3.2.15 Go Operator and File

This graph reads content from a file and uses the go operator to deal with the file content and name using different go functions and message type in the Go operator.

Prerequisites

- Any valid file configured on readFile operator.

Configure and Run the Graph

Follow the steps below to run the example from the SAP Data Intelligence Modeler UI:

1. In the left panel, select the [Graphs](#) tab and navigate to `com/sap/demo/goDataGenerator`.
2. In the tool bar, select [Run](#) (play button).
3. The [Status](#) panel indicates if the graph is running.
4. Use the context menu [Open UI](#) of [terminal \(1mnu\)](#) node to open the both terminal operators.
5. The terminal opens and you see the resulted data.

Optional Step: See the Code for Go Operator

If you would like to see the code used for the operations, use the context menu [Open Script](#) of the [Go Data Generator](#) operator.

3.2.16 Google Dataproc

This example shows how to submit a Spark job to an existing Google Dataproc cluster, and see the results of the job. In this example, we run the SparkPi example to calculate pi using Spark.

Prerequisites

- A project on Google Cloud with access to the Dataproc service, the corresponding access key file as JSON, a running Dataproc cluster, and a Hadoop job.

The Spark examples jar can be found in the download of Spark found here: <https://spark.apache.org/downloads.html> 📄

Configure and Run the Graph

Follow the steps below to run the example from the SAP Data Intelligence Modeler UI:

1. In the left panel, select the *Graphs* tab and open the Google Dataproc example graph.
2. Open the configuration of the *Hadoop Job Submitter Operator*.
3. Configure the connection by opening the connection dialogue box and choosing the correct connection from the Connection Manager app. Specify "spark" as the type of job.
4. Below job type, enter the main jar file URI of the form "gs://[your Google Storage Bucket]/[spark examples jar name].jar". Below the URI, ensure the class name as "org.apache.spark.examples.SparkPi"
5. In the tool bar, select *Run* (play button).
6. The *Status* panel indicates if the graph is running.
7. Use the context menu *Open UI* of the *Output* node to open the terminal. The bottom half shows the output and success messages of the Spark job.
8. If the graph failed, you can find the logs for in the Google Storage bucket specified from the failing message.

3.2.17 HDFS

The Data Generator produces arbitrary data.

The data is written to files on HDFS by Write File. The files are stored in the directory /tmp/demo/ (of the specified bucket). The file names follow the scheme file_<counter>.txt.

The Read File reads this file and the content is written to a terminal.

Configure and Run the Graph

Follow the steps below to run the example from the SAP Data Intelligence Modeler UI:

1. In the left panel, select the [Graphs](#) tab and navigate to `com/sap/demo/hdfs`.
2. Check the configuration of the [Write File](#) node: `hadoopNamenode`, `hadoopUser`
3. Check the configuration of the [Read File](#) node: `hadoopNamenode`, `hadoopUser`
4. In the tool bar, select [Run](#) (play button).
5. The [Status](#) panel indicates if the graph is running.
6. Use the context menu [Open UI](#) of the [Terminal](#) node to open the terminal.
7. The terminal opens and you see the produced files and their content.

Optional Step: Check the files on HDFS

1. Connect to HDFS and browse the directory `/tmp/demo/` to see the produced files.

3.2.18 HTML Viewer

This example graph simulates a Sales Campaign generating pseudorandom data for a large number of stores. The data is generated and structured at the [Python3Operator](#) and then passed as a String containing an HTML/CSS document that is displayed on [HTML Viewer](#) and constantly refreshed for the duration of the simulation.

Configure and Run the Graph

Follow the steps below to run the example from the SAP Data Intelligence Modeler UI:

1. Run the graph at the play-shaped button (remember to save it first in case you've made any changes).
2. The [Status](#) panel indicates whether the graph is running.
3. Right-click the [HTML Viewer](#) operator box and select [Open UI](#) to view the Sales Campaign simulation run over time.
4. (Optional) If you'd like to edit the simulation script, right click the [Python3Operator](#) box and then click [Open Script](#).

3.2.19 Histogram Simple Example

In this simple graph, the first operator generates random numbers that are sent to the [Histogram](#) operator. The [Histogram](#) operator produces the histogram of its inputs as a [Message](#) with an integer array on its body and the range of the interval on its attributes. The message is then converted to string by the [ToString Converter](#) and sent to the [HistogramPlotter](#), which finally plots it in the user interface.

This is a linear graph where the first operator is a [Javascript](#) operator that generates a random float number between 0 and 20 every 1 second. This float is sent to the [Histogram](#) operator that will produce an array of `uint64`, representing the histogram, each time it receives an input.

In this graph, the Histogram operator is configured with `nBins=5`, `rangeMin=2`, and `rangeMax=19`. This means that the outputted histogram array will have size 5 and the interval (2, 19) will be evenly distributed over the array's cells. That is, each cell will cover a subinterval of length $(19-2)/5=3.4$.

The bin of index 0 counts the inputs between 2 and 5.4, index 1 between 5.4 and 8.8, index 2 between 8.8 and 12.2, index 3 between 12.2 and 15.6, and index 4 between 15.6 and 19. Inputs received outside the range 2 and 19 will not be counted in any bin.

The output of the Histogram operator is a Message with the histogram array on its body and the `rangeMin` and `rangeMax` values on its attributes. This is sent to the ToString Converter, which will convert the Message to its string representation. This string representation is then fed into the Histogram Plotter operator, which shows a visual representation of the histogram that can be seen by clicking on the [Open UI](#) button on this operator.

3.2.20 IoT Validation

This graph models a sensor that tracks the temperature and surface illuminance of ice cream.

`Metadata Generator` generates sample metadata for a single sensor and passes it to `Metadata Producer`. `Metadata Producer` pushes the generated metadata to the Kafka cluster.

`Measurements Generator` generates sample sensor measurements and passes it to `Measurements Producer`. `Measurements Producer` pushes the generated sensor measurements to the Kafka cluster.

`Metadata Consumer` consumes the sample metadata, and `Measurements Consumer` consumes the generated sensor measurements.

`Stream Joiner` is a Javascript operator and joins sample sensor readings with its respective metadata. The joined output can be viewed with `Joined Output`.

`toCSVString` is a Javascript operator and converts the joined output to a CSV string.

`Validation Rule` tests the ice cream's temperature value is between -34.8°C and -24.8°C and the illuminance value is between 830 and 970 lux.

Sensor measurements that have passed the `Validation Rule` tests can be viewed with `Pass Output`.

Sensor measurements that have failed the `Validation Rule` tests can be viewed with `Fail Output`.

Detailed rule failure information can be viewed with `Fail Info Output`.

Any errors that may have occurred can be viewed with `Error Output`.

Prerequisites

- You need a running Kafka broker (version 0.9.0 or greater) and Zookeeper.

Configure and Run the Graph

To run the example from the SAP Data Intelligence UI:

1. Check the configuration of the `Metadata Generator` node: brokers and topic.
2. Check the configuration of the `Measurements Generator` node: brokers and topic.
3. Check the configuration of the `Metadata Consumer` node: brokers, topic (must be the same as `Metadata Consumer`), and `kafkaVersion` ($\geq 0.9.0$).
4. Check the configuration of the `Measurements Consumer` node: brokers, topic (must be the same as `Measurements Generator`), and `kafkaVersion` ($\geq 0.9.0$).
5. In the toolbar, select **Run** (play button). The **Status** panel indicates whether the graph is running.

Note

To view output of any terminal, activate the context menu of the desired terminal and select "Open UI".

3.2.21 Kafka

The data generator generates arbitrary data and passes it to the Kafka producer. The Kafka consumer reads the data and writes it to a terminal. You can see the generated data by opening the UI of the terminal operator.

Prerequisites

- A running Kafka broker and Zookeeper.

Configure and Run the Graph

1. In the left panel, select the **Graphs** tab and navigate to `com/sap/demo/kafka`.
2. Check the configuration of the **producer (20fb)** node: brokers
3. Check the configuration of the **consumer (1h7j)** node: zookeepers
4. In the tool bar, select **Run** (play button).
5. The **Status** panel indicates if the graph is running.
6. Use the context menu **Open UI** of the **terminal (z8d)** node to open the terminal.
7. The terminal opens and you see the generated data.

3.2.22 Load into SAP HANA

The data generator produces sensor data as a CSV string. The data is loaded into SAP HANA and in parallel sent to a terminal, which you can use to see the generated data.

Prerequisites

- A running SAP HANA database.

Configure and Run the Graph

1. Right-click the SAP HANA Client operator and select [Open Configuration](#). In the configuration panel, click the [Connection](#) property to show its editor, then fill in the values with valid connection parameters.
2. Save the graph.
3. In the tool bar, click the down-pointing arrow next to the [Run](#) button and choose [Run As](#).
4. In the dialog box, fill in the schema configuration value with a schema name that does not yet exist. The HANA Client operator will create it for you.
5. Click [OK](#).
6. The [Status](#) panel indicates if the graph is running.
7. Use the context menu [Open UI](#) of the [Terminal](#) node to open the terminal.
8. The terminal opens and you see the produced files and their content.

Optional Step: Check the files on HANA

1. Connect to SAP HANA and browse the table "test" under the schema you configured to see the produced sensor data.

3.2.23 Load into SAP Vora

The data generator produces sensor data, which is aggregated by the JavaScript operator. Every 30-record group is stored in a separate file on HDFS. The files are loaded into SAP Vora. The produced sensor data is also sent to a terminal, which you can use to see the generated data.

Prerequisites

- A running SAP Vora server.
- A running HDFS instance.

Configure and Run the Graph

1. Right-click the Write File operator and select [Open Configuration](#). In the configuration panel, click the [Connection](#) property to show its editor, then fill in the values with valid connection parameters.
2. Select the SAP Vora Loader operator and use the configuration panel to set the parameters you need:
 - Point [Vora Connection](#) to your Vora server.
 - Point [HDFS Connection](#) to the HDFS instance from step 1.
 - Use [Init statements](#) if you need the operator to create the target table for you when the graph starts running.
 - Set [Table name](#) to the table where you want the data to be stored.
3. Save and run the graph. You can right-click the [Terminal](#) operator and open its UI to inspect the data being generated.

Optional Step: Check the files on HDFS and the data in SAP Vora

1. Connect to HDFS and look at the files.
2. Open the SAP Vora Tools and browse the loaded data.

3.2.24 Message Generator

The message generator generates arbitrary values packed into a message, simulating a set of sensors. The data is written to a terminal. You can see the generated data by opening the UI of the terminal operator.

Configure and Run the Graph

Follow the steps below to run the example from the SAP Data Intelligence Modeler UI:

1. In the left panel, select the [Graphs](#) tab and navigate to `com/sap/demo/messageGenerator`.
2. In the tool bar, select [Run](#) (play button).
3. The [Status](#) panel indicates if the graph is running.
4. Use the context menu [Open UI](#) of the [terminal \(1mnu\)](#) node to open the terminal.
5. The terminal opens and you see the generated data.

Optional Step: See the Code for Data Generation

1. If you would like to see the code used for the data generation, use the context menu [Open Script](#) of the [Message Generator](#) node.

3.2.25 Person and Firm

Parses and standardizes person and firm (business organization) data. The Person and Firm example graph shows how the Person and Firm Cleanse operator is configured with different types of input: firm, name in a single column, name in multiple columns, person or firm, and country or language or region input data.

Prerequisites

None

Configure and Run the Graph

Follow the steps below to run the example from the Modeler:

1. Open the Person And Firm Cleanse example graph.
2. In the tool bar, choose [Run](#) (play button). The [Status](#) panel indicates whether the graph is running.
3. To view the output data, right-click the Output Terminal operator and select [Open UI](#). The terminal opens and you see the cleansed data.

3.2.26 RClient Example

Shows the different features of the RClient operator.

This graph has an RClient operator at its center, which has 6 input ports and 3 output ports. The script of this RClient operator shows how to use the following `api` methods: `api$setSwitchCallback`, `api$setPortCallback`, `api$addTimer`, and `api$addShutdownHandler`.

Get started by reading the [R Client 3 \[page 364\]](#) information page.

`api$setSwitchCallback`

First the following "switch" callback is set for the input port `input5`, and output port `output2`:

```
switchFunc <- function(data) {  
  pos <- FALSE  
  if (data > 10) {  
    pos <- TRUE  
  }  
  list(switchPosition=pos, output2=toString(data))  
}  
api$setSwitchCallback(c("input5"), c("output2"), "switchFunc")
```

The argument `data2` from `input5` is of type `int64`. Every time inport `input5` has data waiting to be read the `switchFunc` is called with this data as argument. The switch position remains in its "off" (`FALSE`) state until

the incoming data is greater than 10. When data is greater than 10, `pos` is set to `TRUE` and the switch position is moved to its "on" state.

Notice that the `switchPosition` parameter name in the returned `list` should always be used in a switch callback function.

As explained in the `RClient` documentation, the switch "on" position enables all other callbacks associated with other ports. This means that the other callbacks will start reading its input ports only when the inport `input5` receives a value greater than 10.

api\$setPortCallback

There are five port callbacks set in the script of the `RClient` operator. The first two callbacks, `foo` and `bar`, are associated with the same input ports `input1` and `input2`:

```
foo <- function(data1, data2) {
  list(output1=data1 + data2)
}
api$setPortCallback(c("input1", "input2"), c("output1"), "foo")
bar <- function(data1, data2) {
  flag <- (data1*data2) %% (data1 + data2) == 0
  if (!is.na(flag) && flag) {
    print("something") # Will only be printed in the Rserve stdout.
    return(NULL) # skip send to outports in this call
  }
  print("another thing")
  list(output1=data1 * data2, output2=toString(data1 + data2))
}
api$setPortCallback(c("input1", "input2"), c("output1", "output2"), "bar")
```

However, the `foo` sends to output `output1` and `bar` sends to outports `output1` and `output2`. As `bar` is registered after `foo`, when data is available in port `input1` and `input2` the same pair of data will first be used to call `foo` and then used to call `bar`.

The function `bar` shows two interesting features:

- You can use `print("text")` to send data to Rserve stdout.
- If you return `(NULL)` then nothing will be sent to the output ports associated with this callback for this given call. This is useful if for some calls of the callback you don't want to send anything to the output ports associated with it.

Notice that we send a number to the `output1` port as the port type is `int64`, and we send a string to `output2` as the port type is `string`.

The callback `foo2` is on the script just to demonstrate that callbacks associated with different inport groups can send to the same output port. In this case, `bar` and `foo2` both send to `output2`.

```
foo2 <- function(data1) {
  list(output2=paste("concat!", data1))
}
api$setPortCallback(c("input3"), c("output2"), "foo2")
```

The `dataFrameFunc` callback shows that when a message of type `table` is fed to input `input4` then it will be automatically converted to a `dataframe` object. The value of cell (1,2) of the `dataframe` is multiplied by 10 and then sent to `output3`. The class name of the received data is sent on `output2`.

```
dataFrameFunc <- function(data) {  
  data[1,2] <- data[1,2] * 10  
  list(output2=class(data), output3=data)  
}  
api$setPortCallback(c("input4"), c("output2", "output3"), "dataFrameFunc")
```

The `table` message is built using a `python2Operator` that has its output connected to `RClient`'s input `input4`. Below is the Python code that builds the `table` message:

```
msg = api.Message([[u"abc", u"defg"], [counter, 10]], {u"fieldNames": [u"column  
1", u"column 2"]}, u"table")  
api.send("output", msg)  
counter += 1
```

The `foo3` functions shows that a callback doesn't need to be associated with an output port. `foo3` in this case is simply used to update the global variable `counter` (which is going to be used again by another callback):

```
counter <- 0L  
foo3 <- function(data1) {  
  counter <- counter + nchar(data1)  
}  
api$setPortCallback(c("input3"), c(), "foo3")
```

api\$addTimer

The script in the `RClient` operator has two timer callbacks:

```
periodicFunc1 <- function() {  
  counter <- counter + 1L  
  list(output1=counter)  
}  
api$addTimer("1s", c("output1"), "periodicFunc1")  
periodicFunc2 <- function() {  
  counter <- counter * 2L  
}  
api$addTimer("2s", c(), "periodicFunc2")
```

The first increments the `counter` global variable and sends it to `output1` port, and the second is not associated with any port, and just multiply the `counter` by 2.

api\$addShutdownHandler

There are two shutdown callbacks in the `RClient` script. Those two will be called in the order they appear below when the graph is scheduled to stop or goes to dead.

```
There are two shutdown callbacks in the RClient script. Those two will be called  
in the order they appear below when the graph is scheduled to stop or goes to  
dead.
```


3.2.27 Simple Wiring

Some operators are capable of connecting multiple connections from and to a single port as well as connecting between different port types that are supported by the automatic conversion of the pipeline engine. This graph directly connects three data generating operators of different output port types to a wiretap that is directly connected to a terminal and another wiretap.

Prerequisites

Optional: basic understanding of REST services and OpenAPI 2.0

Configure and Run the Graph

Follow the steps below to run the example from the Data Pipeline UI:

1. Start the graph. This demo service will be started at path `/openapi/service/test/smart-wiring` relative to the pipeline instance URL.
2. Open the UI from Wiretap 1 and Terminal to see data from Data Generator and Message Generator are shown.
3. Optional: open the UI from OpenAPI Message to see its Swagger Specification and invoke the service.

3.2.28 Storage

The data generator produces arbitrary data.

The data is written to files by Write File. The files are stored in the directory `./tmp/demo/` (where `./` is the current working directory, for example, where the Data Pipeline Engine was started). The file names follow the scheme `file_<counter>.txt`.

The Move Message generates a message with the received filepath as the source, moving the file to the archived or directory, in the same path (destination is `${dirname}/archived/${basename}`).

The file content is read from the archived directory by Read File. After reading, the content is written to a terminal and the file is removed.

Configure and Run the Graph

Follow the steps below to run the example from the SAP Data Intelligence Modeler UI:

1. In the left panel, select the [Graphs](#) tab and navigate to `com/sap/demo/storage`.
2. In the tool bar, select [Run](#) (play button).

3. The [Status](#) panel indicates if the graph is running.
4. Use the context menu [Open UI](#) of the [Terminal](#) node to open the terminal.
5. The terminal opens and you see the produced files and their content.

Optional Step: Check the files on S3

1. Browse the directory `./tmp/demo/` and `./tmp/demo/archived/`, and check if the files are being removed.
2. Disconnect the input from the `Remove File` node and run the graph again.
3. Browse the directory `./tmp/demo/archived/` and check if the moved files are still there.

3.2.29 S3

The data generator produces arbitrary data.

The data is written to files on S3 by `Write File`. The files are stored in the directory `/tmp/demo/` (of the specified bucket). The file names follow the scheme `file_<counter>.txt`.

The `Read File` reads these files and the content is written to a terminal.

Prerequisites

- You need your AWS credentials, the bucket, and the region to use.

Configure and Run the Graph

Follow the steps below to run the example from the SAP Data Intelligence Modeler UI:

1. In the left panel, select the [Graphs](#) tab and navigate to `com/sap/demo/s3`.
2. Check the connection configuration of the [Write File](#) node: `awsAccessKeyId`, `awsSecretAccessKey`, `bucket`, `region`.
3. Check the configuration of the [Read File](#) node: `awsAccessKeyId`, `awsSecretAccessKey`, `bucket`, `region`.
4. In the tool bar, select [Run](#) (play button).
5. The [Status](#) panel indicates if the graph is running.
6. Use the context menu [Open UI](#) of the [Terminal](#) node to open the terminal.
7. The terminal opens and you see the produced files and their content.

Optional Step: Check the files on S3

1. Connect to S3 and browse the directory `/tmp/demo/` (of the specified bucket) to see the produced files.

3.2.30 Vora SQL Console

This graph shows how to use a Vora Client operator to execute SQL commands on SAP Vora.

Prerequisites

- A running SAP Vora server.

Configure and Run the Graph

1. Set the [Connection](#) property on Vora Client to point to your SAP Vora server.
2. [Save](#) and [Run](#) the graph.

3.3 Graph Snippets

Related Information

[Copy Files \(Full Load\) \[page 652\]](#)
[Copy Files \(Incremental Load\) \[page 652\]](#)
[Load Files into HANA \(Full Load\) \[page 653\]](#)
[Load Files into VORA \(Incremental Load\) \[page 653\]](#)
[Simple Javascript DB-to-File Data Manipulation \[page 654\]](#)
[Simple Javascript File Data Manipulation \[page 655\]](#)
[Simple Javascript File-to-DB Data Manipulation \[page 655\]](#)
[Simple Node.js File Data Manipulation \[page 656\]](#)
[Simple Python File Data Manipulation \[page 657\]](#)
[Simple Python File-to-DB Data Manipulation \[page 657\]](#)
[Simple R File Data Manipulation \[page 658\]](#)
[Simple R File-to-DB Data Manipulation \[page 658\]](#)

3.3.1 Copy Files (Full Load)

This snippet reads all files from a source Data Lake and writes them to a destination Data Lake. The source points to a folder or a single file.

Prerequisites

For this graph, you will need:

- A configured connection to the source and destination Data Lakes.

Configure and Run

1. **List Files** lists all the files in the configured path. The operation takes place only once. It fails if no files are listed.
2. **Read File** reads the content of the listed files and sends to the **Write File**.
3. **Write File** writes the content to the configured destination, using the configured path.
4. **Is Done** indicates that the operation is done. That is, it checks for the `message.lastBatch` and `message.batchParent.lastBatch` attributes, and sends the output to the **Graph Terminator** once both are true.
5. **Graph Terminator** terminates the graph as **completed** once an input is received.

3.3.2 Copy Files (Incremental Load)

This snippet monitors new and modified files from a source Data Lake and writes them to a destination Data Lake. It also does an initial load for all existing files. The source points to a folder or a single file.

Prerequisites

For this graph, you will need:

- A configured connection to the source and destination Data Lakes.

Configure and Run

1. **Monitor Files** monitors the configured path for new and modified files. When the graph starts, all files are listed, because the "Already exists" event is enabled.

2. **Read File** reads the content of the listed files and sends to the **Write File**.
3. **Write File** writes the content to the configured destination, using the configured path.

3.3.3 Load Files into HANA (Full Load)

This snippet loads data from CSV or JSON files stored in a (cloud) storage into a HANA table and terminates once all files are loaded.

Prerequisites

For this graph, you will need:

- A connection to one of the supported (cloud) storages.
- A running SAP HANA database with a matching registered connection.

Configure and Run

1. **List Files** lists all files in the configured path. The operation takes place only once. It fails if no files are listed.
2. **Read File** operator receives any not yet processed file, reading its contents and sending to the SAP HANA Client.
3. **From File** converts the incoming data of type message.file to the type message.
4. **SAP HANA Client** takes the CSV or JSON content and loads it into the configured Table name. It also creates the table if it doesn't exist when the graph starts, according to the Table initialization configurations. In case of a restart it may happen that files are sent twice resulting in duplicate records. To avoid that you may change the insert mode of the HANA Client to "UPSERT" and for that create primary keys on the HANA table.
5. **Stop Handler** checks for the message.lastBatch header, sending a signal to the Graph Terminator.
6. **Graph Terminator** terminates the graph as completed when an input is received.

3.3.4 Load Files into VORA (Incremental Load)

Continuously ingests data from CSV or JSON files stored in a (cloud) storage into a VORA streaming table by first processing all existing files and then monitoring the directory for any new or modified files.

Prerequisites

For this graph, you will need:

- A connection to one of the supported (cloud) storages.
- A running SAP HANA database with a matching registered connection.

Configure and Run

1. **Monitor Files** monitors the configured path for new and modified files matching the `products.*\*.csv` regular expression. When the graph starts, all files are listed once, because the "Already exists" event is also enabled.
2. **Read File** operator receives any not yet processed file, reading its contents and sending to the **Vora Avro Ingestor**.
3. **Vora Avro Ingestor** operator receives the files and loads them to a Vora streaming table. It uses `ingestionMode` as **UPSERT**, avoiding duplicate data.

Optional Step

Configure the **Multiplicity** of the Group to control how many parallel instances shall read files and ingest the data into HANA.

3.3.5 Simple Javascript DB-to-File Data Manipulation

Manipulation of data from an SAP HANA database table using plain Javascript (ECMAScript 2009 / ES5) and writing the manipulated data to a file.

Prerequisites

For this graph, you will need:

- a connection to an SAP HANA database.
- a database table in SAP HANA containing data.

Configure and Run

1. Enter the SQL statement that reads the input data from the SAP HANA database table in the [Content](#) configuration parameter of the **SQL Statement** operator.
2. Choose the connection for the input data in the [Connection](#) configuration parameter of the **Get Data from DB** operator.
3. Click on the [Script](#) button of the [Process Data](#) operator and enter the Javascript code that performs the data manipulation. Check the documentation for details. Please note that the operator supports ECMAScript 2009 (ES5) specifications.
4. Enter the file path for the data manipulation result in the [Path](#) configuration parameter of the **Write Results File** operator.

5. Run the resulting graph by clicking the [Run](#) button.

3.3.6 Simple Javascript File Data Manipulation

File data manipulation using plain Javascript (ECMAScript 2009 / ES5) and writing the manipulated data back to another file.

Prerequisites

For this graph, you will need:

- a file containing data.

Configure and Run

1. Click on the Script button of the **Process Data** operator and enter the Javascript code that performs the data manipulation. Consult the operator documentation for details. Please note that the operator supports ECMAScript 2009 (ES5) specifications.
2. Enter the path of the file containing the input data in the **Path** configuration parameter of the **Read Input File** operator.
3. Enter the file path for the data manipulation result in the **Path** configuration parameter of the **Write Results File** operator.
4. Run the resulting graph by clicking the [Run](#) button.

3.3.7 Simple Javascript File-to-DB Data Manipulation

File data manipulation using plain Javascript (without any libraries) and writing the manipulated data to an SAP HANA database table.

Prerequisites

For this graph, you will need:

- a connection to an SAP HANA database.
- a file containing data.

Configure and Run

1. Click on the Script button of the **Process Data** operator and enter the Javascript code that performs the data manipulation. Consult the operator documentation for details. Please note that the operator supports ECMAScript 2009 (ES5) specifications.
2. Enter the path of the file containing the input data in the Path configuration parameter of the **Read Input File** operator.
3. Configure the database connection and the target table for the data manipulation result in the [Connection](#), [Table Name](#), and [Table Columns](#) configuration parameters of the **Write to HANA DB** operator.
4. Run the resulting graph by clicking the [Run](#) button.

3.3.8 Simple Node.js File Data Manipulation

File data manipulation using Node.js and writing the manipulated data back to another file.

Prerequisites

For this graph, you will need:

- a file containing data.

Configure and Run

1. Click on the Script button of the **Process Data** operator and enter the Node/Javascript code that performs the data manipulation. Consult the operator documentation for details..
2. Enter the path of the file containing the input data in the **Path** configuration parameter of the **Read Input File** operator.
3. Enter the file path for the data manipulation result in the **Path** configuration parameter of the **Write Results File** operator.
4. Run the resulting graph by clicking the [Run](#) button.

3.3.9 Simple Python File Data Manipulation

File data manipulation using plain Python with only built-in modules and writing the manipulated data back to another file.

Prerequisites

For this graph, you will need:

- a file containing data.

Configure and Run

1. Click on the Script button of the **Process Data** operator and enter the Python code that performs the data manipulation. Consult the operator documentation for details..
2. Enter the path of the file containing the input data in the **Path** configuration parameter of the **Read Input File** operator.
3. Enter the file path for the data manipulation result in the **Path** configuration parameter of the **Write Results File** operator.
4. Run the resulting graph by clicking the **Run** button.

3.3.10 Simple Python File-to-DB Data Manipulation

Prerequisites

For this graph, you will need:

- a connection to the SAP Data IntelligenceSAP Data Hub built-in Vora database.
- a file containing data.

Configure and Run

1. Click on the Script button of the **Process Data** operator and enter the Python code that performs the data manipulation. Consult the operator documentation for details.
2. Enter the path of the file containing the input data in the Path configuration parameter of the **Read Input File** operator.

3. Configure the database connection and the target table for the data manipulation result in the [connectionType](#) and [Connection](#) configuration parameters of the **Write to Vora DB** operator.
4. Run the resulting graph by clicking the [Run](#) button.

3.3.11 Simple R File Data Manipulation

This graph exemplifies file data manipulation using plain R (using only built-in libraries) and writing the manipulated data back to another file.

Prerequisites

For this graph, you will need:

- a file containing data.

Configure and Run

1. Click on the Script button of the **Process Data** operator and enter the R code that performs the data manipulation. Consult the operator documentation for details..
2. Enter the path of the file containing the input data in the [Path](#) configuration parameter of the **Read Input File** operator.
3. Enter the file path for the data manipulation result in the [Path](#) configuration parameter of the **Write Results File** operator.
4. Run the resulting graph by clicking the [Run](#) button.

3.3.12 Simple R File-to-DB Data Manipulation

File data manipulation using plain R (using only built-in libraries) and writing the manipulated data to an SAP HANA database table.

Prerequisites

For this graph, you will need:

- a connection to an SAP HANA database.
- a file containing data.

Configure and Run

1. Click on the Script button of the **Process Data** operator and enter the R code that performs the data manipulation. Consult the operator documentation for details.
2. Enter the path of the file containing the input data in the Path configuration parameter of the **Read Input File** operator.
3. Configure the database connection and the target table for the data manipulation result in the [Connection](#), [Table Name](#), and [Table Columns](#) configuration parameters of the **Write to Vora DB** operator.
4. Run the resulting graph by clicking the [Run](#) button.

3.4 Ingestion via Flowagent

Related Information

[CDC Graph Generator Sample Graph \[page 659\]](#)

[CSV Producer \[page 660\]](#)

[Flowagent Consumer \[page 661\]](#)

[Ingest OData Query \[page 662\]](#)

[Ingest to Table via Flowagent \[page 663\]](#)

[Ingest Google BigQuery SQL \[page 663\]](#)

[Ingest Google BigQuery Table Producer \[page 664\]](#)

[MySQL Consumer \[page 664\]](#)

[Open Connectors SQL Consumer \[page 670\]](#)

[Open Connectors Table Consumer \[page 671\]](#)

[Table Consumer via Flowagent \[page 672\]](#)

3.4.1 CDC Graph Generator Sample Graph

Demonstrates how to use CDC Graph Generator operator. CDC Graph Generator operator generates the required SQL Scripts to allow customers to capture changes from a database source and the graphs to capture the changes.

See [CDC Graph Generator \(Deprecated\) \[page 543\]](#) for more details on how to configure the operator. This graph provides one example on how it can be used to generate the appropriate files.

When you run this graph, it will generate 3 graphs for you under `/vrep/vflow/graphs/com/sap/dh/cdc/`. You can now search for `cdc` or the actual table name under the graphs to see the newly created graphs.

Components

Defining CDC Graph Generator connections

- `cdcgraphgenerator1`: provides all required source and target definitions.

Loader Definition

- `writefile1`: Creates the graph based on message header for table name, owner and graph type under `/vrep/vflow/graphs/com/sap/dh/cdc/{source.owner}/{source.name}/{source.graph.type}/graph.json`

Related Information

[Improving CDC Graph Generator Operator Performance](#)
[Changing Data Capture \(CDC\)](#)
[CDC Graph Generator Sample Graph \[page 659\]](#)

3.4.2 CSV Producer

This graph demonstrates how to use the CSV Producer operator, mainly detailing the usability of the `outMessage` port.

We are reading from an OData source and using CSV Producer to get data in batches of 100 rows. We are writing all the data into a file using a Write File operators. Because Write File preservse the message headers, we use the JavaScript Message operator to detect when the last batch was written, and then, we send a signal to terminate the graph execution.

Components

Defining Consumer

- `odataqueryconsumer1`: provides an OData consumer information, but any consumer can be used.

Loader Definition

- `flowagentcsvproducer1`: CSV producer, which will output, in batches, the content being read from source.

Merge Header to Body

- `jsmessageoperator1`: operator that will take the attributes header and merge it to the CSV body. This is required by some operators that expect the header to be present in every batch.

Graph Termination

- `jsmessageoperator2`: operator that will wait for the `message.lastBatch` attribute to be set to true, which, then, will output a message so that the graph can be terminated.

Substitution variables

- `OUT_FILE_NAME`: Name of the file to which the table contents will be written

3.4.3 Flowagent Consumer

This graph demonstrates how to use DB2 SQL Consumer. DB2 SQL Consumer is a Flowagent operator and uses Flowagent subengine for execution, thus it needs to be connected to a Flowagent based consumer operator to read the data.

SQL Consumer takes in native SQL statement that you would run on DB2 as input and it will read the response result set from the source piece-wise (N rows per call, defined by fetch size) and load to next operator in line.

Prerequisites

For driver configuration, check the documentation in Connection Management.

Components

Triggering the Graph

- `constantgenerator1`: provide constant input to Consumer, which triggers the ingestion.

Defining DB2 Connection

- `oracletableconsumer1`: provides DB2 consumer connection information. DB2 default port is 1521. If no port is provided in connection definition, the default value will be assigned.

Loader Definition

- `flowagentfileproducer1`: Data Services required to provide a loader either CSV loader to load to Modeler operators.

Substitution Variables

- DB2_HOST: Hostname of the DB2 server
- DB2_PORT: Port of the DB2 server
- DB2_DATABASE_NAME: Database to which the query will be applied to
- DB2_USER: User of the DB2 database
- DB2_PASSWORD: User password of the DB2 database
- SQL_QUERY: Query that will be performed on the DB2 database
- OUT_FILE_NAME: Name of the file to which the query result will be written

3.4.4 Ingest OData Query

OData Query Consumer is based on Flowagent and uses Dataservices as execution engine.

Dataservices provides reading data from OData sources using OData Query Consumer to read the result set of the native OData Query URI.

Query Consumer takes in an OData Query URI that you would run on OData as input and it will read the response result set from the source piece-wise (100 rows per call) and load to the next operator in line.

Example:

- Entity set: Orders
- Projections: `Orders?$select=CustomerID,EmployeeID,Employee/LastName,Employee/FirstName&$expand=Employee`
- Filters: `Orders?$filter=Employee/FirstName eq 'Steven'`

Components

- Defining OData connection
ODataqueryconsumer1: provides OData consumer connection information
- Flowagent Producer
flowagentfileproducer1: Flowagent based consumer requires a flowagent based producer like file, database or CSV producer. The output of the odata is written to `/vrep/odata_<sourceTableName>.csv`
- Terminating Graph
graphterminator1: When the Flowagent producer writes the output to the file, it will terminate the graph.

3.4.5 Ingest to Table via Flowagent

This graph demonstrates how to use Flowagent Table Producer to write to a HANA Table. Flowagent Table Producer is a Flowagent based producer operator, thus it needs to be connected to a Flowagent based consumer operator.

This demo graph demonstrate how the Flowagent Database Producer can be used together with other Flowagent based producers.

In this example, we use the OData Table consumer, reading a table from the Northwind source, and loading its content to a table in HANA.

More details on how the consumers work can be seen in the consumer ingestion sample graphs.

Components

Defining OData Connection

- `odataconsumer1`: provides Odata consumer connection information

Loader Definition

- `flowagentdatabaseproducer1`: provides a loader to write data to database targets

Substitution variables

- `HANA_HOST`: Hostname of the HANA server
- `HANA_PORT`: Port of the HANA server
- `HANA_USER`: User of the HANA database
- `HANA_PASSWORD`: User password of the HANA database
- `TABLE_NAME`: Table to which the data will be inserted to
- `OUT_FILE_NAME`: File to which the table name will be written to in case of success

3.4.6 Ingest Google BigQuery SQL

Google BigQuery SQL Consumer is based on Flowagent and uses Data Services as execution engine.

SQL Consumer takes in native SQL statement that you would run on Google BigQuery as input, and it will read the response result set from the source piece-wise and load to the next operator in line.

Components

- Triggering the graph:
constantgenerator1: Provides constant input of SQL text to Consumer, which also triggers the ingestion.
- Defining connection:
googlebigquerysqlconsumer1: Provides consumer connection information.
- Loader definition:
flowagentfileproducer1: Data Services requires a loader, such as CSV Loader, to load to Modeler operators.

3.4.7 Ingest Google BigQuery Table Producer

Google BigQuery Table Producer Graph is based on Flowagent and uses Data Services as an execution engine. The graph uses the SAP ABAP ODP extractor as sample source data for ingestion.

Table Producer takes in the full path of Google Cloud Storage (GCS) file and loads it into Google BigQuery Table defined as "targetObjectName".

Components

- Source Data:
sapabapodpconsumer1: Data is read from SAP ABAP ODP object.
- Load to GCS:
flowagentfileproducer1: The data is loaded to GCS before sending to Google BigQuery table.
- Defining Connection:
googlebigquerytableproducer1: Provides table producer connection information.

3.4.8 MySQL Consumer

This graph demonstrates how to use MySQL SQL Consumer to perform a SQL query and read it in parallel. MySQL SQL Consumer is a Flowagent operator and uses Flowagent subengine for execution, thus it needs to be connected to a Flowagent based consumer operator to read the data.

SQL Consumer takes in native SQL statement that you would run on MySQL as input and it will read the response result set from the source piece-wise (N rows per call, defined by fetch size) and load to next operator in line.

Prerequisites

See MySQL Client configuration on the "Ingest MySQL Table" graph (`com.sap.dh.ingestion.oracle.tableconsumer`).

Components

Triggering the Graph

- `constantgenerator1`: provide constant input to Consumer, which triggers the ingestion.

Defining Oracle Connection

- `oracletableconsumer1`: provides oracle consumer connection information. MySQL default port is 1521. If no port is provided in connection definition, the default value will be assigned.

Loader Definition

- `flowagentfileproducer1`: Data Services requires to provide a loader either CSV loader to load to Modeler operators.

Substitution variables

- `MYSQL_HOST`: Hostname of the MySQL server.
- `MYSQL_PORT`: Port of the MySQL server.
- `MYSQL_DATABASE_NAME`: Database to which the query will be applied to.
- `MYSQL_USER`: User of the MySQL database.
- `MYSQL_PASSWORD`: User password of the MySQL database.
- `SQL_QUERY`: Query that will be performed on the MySQL database.
- `OUT_FILE_NAME`: Name of the file to which the query result will be written.
- `PARTITION_SPECIFICATION`: Specification of the logical partitions. See the **Partition** section below.
- `NUM_PARTITIONS`: Number of logical partitions specified.

Partition

Partitioning the source data allows us to load data in chunks there by overcome memory pressure and by doing it in parallel we can load data faster and improve overall performance. Below are supported partitioning methods with this operator.

Logical Partition

Logical Partitions are user-defined partitions on how to read data from source. For Table Consumer, the user can choose the partition type then add the partition specification as described below.

- `defaultPartition`: The partition that will fetch all rows that were not filtered by the other partitions (you can disable it by setting `defaultPartition` to false);
- `conditions`: Each condition represents a set of filters, which will be performed on a certain column;
 - `columnExpression`: The column where the filters are applied. The filter can be on a column name or a function (for example, `TRUNC(EMPLOYEE_NAME)`)
 - `type`: The filter type. It can be either "LIST" (to filter exact values) or "RANGE" (to filter a range of values);

- `dataType`: The data type of the elements. It can be "STRING", "NUMBER" or "EXPRESSION";
- `elements`: Each element represents a different filter, and its semantics depends on the "type" (see above).

When more than one condition is presented, it will be performed a cartesian product among the elements of each condition, for example:

- Condition A: [C1, C2]
- Condition B: [C3, C4]
- Resulting filters: [(C1, C3), (C1, C4), (C2, C3), (C2, C4)]

Example: Let's assume we have a source table called EMPLOYEE that has the following schema:

```
CREATE TABLE EMPLOYEE (
  EMPLOYEE_ID NUMBER(5) PRIMARY KEY,
  EMPLOYEE_NAME VARCHAR(32),
  EMPLOYEE_ADMISSION_DATE DATE
);
```

And the following SQL statement:

```
SELECT * FROM EMPLOYEE E $partition_specification
```

Range

A JSON representing a RANGE partition is shown below, where the elements represent ranges, thus, they need to be sorted in ascending order.

The default partition will filter values higher or equal to the last defined interval. Additionally, it will include any null values in the column.

Numeric partitions:

```
{
  "defaultPartition": true,
  "conditions": [
    {
      "columnExpression": "EMPLOYEE_ID",
      "type": "RANGE",
      "dataType": "NUMBER",
      "elements": [
        "10",
        "20",
        "30"
      ]
    }
  ]
}
```

This will generate the following SQL statements:

```
SELECT * FROM EMPLOYEE WHERE EMPLOYEE_ID < 10
SELECT * FROM EMPLOYEE WHERE EMPLOYEE_ID >= 10 AND EMPLOYEE_ID < 20
SELECT * FROM EMPLOYEE WHERE EMPLOYEE_ID >= 20 AND EMPLOYEE_ID < 30
SELECT * FROM EMPLOYEE WHERE EMPLOYEE_ID >= 30 OR EMPLOYEE_ID IS NULL
```

String partitions:

```
{
  "defaultPartition": true,
  "conditions": [
    {
```

```

        "columnExpression": "EMPLOYEE_NAME",
        "type": "RANGE",
        "dataType": "STRING",
        "elements": [
            "M",
            "T"
        ]
    }
]
}

```

This will generate the following SQL statements:

```

SELECT * FROM EMPLOYEE WHERE EMPLOYEE_NAME < 'M'
SELECT * FROM EMPLOYEE WHERE EMPLOYEE_NAME >= 'M' AND EMPLOYEE_NAME < 'T'
SELECT * FROM EMPLOYEE WHERE EMPLOYEE_NAME >= 'T' OR EMPLOYEE_NAME IS NULL

```

Expression partitions:

```

{
  "defaultPartition": true,
  "conditions": [
    {
      "columnExpression": "EMPLOYEE_ADMISSION_DATE",
      "type": "RANGE",
      "dataType": "EXPRESSION",
      "elements": [
        "TO_DATE('2012-01-01', 'YYYY-MM-DD')",
        "TO_DATE('2015-01-01', 'YYYY-MM-DD')"
      ]
    }
  ]
}

```

This will generate the following SQL statements:

```

SELECT * FROM EMPLOYEE WHERE EMPLOYEE_ADMISSION_DATE < TO_DATE('2012-01-01',
'YYYY-MM-DD')
SELECT * FROM EMPLOYEE WHERE EMPLOYEE_ADMISSION_DATE >= TO_DATE('2012-01-01',
'YYYY-MM-DD') AND EMPLOYEE_ADMISSION_DATE < TO_DATE('2015-01-01', 'YYYY-MM-DD')
SELECT * FROM EMPLOYEE WHERE EMPLOYEE_ADMISSION_DATE >= TO_DATE('2015-01-01',
'YYYY-MM-DD') OR EMPLOYEE_ADMISSION_DATE IS NULL

```

List

List partitions can be used to filter exact values. Each element represents a different filter.

Numeric partitions:

```

{
  "defaultPartition": true,
  "conditions": [
    {
      "columnExpression": "EMPLOYEE_ID",
      "type": "LIST",
      "dataType": "NUMBER",
      "elements": [
        "10",
        "20",
        "50"
      ]
    }
  ]
}

```

This will generate the following SQL statements:

```
SELECT * FROM EMPLOYEE WHERE EMPLOYEE_ID = 10
SELECT * FROM EMPLOYEE WHERE EMPLOYEE_ID = 20
SELECT * FROM EMPLOYEE WHERE EMPLOYEE_ID = 50
SELECT * FROM EMPLOYEE WHERE EMPLOYEE_ID != 10 AND EMPLOYEE_ID != 20 AND
EMPLOYEE_ID != 50 OR EMPLOYEE_ID IS NULL
```

String partitions:

```
{
  "defaultPartition": false,
  "conditions": [
    {
      "columnExpression": "EMPLOYEE_NAME",
      "type": "LIST",
      "dataType": "STRING",
      "elements": [
        "Jhon",
        "Ana",
        "Beatrice"
      ]
    }
  ]
}
```

This will generate the following SQL statements:

```
SELECT * FROM EMPLOYEE WHERE EMPLOYEE_NAME = 'Jhon'
SELECT * FROM EMPLOYEE WHERE EMPLOYEE_NAME = 'Ana'
SELECT * FROM EMPLOYEE WHERE EMPLOYEE_NAME = 'Beatrice'
```

Expression partitions:

```
{
  "defaultPartition": false,
  "conditions": [
    {
      "columnExpression": "TRUNC(EMPLOYEE_ADMISSION_DATE)",
      "type": "LIST",
      "dataType": "EXPRESSION",
      "elements": [
        "TO_DATE('2012-07-17', 'YYYY-MM-DD')"
      ]
    }
  ]
}
```

This will generate the following SQL statements:

```
SELECT * FROM EMPLOYEE WHERE TRUNC(EMPLOYEE_ADMISSION_DATE) =
TO_DATE('2012-07-17', 'YYYY-MM-DD')
```

Combined:

```
{
  "defaultPartition": true,
  "conditions": [
    {
      "columnExpression": "EMPLOYEE_NAME",
      "type": "LIST",
      "dataType": "NUMBER",
      "elements": [
```

```

        "Beatrice",
        "Ana"
    ]
},
{
    "columnExpression": "EMPLOYEE_ADMISSION_DATE",
    "type": "RANGE",
    "dataType": "EXPRESSION",
    "elements": [
        "TO_DATE('2015-01-01', 'YYYY-MM-DD')",
        "TO_DATE('2016-01-01', 'YYYY-MM-DD')"
    ]
}
]
}

```

This will generate the following SQL statements:

```

SELECT * FROM EMPLOYEE WHERE EMPLOYEE_NAME = 'Beatrice' AND
EMPLOYEE_ADMISSION_DATE < TO_DATE('2015-01-01', 'YYYY-MM-DD')
SELECT * FROM EMPLOYEE WHERE EMPLOYEE_NAME = 'Beatrice' AND
EMPLOYEE_ADMISSION_DATE >= TO_DATE('2015-01-01', 'YYYY-MM-DD') AND
EMPLOYEE_ADMISSION_DATE < TO_DATE('2016-01-01', 'YYYY-MM-DD')
SELECT * FROM EMPLOYEE WHERE EMPLOYEE_NAME = 'Ana' AND EMPLOYEE_ADMISSION_DATE <
TO_DATE('2015-01-01', 'YYYY-MM-DD')
SELECT * FROM EMPLOYEE WHERE EMPLOYEE_NAME = 'Ana' AND EMPLOYEE_ADMISSION_DATE
>= TO_DATE('2015-01-01', 'YYYY-MM-DD') AND EMPLOYEE_ADMISSION_DATE <
TO_DATE('2016-01-01', 'YYYY-MM-DD')
SELECT * FROM EMPLOYEE WHERE (EMPLOYEE_ADMISSION_DATE >= TO_DATE('2016-01-01',
'YYYY-MM-DD') OR EMPLOYEE_ADMISSION_DATE IS NULL) OR (EMPLOYEE_NAME !=
'Beatrice' AND EMPLOYEE_NAME != 'Ana' OR EMPLOYEE_NAME IS NULL)

```

Note

- Operator auto-generates the default partition, so the user does not need to define it, but it can be disabled by setting the property "defaultPartition": false.
- For range partition the default partition is greater than last element. So ensure that the elements are ordered in ascending order.

Parallel load with partitioning

When an operator has partitions, it will send SQL statements in the output port equal to the number of partitions (including the default partition), each SQL with a WHERE condition equivalent to the partition. In order to enable parallel load of partitions, it is necessary to put the next Flowagent producer in the pipeline inside a group with multiplicity higher than one, preferentially with multiplicity equal to the number of partitions to enable parallel load of all partitions at the same time.

Note that each producer inside the group will generate an output. If you want to prevent the next operator after the producer to be triggered before all partitions have finished (example: Graph Terminator), connect the producer to a `Constant Generator` with `counter` equal to the number of partitions.

Partition recovery

Currently there are none automatic recovery mechanism in place. In case any partition fail during the load, the whole graph will stop (unless the error port of the producer is mapped). Here are some ideas for partition recovery:

- Retrigger the graph with table producer mode set to truncate or overwrite
- Use a Javascript operator to retrigger only the failed partitions.

3.4.9 Open Connectors SQL Consumer

This graph demonstrates how to use Open Connectors SQL Consumer. Open Connectors SQL Consumer is a Flowagent operator and uses Flowagent sub engine for execution, thus it needs to be connected to a Flowagent based consumer operator in order to read the data.

Table Consumer takes in a standard SQL as input and it will read the table from the source piece-wise (N rows per call) and load to next operator in line.

Components

Defining Open Connectors Connection

- `openconnectorssqlconsumer1`: Provides consumer connection information. Choose an existed Open Connectors connection from Configuration Manager or replace substitution variables listed in below Substitution variables section. Input a standard SQL into the `native_sql_statement` field to load and filter data. Supported SQL syntax is limited. See Open Connectors SQL Consumer operator document for details about supported syntax.

Flowagent Producer

- `tomessageconverter1`: Convert string output of the `flowagentcsvproducer1` to message type, which is required by a Write File operator.
- `writefile1`: Write the input content to a file. File name can be changed in path configuration.

Terminating Graph

- `graphterminator1`: When the Flowagent producer writes all data to the file, it will terminate the graph.

Substitution variables

- `OCN_INSTANCE_ID`: Open Connectors instance ID
- `OCN_BASE_URL`: Open Connectors API base URL
- `OCN_USER_SECRET`: Open Connectors user secret
- `OCN_ORG_SECRET`: Open Connectors organization secret
- `SQL_STATEMENT`: Standard SQL statement to query and filter data, like `select Name from UserRole where Name like 'VP%'`

3.4.10 Open Connectors Table Consumer

This graph demonstrates how to use Open Connectors Table Consumer. Open Connectors Table Consumer is a Flowagent operator and uses Flowagent sub engine for execution, thus it needs to be connected to a Flowagent based consumer operator to read the data.

Table Consumer takes in a table name as input and it will read the table from the source piece-wise (N rows per call) and load to next operator in line.

Components

Defining Open Connectors connection

- `openconnectorstableconsumer1`: Provides consumer connection information. Choose an existed Open Connectors connection from Configuration Manager or replace substitution variables listed in below Substitution variables section. Specify table name in the `table_name` field to load all data from that table.

Flowagent Producer

- `flowagentcsvproducer1`: Flowagent based consumer requires a Flowagent based producer like file, database or CSV producer. Data will be loaded by Flowagent and outputs as CSV format.

Write file

- `tomessageconverter1`: Convert string output of the `flowagentcsvproducer1` to message type, which is required by a Write File operator.
- `writefile1`: Write the input content to a file. File name can be changed in path configuration.

Terminating Graph

- `graphterminator1`: When the Flowagent producer writes all data to the file, it will terminate the graph.

Substitution variables

- `OCN_INSTANCE_ID`: Open Connectors instance ID
- `OCN_USER_SECRET`: Open Connectors user secret
- `OCN_ORG_SECRET`: Open Connectors organization secret
- `TABLE_NAME`: Dataset name in Open Connectors

3.4.11 Table Consumer via Flowagent

This graph demonstrates how to use Oracle Table Consumer. Oracle Table Consumer is a Flowagent operator and uses Flowagent subengine for execution, thus it needs to be connected to a Flowagent based consumer operator to read the data.

Table Consumer takes in schema.tablename as input and it will read the table from the source piece-wise (N rows per call, defined by fetch size) and load to next operator in line.

Prerequisites

For driver configuration, check the documentation in Connection Management.

Components

Triggering the Graph

- `constantgenerator1`: provide constant input to Consumer, which triggers the ingestion.

Defining Oracle Connection

- `oracletableconsumer1`: provides oracle consumer connection information. Oracle default port is 1521. If no port is provided in connection definition, the default value will be assigned.

Loader Definition

- `flowagentfileproducer1`: Data Services require to provide a loader either CSV loader to load to Modeler operators.

Substitution variables

- `ORACLE_HOST`: Hostname of the Oracle server
- `ORACLE_PORT`: Port of the Oracle server
- `ORACLE_DATABASE_NAME`: Database that contains the table
- `ORACLE_USER`: User of the Oracle database
- `ORACLE_PASSWORD`: User password of the Oracle database
- `TABLE_NAME`: Name of the table which will be read
- `OUT_FILE_NAME`: Name of the file to which the table contents will be written

Partition

Partitioning the source data allows us to load data in chunks there by overcome memory pressure and by doing it in parallel we can load data faster and improve overall performance. Below are supported partitioning methods with this operator.

Logical Partition

Logical Partitions are user-defined partitions on how to read data from source. For Table Consumer, the user can choose the partition type then add the partition specification as described below.

- **defaultPartition:** The partition that will fetch all rows that were not filtered by the other partitions (you can disable it by setting defaultPartition to false);
- **conditions:** Each condition represents a set of filters, which will be performed on a certain column;
 - **columnExpression:** The column where the filters are applied. The filter can be on a column name or a function (for example, `TRUNC(EMPLOYEE_NAME)`)
 - **type:** The filter type. It can be either "LIST" (to filter exact values) or "RANGE" (to filter a range of values);
 - **dataType:** The data type of the elements. It can be "STRING", "NUMBER" or "EXPRESSION";
 - **elements:** Each element represents a different filter, and its semantics depends on the "type" (see above).

When more than one condition is presented, it will be performed a cartesian product among the elements of each condition, for example:

- Condition A: [C1, C2]
- Condition B: [C3, C4]
- Resulting filters: [(C1, C3), (C1, C4), (C2, C3), (C2, C4)]

Example: Let's assume we have a source table called EMPLOYEE that has the following schema:

```
CREATE TABLE EMPLOYEE(  
  EMPLOYEE_ID NUMBER(5) PRIMARY KEY,  
  EMPLOYEE_NAME VARCHAR(32),  
  EMPLOYEE_ADMISSION_DATE DATE  
);
```

Range

A JSON representing a RANGE partition is shown below, where the elements represent ranges, thus, they need to be sorted in ascending order.

The default partition will filter values higher or equal to the last defined interval.

Numeric partitions:

```
{  
  "defaultPartition": true,  
  "conditions": [  
    {  
      "columnExpression": "EMPLOYEE_ID",  
      "type": "RANGE",  
      "dataType": "NUMBER",  
      "elements": [  
        "10",  
        "20",  
        "30"  
      ]  
    }  
  ]  
}
```

```

    }
  ]
}

```

This will generate the following SQL statements:

```

SELECT * FROM EMPLOYEE WHERE EMPLOYEE_ID < 10
SELECT * FROM EMPLOYEE WHERE EMPLOYEE_ID >= 10 AND EMPLOYEE_ID < 20
SELECT * FROM EMPLOYEE WHERE EMPLOYEE_ID >= 20 AND EMPLOYEE_ID < 30
SELECT * FROM EMPLOYEE WHERE EMPLOYEE_ID >= 30 OR EMPLOYEE_ID IS NULL

```

String partitions:

```

{
  "defaultPartition": true,
  "conditions": [
    {
      "columnExpression": "EMPLOYEE_NAME",
      "type": "RANGE",
      "dataType": "STRING",
      "elements": [
        "M",
        "T"
      ]
    }
  ]
}

```

This will generate the following SQL statements:

```

SELECT * FROM EMPLOYEE WHERE EMPLOYEE_NAME < 'M'
SELECT * FROM EMPLOYEE WHERE EMPLOYEE_NAME >= 'M' AND EMPLOYEE_NAME < 'T'
SELECT * FROM EMPLOYEE WHERE EMPLOYEE_NAME >= 'T' OR EMPLOYEE_NAME IS NULL

```

Expression partitions:

```

{
  "defaultPartition": true,
  "conditions": [
    {
      "columnExpression": "EMPLOYEE_ADMISSION_DATE",
      "type": "RANGE",
      "dataType": "EXPRESSION",
      "elements": [
        "TO_DATE('2012-01-01', 'YYYY-MM-DD')",
        "TO_DATE('2015-01-01', 'YYYY-MM-DD')"
      ]
    }
  ]
}

```

This will generate the following SQL statements:

```

SELECT * FROM EMPLOYEE WHERE EMPLOYEE_ADMISSION_DATE < TO_DATE('2012-01-01',
'YYYY-MM-DD')
SELECT * FROM EMPLOYEE WHERE EMPLOYEE_ADMISSION_DATE >= TO_DATE('2012-01-01',
'YYYY-MM-DD') AND EMPLOYEE_ADMISSION_DATE < TO_DATE('2015-01-01', 'YYYY-MM-DD')
SELECT * FROM EMPLOYEE WHERE EMPLOYEE_ADMISSION_DATE >= TO_DATE('2015-01-01',
'YYYY-MM-DD') OR EMPLOYEE_ADMISSION_DATE IS NULL

```

List

List partitions can be used to filter exact values. Each element represents a different filter.

Numeric partitions:

```
{
  "defaultPartition": true,
  "conditions": [
    {
      "columnExpression": "EMPLOYEE_ID",
      "type": "LIST",
      "dataType": "NUMBER",
      "elements": [
        "10",
        "20",
        "50"
      ]
    }
  ]
}
```

This will generate the following SQL statements:

```
SELECT * FROM EMPLOYEE WHERE EMPLOYEE_ID = 10
SELECT * FROM EMPLOYEE WHERE EMPLOYEE_ID = 20
SELECT * FROM EMPLOYEE WHERE EMPLOYEE_ID = 50
SELECT * FROM EMPLOYEE WHERE EMPLOYEE_ID != 10 AND EMPLOYEE_ID != 20 AND
EMPLOYEE_ID != 50 OR EMPLOYEE_ID IS NULL
```

String partitions:

```
{
  "defaultPartition": false,
  "conditions": [
    {
      "columnExpression": "EMPLOYEE_NAME",
      "type": "LIST",
      "dataType": "STRING",
      "elements": [
        "Jhon",
        "Ana",
        "Beatrice"
      ]
    }
  ]
}
```

This will generate the following SQL statements:

```
SELECT * FROM EMPLOYEE WHERE EMPLOYEE_NAME = 'Jhon'
SELECT * FROM EMPLOYEE WHERE EMPLOYEE_NAME = 'Ana'
SELECT * FROM EMPLOYEE WHERE EMPLOYEE_NAME = 'Beatrice'
```

Expression partitions:

```
{
  "defaultPartition": false,
  "conditions": [
    {
      "columnExpression": "TRUNC(EMPLOYEE_ADMISSION_DATE)",
      "type": "LIST",
      "dataType": "EXPRESSION",
      "elements": [
        "TO_DATE('2012-07-17', 'YYYY-MM-DD')"
      ]
    }
  ]
}
```

```
}
```

This will generate the following SQL statements:

```
SELECT * FROM EMPLOYEE WHERE TRUNC(EMPLOYEE_ADMISSION_DATE) =  
TO_DATE('2012-07-17', 'YYYY-MM-DD')
```

Combined:

```
{  
  "defaultPartition": true,  
  "conditions": [  
    {  
      "columnExpression": "EMPLOYEE_NAME",  
      "type": "LIST",  
      "dataType": "NUMBER",  
      "elements": [  
        "Beatrice",  
        "Ana"  
      ]  
    },  
    {  
      "columnExpression": "EMPLOYEE_ADMISSION_DATE",  
      "type": "RANGE",  
      "dataType": "EXPRESSION",  
      "elements": [  
        "TO_DATE('2015-01-01', 'YYYY-MM-DD')",  
        "TO_DATE('2016-01-01', 'YYYY-MM-DD')"  
      ]  
    }  
  ]  
}
```

This will generate the following SQL statements:

```
SELECT * FROM EMPLOYEE WHERE EMPLOYEE_NAME = 'Beatrice' AND  
EMPLOYEE_ADMISSION_DATE < TO_DATE('2015-01-01', 'YYYY-MM-DD')  
SELECT * FROM EMPLOYEE WHERE EMPLOYEE_NAME = 'Beatrice' AND  
EMPLOYEE_ADMISSION_DATE >= TO_DATE('2015-01-01', 'YYYY-MM-DD') AND  
EMPLOYEE_ADMISSION_DATE < TO_DATE('2016-01-01', 'YYYY-MM-DD')  
SELECT * FROM EMPLOYEE WHERE EMPLOYEE_NAME = 'Ana' AND EMPLOYEE_ADMISSION_DATE <  
TO_DATE('2015-01-01', 'YYYY-MM-DD')  
SELECT * FROM EMPLOYEE WHERE EMPLOYEE_NAME = 'Ana' AND EMPLOYEE_ADMISSION_DATE  
>= TO_DATE('2015-01-01', 'YYYY-MM-DD') AND EMPLOYEE_ADMISSION_DATE <  
TO_DATE('2016-01-01', 'YYYY-MM-DD')  
SELECT * FROM EMPLOYEE WHERE (EMPLOYEE_ADMISSION_DATE >= TO_DATE('2016-01-01',  
'YYYY-MM-DD') OR EMPLOYEE_ADMISSION_DATE IS NULL) OR (EMPLOYEE_NAME !=  
'Beatrice' AND EMPLOYEE_NAME != 'Ana' OR EMPLOYEE_NAME IS NULL)
```

Note

- Operator auto-generates the default partition, so the user does not need to define it, but it can be disabled by setting the property "defaultPartition": false.
- For range partition the default partition is greater than last element. So ensure that the elements are ordered in ascending order.

Physical Partition

Physical partitions are partitions that are defined directly on the source, using Oracle partitioning concept.

Limitations:

- Hash partitioning is not supported
- Sub-partitioning is not supported

Row ID Partition

Row ID partitions are partitions that are generated automatically based on the row id of the columns. The user must supply the number of partitions, and the range of row id partitions will be generated automatically based on it.

3.5 ML Scenario Manager Templates and Examples

Related Information

[Auto ML Inference Template \[page 678\]](#)

[Auto ML Training Template \[page 678\]](#)

[Basic Training \(TensorFlow Training Template\) \[page 678\]](#)

[HANA ML Inference on Dataset \[page 680\]](#)

[HANA ML Inference on API Requests \[page 680\]](#)

[HANA ML Training Template \[page 681\]](#)

[HANA ML Forecast Template \[page 682\]](#)

[ML Batch Inference \[page 683\]](#)

[ML Multi-Model Inference \[page 684\]](#)

[Python Consumer Template \[page 687\]](#)

[Python Data Preprocessing Example \[page 689\]](#)

[Python Producer Template \[page 691\]](#)

[Pytorch Text Classification \[page 693\]](#)

[R Consumer - Consuming an ML Model via API using R Template \[page 694\]](#)

[R Consumer - Exposing a trained ML model via API Example \[page 696\]](#)

[R Producer - Training an ML Model Using R Template \[page 697\]](#)

[R Producer - Training a classification model using R Example \[page 698\]](#)

[R Producer - Training a regression model using R \[page 700\]](#)

[TensorFlow MNIST Training Example \[page 701\]](#)

[Tensorflow Serving Template \[page 702\]](#)

[TensorFlow 2 and Scikit \[page 702\]](#)

3.5.1 Auto ML Inference Template

i Note

The documentation for this graph is yet to be released.

3.5.2 Auto ML Training Template

i Note

The documentation for this graph is yet to be released.

3.5.3 Basic Training (TensorFlow Training Template)

The Basic Training template provides a basic structure to use the Training operator to train TensorFlow models.

Training often involves one or more artifacts that represent the input datasets. It produces one or more output artifacts that represent the trained model or other generated data.

The `Training` operator can consume an input artifact in two ways:

- Via an input port.
- Via the `Artifacts` configuration

To produce an artifact for a trained model or other output data, you need to create an output port for the artifact on the `Training` operator.

This template illustrates a training case that involves two input artifacts and one output artifact. One of the input artifacts is consumed via the `Artifacts` configuration and the other is consumed via an input port.

Configure and Run the Graph

1. In your machine learning scenario, choose the [Pipelines](#) tab.
2. Choose [Create](#) to create a new pipeline.
3. Enter a name for your pipeline and select the [Basic Training](#) pipeline template from the drop-down list.
4. Choose [Create](#).
5. Click the created pipeline to open the Modeler.
6. Modify the pipeline in the Modeler to fit your own training case. This involves:
 - Configuring input and output artifacts.
If all input artifacts for the training have been created already, you can delete the input port `inputdata2` and modify the `Artifacts` configuration of the `Training` operator to use the existing artifacts.

If no input artifacts have been created and you want to incorporate artifact creation into the current pipeline, you can use `Artifact Producer` operators to create artifacts. Specifically, you add an input port to the `Training` operator for each of the input artifacts. You then link the `outArtifact` output port of an `Artifact Producer` to the input port for the corresponding artifact on the `Training` operator. For more information, see [Artifact Producer \[page 420\]](#). The `Artifacts` configuration in the `Training` operator in the template can be empty.

It is also possible to use existing input artifacts and create new ones at the same time.

For each artifact to be generated by the training, create an output port on the `Training` operator.

- Replacing the script of the `Training` operator in the template with your own training script
- Adding other operators to the pipeline if needed

7. When you have finished making your changes, save the graph.

Go back to [ML Scenario Manager](#), select your pipeline from the list, and click [Execute](#).

Managing Artifacts in Training

You can use the Python SDK `sapdi` to manage artifacts in the training script.

```
import sapdi
from sapdi.artifact import ArtifactKind, ArtifactFileType
```

Consuming Input Artifacts

```
artifact_1 = sapdi.get_artifact(artifact_alias="inputdata1")
artifact_2 = sapdi.get_artifact(artifact_alias="inputdata2")
```

Creating an Artifact for the Trained Model

```
out_artifact = sapdi.create_artifact(
    artifact_alias="model",
    file_type=ArtifactFileType.ZIP,
    artifact_kind=ArtifactKind.MODEL,
    description="Best model ever",
    artifact_name="image_classification"
)
```

3.5.4 HANA ML Inference on Dataset

This is a template graph for the `HANA ML Inference` operator with the option to use a dataset for inference. The template is provided as an example of a pipeline to expose a previously trained model and make inference on a provided dataset.

i Note

The [Wiretap \[page 541\]](#) operator can be replaced by any other operator the user may choose to handle the inference results.

Configure and Run

This graph is intended to be used from the Machine Learning Scenario Manager (MLSM) app.

Follow the steps below to run the template:

1. Open MLSM from the SAP DI Launchpad.
2. Create a scenario.
3. Create a pipeline and choose the [HANA ML Inference on Dataset](#) template. DI will open the Modeler app with the template.
4. Select the [HANA ML Inference](#) operator and configure the following settings (for details, check the operator documentation):
 - Connection
 - Key Column
 - Inference Dataset
5. Save the graph.
6. Go back to MLSM and [execute](#) the pipeline.
7. When the wizard prompts for the model to be used, select a previously trained one for the current scenario.

3.5.5 HANA ML Inference on API Requests

This is a template graph for the `HANA ML Inference` operator. The template is provided as an example of a pipeline to expose a previously trained model as a REST endpoint.

Configure and Run

This graph is intended to be used from the Machine Learning Scenario Manager (MLSM) app.

Follow the steps below to run the template:

1. Open MLSM from the SAP DI Launchpad

2. Create a scenario
 3. Create a pipeline and choose the [HANA ML Inference](#) template. DI will open the Modeler app with the template
 4. Select the [HANA ML Inference](#) operator and configure the following settings (for details, check the operator documentation):
 1. Connection
 2. Key Column
 3. Data Source = **Input Data Port**
 5. Save the graph
 6. Go back to MLSM and [deploy](#) the pipeline
 7. When the wizard prompts for the model to be used, select a previously trained one for the current scenario.
- The request to the endpoint should look like the following:

```
curl --location --request POST 'https://<host>/app/pipeline-modeler/openapi/service/<deploy id>/v1/inference' \
--header 'Content-Type: application/json' \
--header 'If-Match: *' \
--header 'X-Requested-With: Fetch' \
--header 'Authorization: Basic #####' \
--data-raw '{
  "ID": [1,2,3,...],
  "Column1": ["val1","val2","val3"...],
  "Column2": [10,20,30,...]
}'
```

3.5.6 HANA ML Training Template

The HANA ML Training template provides an example of a pipeline that uses the HANA PAL functionality to create a pipeline and store it in the data lake.

Configure and Run

This graph is intended to be used from the Machine Learning Scenario Manager (MLSM) app.

Follow the steps below to run the template:

The request to the endpoint should look like the following:

1. Open MLSM from the SAP DI Launchpad
2. Create a scenario
3. Create a pipeline and choose the HANA ML Inference template
4. DI will open the Modeler app with the template

Select the HANA ML Training operator and configure the following settings (for details, check the operator documentation):

 1. Connection
 2. Training Dataset
 3. Training Features

4. Task
5. Algorithm
6. Test Dataset
7. Hyperparameters (optional)
8. Key Column
9. Target Column
10. Fit Parameters (optional)
5. Save the graph
6. Go back to MLSM and run the pipeline
7. When the wizard prompts for an artifact name, enter the name of the model to be created.

3.5.7 HANA ML Forecast Template

The HANA ML Forecast template provides an example of a pipeline to forecasts data on-the-fly without the need to persist a model.

Configure and Run

This graph is intended to be used from the [Machine Learning Scenario Manager](#) app.

Follow the steps below to run the template:

1. Open [ML Scenario Manager](#) from the launchpad.
2. Create a scenario.
3. Create a pipeline and choose the [HANA ML Forecast](#) template.
The [Modeler](#) app appears containing the template.
4. Select the HANA ML Forecast operator and configure (at least) the following settings (for details, check the operator documentation):

Connection

Algorithm

Key Column

5. Save the graph.
6. Go back to ML Scenario Manager and deploy the pipeline.

❖ Example

The request that can be made to the operator could be as follows:

```
curl --location --request POST 'https://<host>/app/pipeline-modeler/openapi/
service/<deploy id>/v2/inference' \
--header 'Content-Type: application/json' \
--header 'If-Match: *' \
--header 'X-Requested-With: Fetch' \
--header 'Authorization: Basic #####' \
--data-raw '{
  "data": {
    "ID": [1,2,3,...],
    "Y": [10,20,30,...]
```

```
}'  
}
```

Related Information

[HANA ML Forecast Operator \[page 424\]](#)

3.5.8 ML Batch Inference

The ML Batch Inference Example template uses file operators and the Model Serving operator to deploy an ML model and run batch inference for a given set of images. In particular, the graph shows how you can use a Python operator to prepare the input payload for TensorFlow Inception models by preprocessing the input image to base64-encoded JSON and sending it to the Model Serving operator for inference.

The sample *Inception Pre-processor* receives a `message.file` (JPG files) from the *Read File* operator once the pipeline is deployed. It then converts the file into base64-encoded JSON payload (see below) and sends it to the *Model Serving* operator.

```
{  
  "signature_name": "predict_images",  
  "inputs": {  
    "images": [  
      {  
        "b64": "<<<base64 encoded image here>>>"  
      }  
    ]  
  }  
}
```

Before You Start

This example pipeline is a generic template that can be modified to handle any ML model and inference data. However, to run the example as is, you need JPG images for inference and a TensorFlow Inception model. Upload your data to `DI_DATA_LAKE` or a preferred Object Store Service such as S3. Then register the connection in *Connection Manager*. This graph also assumes a model has been trained and registered in *ML Scenario Manager*.

Configure and Run the Graph

This graph is intended to be originated (or created) as an ML Batch Inference pipeline from the *ML Scenario Manager* application.

Follow the steps below to use the template:

1. Open [ML Scenario Manager](#) app from the Fiori launchpad.
2. Select an existing ML scenario or create a new one as required.
3. (Skip this step if you already have a trained model.) Choose the [Models](#) tab and register each of the models that you want to deploy.
4. On the [Pipelines](#) tab, create a new pipeline:
 1. Click [Create](#).
 2. Enter a name for your pipeline and select the [ML Batch Inference Example](#) template from the dropdown list.
 3. Click [Create](#).
The [Modeler](#) application will open in a new tab in your browser. If it doesn't, click the pipeline name to open it.
5. Modify the pipeline in the [Modeler](#) app.
 - [Model Serving Operator](#) has been configured to deploy a single model using the substitution parameter `${ARTIFACT:MODEL}` in Models configuration. If you want to deploy multiple models, add more items to the list. The operator is also configured to accept the model runtime via `${ModelRuntimeId}`.
 - The [List File](#) and [Write File](#) operators are configured with substitute parameter `${InputFilePath}` to accept the directory path containing JPG files from the `DI_DATA_LAKE` connection. Change the [List File](#) and [Write File](#) operators accordingly if you want to use another connection.
 - The [Inception Preprocessor](#) operator is a simple Python3 operator with preprocessing code to create a prediction JSON request for a TensorFlow Inception model. This is sample code that you must modify for productive use.
6. After you have made the required changes, save the graph.
7. Return to [ML Scenario Manager](#), select your pipeline from the list, and click [Execute](#).
8. Enter the pipeline parameters when prompted.
 - Select a model from the dropdown list.
 - Set the `ModelRuntimeId` as `mlserving-1.1` or `tf-1.15`.
 - Set the `InputFilePath` as the actual path of a directory containing the images data. For example, `/shared/my_sample_data/images`.
 - Click [Save](#) to trigger the pipeline.
9. Go back to the [Modeler](#) app to monitor the pipeline status. You can open the UI of the [Wiretap](#) operator for the running pipeline to follow the log messages.
10. Once the pipeline is completed, inference outputs will be written to `${InputFilePath}/results.txt`. For example, if your `${InputFilePath}` is set to `/shared/my_sample_data/images`, prediction outputs from the model will be written to `/shared/my_sample_data/images/results.txt`. You can download this file or view its contents in the [Metadata Explorer](#) app.

3.5.9 ML Multi-Model Inference

The ML Multi-Model Inference template provides an example of an inference pipeline using the Model Serving operator to deploy multiple ML models and serve inference requests via an HTTP/REST endpoint. In particular, the graph shows how an HTTP inference request can be preprocessed by a Python operator to send a message to the Model Serving operator for the relevant model.

The Python *Preprocessor Operator* receives a message from the *OpenAPI Servlow* operator when an HTTP request is invoked and the pipeline is deployed. It does the following:

1. Constructs the message attributes `mso.model_name` and `mso.method` with the values based on the received HTTP request URI and method as follows:

<code>mso.method</code>	If the HTTP request method is GET, the <code>mso.method</code> attribute value is set as <code>info</code>, which returns a list of the deployed models. If the HTTP request method is POST, the <code>mso.method</code> attribute value is set as <code>predict</code> to call the <code>predict</code> method on the model server for the given model name.
<code>mso.model_name</code>	<code>mso.model_name</code> (for which inference to be made) is set as the value parsed from the HTTP request URI path parameter. For example, <code><OpenAPI Servlow URL>/v1/models/<model_name></code> where <code><model_name></code> is one of the model names that was configured in Models configuration. If a single model is deployed, the model name does not have to be passed in the URL path parameter.

Based on the values of `mso.method` and `mso.model_name`, the following methods are called and return a response to the *OpenAPI Servlow* operator:

- `info` lists the models that are deployed on this model server.
 - `predict` is an inference method for the given model name.
2. Calls the *Model Serving Operator*. Based on the values `mso.method` and `mso.model_name`, the following methods are called and return a response to the *OpenAPI Servlow* operator as response.
 - Method to list the models that are deployed on this model server.
 - `predict` to call the `predict` method for the given model name on the model server.

Configure and Run the Graph

This graph is intended to be used from the *ML Scenario Manager* app and must be modified as follows in the *Modeler* app before it is deployed.

1. Open *ML Scenario Manager* app from the Fiori launchpad.
2. Select an existing ML scenario or create a new one as required.
3. (Skip this step if you already have a trained model.) Choose the *Models* tab and register each of the models that you want to deploy.
4. On the *Pipelines* tab, create a new pipeline:
 1. Click *Create*.
 2. Enter a name for your pipeline and select the *ML Multi-Model Inference* template from the dropdown list.
 3. Click *Create*.

5. Click the created pipeline to open the Modeler. The graph should be modified to add the models configuration before this pipeline is deployed.
6. Modify the pipeline in the Modeler to fit your model deployment case. This mainly involves configuring the models as described at [Configure the Models List \[page 686\]](#).

i Note

Use the `Models` configuration to deploy one or more models. If you specify both `artifactId` and `Models`, the graph will not run.

7. After you have made the required changes, save the graph.
8. Return to [ML Scenario Manager](#), select your pipeline from the list, and click [Deploy](#).
9. Provide pipeline parameters such as `modelRuntimeId`, select the models from the dropdown, and save.
10. Monitor the pipeline status from the [Wiretap](#) operator in the Modeler. The deployment is ready for inference when it says `Model is deployed and ready for inference`.
11. Copy the deployment URL from [ML Scenario Manager](#).
This is the URL to call the [OpenAPI Servlow](#) operator. It will receive the HTTP request and forward the request to the [Preprocessor Operator](#) in the template.
The preprocessor constructs `mso.model_name` and `mso.method` message attributes and sends them to the [Model Serving Operator](#) to inference or list the models deployed based on the message attribute values that are sent.

Configure the Models List

The models list is a list of models that are to be served. Each item in the list represents a model and contains two fields: [Name](#) and [ID](#):

- [Name](#) – The value of this field is used as the model name during deployment.
This is an optional field. If it is empty, the ID (`artifactId`) is used as the model name.
The name must be unique in the Models configuration. If a name is duplicated, the graph will become dead.
You can use alphanumeric characters, hyphens, and underscores for the name. It cannot exceed 64 characters.
- [ID](#) – This field receives the `artifactId` value of a model when the pipeline is deployed.
It should be configured as `${ARTIFACT:MODEL:<modelname>}` so that it can be used in [ML Scenario Manager](#) to select from the registered models. `modelname` should be configured as a registered model name in [ML Scenario Manager](#) so that a model with that `modelname` is listed in the dropdown when the pipeline is deployed.

Inference and List Models

Once the pipeline has been deployed, the model server is ready to serve the inference request. The model server can also be queried to list the deployed models.

List Deployed Models

To list the models deployed:

- End point: <Deployment URL from ML Scenario Manager Deployment>/v1/models
- HTTP Method: GET
- Headers
 - Header Key : Authorization
 - Header Value : Basic <Base64 Decoded value of "tenant/username" as "user name" and password>

Inference

- End point: <Deployment URL from ML Scenario Manager Deployment>/v1/models/{model_name}
model_name is the name of a model specified in the Models configuration. Alternatively, choose one from the list endpoint.
- HTTP Method: POST
- Payload Body: Inference data accepted by the model in JSON format
- Headers:

Header Key	Header Value
Authorization	Basic <Base64 Encoded value of "tenant/username" as "user name" and password>
Content-Type	application/json
X-Requested-With	Fetch

3.5.10 Python Consumer Template

The Python Consumer template provides a basic structure to consume a trained ML model and expose it via an API endpoint. The Python3 inference operator loads a model binary from Artifact Producer and receives input data from the OpenAPI Servlow operator.

Prerequisites

- A model binary stored in ML Scenario (you can create one using the Python Producer template).
- Adjusted Python code in Python36 - Inference operator to perform inference using your model.

Configure and Run

1. In your ML scenario, go to the Pipelines tab;

2. Choose Create to create a new pipeline;
3. Input a new name for your pipeline and select the Python Consumer template from the drop-down list;
4. Choose Create;
5. Select your pipeline from the list and select the Deploy button;
6. Optional: write a deployment description;
7. On the Previous Configurations screen, choose the Step 4 button to go to next configuration step;
8. On Pipeline Parameters, select your artifact model from the drop-down list and choose Save;
9. When you get to Deployment screen, check if the status is "Running" and copy the Deployment URL.

Now, your API endpoint is available and you can use a client tool (like Postman) to use your ML model.

Python36 Inference Operator

Input

Input	Type	Description
model	blob	Model in binary format loaded using the Artifact Consumer operator.
input	message	A message type received from API endpoint containing the input data.

Output

Output	Type	Description
output	message	A message type to be returned as API response containing the result of your model inference.

Script

The code example loads the model and checks if the input data is a valid JSON format. In order to process the incoming data, you need to replace the comment "apply your model" by the predict function of your model and create a response using these results.

Postman Configuration

- Authorization Tab
Type: Basic Auth
Username: <tenant>\<user>
Password: <your pass>
- Headers tab

Key	Value
If-Match	*
x-requested-with	Fetch
Content-Type	application/json

- Body Tab

A JSON with your data.

For example:

```
{
  "x0": [0.10, 0.20],
  "x1": [10.0, 20.0],
  "x2": [1, 0]
}
```

- Request

Type: POST

URL: <URL deployment>/v1/uploadjson/

- 'URL deployment' is the URL you copied from step 5 in the `Configure and Run the Graph` section.
- '/v1/uploadjson/' is the path defined in the swaggerSpec field in the OpenAPI Servlow operator.

3.5.11 Python Data Preprocessing Example

The Python Data Preprocessing Example template provides a sample graph that illustrates how to undertake data preprocessing with Python, register a dataset, and submit metrics in an ML scenario.

A file in CSV format is read by the Read File and will be delivered through the outFile output port. The data will be fed into the Python Operator through the ToString converter as string.

In the Python3 operator, metrics are created and the raw dataset is preprocessed.

The metrics are delivered to the Submit Metrics operator, which registers the metrics as part of the scenario.

The dataset is sent to the Artifact Producer operator, which will register the dataset with an artifact ID. In the Configuration, the following parameters are set with the default value:

- Artifact Kind: `dataset`
- Filename Suffix: `.csv`

When both the response of Submit Metrics and the Artifact Producer are delivered to the Operators Complete operator, a message will be sent to the Graph Terminator operator and the graph will complete.

Prerequisites

A CSV file stored in vrep with columns separated by semicolon (;). In the example, the file is saved under `/files/vflow/blobs/com/sap/ml/sampleData/boston_housing_dataset.csv`.

Configure and Run the Graph

1. In your ML scenario, choose the Pipelines tab;
2. Choose Create to create a new pipeline;
3. Write a new name for your pipeline and select the Python Data Preprocessing Example template from the drop-down list;
4. Choose Create;
5. Select your pipeline from the list and press the Execute;
6. Optional: write an execution description;
7. On the Previous Configurations screen, press Step 4 to go to next configuration step;
8. In Pipeline Parameters, write a new name for your model in NewArtifactName field;
9. Choose Save to execute the graph.

Dataset

Here we are using the Boston Housing dataset as an example to do data preprocessing. It is assumed that the last column of the input dataset is the target column. A sample of the CSV file is shown below:

crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	black	lstat	medv
0.00632	18	2.31	0	0.538	6.575	65.2	4.09	1	296	15.3	396.9	4.98	24
0.02731	0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.9	9.14	21.6

Basic Examples

Remove feature with high correlation

```
def remove_high_corr(data, unrelated_cols):
    df_o = data
    df_u = df_o.drop(columns=unrelated_cols)
    cor = df_u.corr()
    cor_cols = [j for i in range(len(df_u.columns)) for j in range(i+1,
len(df_u.columns)) if cor.iloc[i, j] >= 0.9]
    cols = [False if index in cor_cols else True for index in
range(len(df_u.columns))]
    sel_cols = df_u.columns[cols]
    df_new = df_u[sel_cols]
    df_new[unrelated_cols] = df_o[unrelated_cols]
    return df_new
```

This method removes one of the two features that have high correlation with each other ($\text{corr} \geq 0.9$). In the example, the column **tax** is removed due to the high correlation with the column **rad**.

- **Input**
 - data: pandas DataFrame

- `unrelated_cols`: string or list of string, columns that should be excluded for the correlation assessment such as target column
- **Output**
 - `df_new`: new DataFrame with selected Columns

Fill NaN cells

- To replace the **NaN** value with **0**: `df_final = df.fillna(0)`
- To replace the **NaN** value with column **mean**: `df_final = df.fillna(df.mean())`

Remove Outliers

```
def remove_outlier(data, unrelated_cols):
    df_o = data
    df_u = df_o.drop(columns=unrelated_cols)
    df_zscore = (df_u - df_u.mean()) / df_u.std()
    df_new = df_o[(df_zscore < 3).all(axis=1)]
    return df_new
```

This method removes the rows that contain data points more than 3 standard deviations away from the mean. In this example, 67 rows of data are removed.

- **Input**
 - `data`: pandas DataFrame
 - `unrelated_cols`: string or list of string, DataFrame column names that shouldn't be assessed, for example the target column.
- **Output**
 - `df_new`: new DataFrame with removed outliers.

One Hot Encoding

```
def one_hot_encoding(data):
    return pd.get_dummies(data)
```

This method changes the encoding of columns that contain categorical data to numerical data.

- **Input**
 - `data`: pandas DataFrame
- **Output**
 - new pandas DataFrame with encoded numerical data.

3.5.12 Python Producer Template

The Python Producer template provides a basic structure to process data or train models using the Python language.

It consists of four steps, in which it:

1. Reads file from data storage;
2. Executes code in Python language to produce a new artifact and metrics;
3. Saves artifact binary and metrics in the ML Scenario;

4. Successfully completes the graph execution.

Prerequisites

- A file stored in one of the storage services available (check Read File operator for more details).
- Adjusted Python code in Python3 operator to consume the content from Read File and to process the input data.

Configure and Run

1. In your ML Scenario, choose the Pipelines tab;
2. Choose Create to create a new pipeline;
3. Write a new name for your pipeline and select the Python Producer template from the drop-down list;
4. Choose Create;
5. Select your pipeline from the list and press Execute;
6. Optional: write an execution description;
7. On the Previous Configurations screen, select the Step 4 button to go to next configuration step;
8. In Pipeline Parameters, write a new name for your model in NewArtifactName field and the file path in inputFilePath;
9. Choose Save to execute the graph.

Python 3 Operator

Input

Input	Type	Description
Input	string	A string type with the file content provided by the Read File operator.

Output

Output	Type	Description
metrics	message	A key-value pair dictionary containing the metrics and their respective values.
modelBlob	blob	The generated object in binary format.

Script

The code example has two main sections:

- Metrics producer: create a dictionary with one or more metrics;
- Model producer: you can use any Python package available in your docker image to process data or to train a model and save it as a binary object in the ML Scenario.

3.5.13 Pytorch Text Classification

The Pytorch Text Classification Example template illustrates how to use the Training operator to train a supervised learning algorithm for classification using Pytorch, and submit metrics in an ML scenario.

The [Training](#) operator uses the basic resource plan and a docker image tailored for running TensorFlow on CPUs for the training. The trained model is produced as an artifact via the `TORCHMODEL` output port of the operator. In addition, logs generated by the training script are sent via the logs output port. A Python3 operator [Metrics Extractor](#) is used to extract metrics from the training logs. The extracted metrics are delivered to the [Submit Metrics](#) operator, which registers the metrics as part of the scenario.

The [Training](#) operator outputs the model artifact only after all logs generated during training have been output. When the message representing the model artifact is delivered to the [Workflow Terminator](#), the graph will complete.

Configure and Run the Graph

1. In your ML scenario, choose the [Pipelines](#) tab.
2. Click [Create](#) to add a pipeline.
3. Enter a name for your pipeline and select the [Pytorch text classification example](#) template from the dropdown list;
4. Click [Create](#).
5. Select your pipeline from the list and click [Execute](#).

Managing Artifacts in Training

The training script uses the Python SDK `sapdi` to manage artifacts as follows:

```
import sapdi
from sapdi.artifact import ArtifactKind, ArtifactFileType
```

Creating Artifact for the Trained Model

```
out_model_artifact = sapdi.create_artifact(
    artifact_alias="TORCHMODEL",
    file_type=ArtifactFileType.ZIP,
    artifact_kind=ArtifactKind.MODEL,
    description="Pytorch text classification model",
    artifact_name="pytorch_text_classification"
)
model_filename = "model.zip"
model_path = out_model_artifact.get_path()
torch.save(model.state_dict(), model_path+model_filename)
```

3.5.14 R Consumer - Consuming an ML Model via API using R Template

This template provides a basic structure to consume a trained ML model and expose it via API endpoint. The RClient - Inference operator loads a model binary from Artifact Producer and receives input data from the OpenAPI Servlow operator.

Prerequisites

- A model binary stored in ML Scenario (you can create one using the R Producer template).
- Adjust the R code in RClient - Inference operator to perform inference using your model.

Configure and Run

1. In your ML scenario, go to the Pipelines tab;
2. Choose Create to create a new pipeline;
3. Input a new name for your pipeline and select the R Consumer template from the drop-down list;
4. Choose Create;
5. Select your pipeline from the list and select the Deploy button;
6. Optional: write a deployment description;
7. On the Previous Configurations screen, choose the Step 4 button to go to next configuration step;
8. On Pipeline Parameters, select your artifact model from the drop-down list and choose Save;
9. When you get to Deployment screen, check if the status is "Running" and copy the Deployment URL.

Now, your API endpoint is available and you can use a client tool (like Postman) to use your ML model.

RClient Inference operator

Input

Input	Type	Description
model	blob	Model in binary format loaded using the Artifact Consumer operator.
input	message	A message type received from API endpoint containing the input data.

Output

Output	Type	Description
response	message	A message type to be returned as API response containing the result of your model inference.

Script

The code example loads the model and checks if the input data is a valid JSON format. In order to process the incoming data, you need to replace the comment "apply model & generate results" by the predict function of your model and create a response using these results.

Postman Configuration

- Authorization Tab
Type: Basic Auth
Username: <tenant>\<user>
Password: <your pass>
- Headers tab

Key	Value
If-Match	*
x-requested-with	Fetch
Content-Type	application/json

- Body Tab
A JSON with your data.
For example:

```
{
  "x0": [0.10, 0.20],
  "x1": [10.0, 20.0],
  "x2": [1, 0]
}
```

- Request
Type: POST
URL: <URL deployment>/v1/uploadjson/
 - 'URL deployment' is the URL you copied from step 5 in the 'Configure and Run the Graph' section.
 - '/v1/uploadjson/' is the path defined in the swaggerSpec field in the OpenAPI Servlow operator.

3.5.15 R Consumer - Exposing a trained ML model via API Example

This graph loads a trained ML model and exposes it via an API endpoint.

Prerequisites

A trained model from your ML scenario to do inferences. If you don't have any ML models, please check the graph example R producer regression example or R producer classification example to train and create an ML model.

Configure and Run the Graph

1. In your ML scenario, go to the [Pipelines](#) tab;
2. Choose [Create](#) to create a new pipeline;
3. Input a new name for your pipeline and select the [R Consumer Example Template](#) from the drop-down list;
4. Choose [Create](#);
5. Find your pipeline on the list and select the [Deploy](#) button;
6. Optional: write a deployment description;
7. On the [Previous Configurations](#) screen, choose the [Step 4](#) button to go to next configuration step;
8. On [Pipeline Parameters](#), select your artifact model from the drop-down list and choose [Save](#);
9. When you get to [Deployment](#) screen, check if the status is [Running](#) and copy the Deployment URL.

Now, your API endpoint is available and you can use a client tool (like Postman) to use your ML model.

Postman Configuration

Authorization Tab

- Type: Basic Auth;
- Username: <tenant>\<user>;
- Password: <your pass>.

Headers Tab

Key	Value
If-Match	*
x-requested-with	Fetch
Content-Type	application/json

Body Tab

A JSON with your data.

For example:

```
{
  "x0": [0.10, 0.20],
  "x1": [10.0, 20.0],
  "x2": [1, 0]
}
```

Request

- Type: POST
- URL: <URL deployment>/v1/inference/
 - 'URL deployment' is the URL you copied from step 5 in the **Configure and Run the Graph** section.
 - /v1/inference/ is the path defined in the `swaggerSpec` field in the OpenAPI Servlow operator.

3.5.16 R Producer - Training an ML Model Using R Template

This template provides a basic structure to train an ML model using R language.

It consists of four steps:

1. Reads file from data storage;
2. Executes code in R to produce a new artifact and metrics;
3. Saves artifact binary and metrics in the ML Scenario;
4. Completes the graph execution.

Prerequisites

- A file stored in one of the storage services available (check Read File operator for more details).
- Adjusted R code in RClient - Training operator to consume the content from Read File and to process the input data.

Configure and Run

1. In your ML Scenario, choose the *Pipelines* tab;
2. Choose *Create* to create a new pipeline;
3. Write a new name for your pipeline and select the **R Producer** template from the drop-down list;
4. Choose *Create*;
5. Select your pipeline from the list and press *Execute*;
6. Optional: write an execution description;

7. On the [Previous Configurations](#) screen, select the [Step 4](#) button to go to next configuration step;
8. In Pipeline Parameters, write a new name for your model in **NewArtifactName** field and the file path in **inputFilePath**;
9. Choose [Save](#) to execute the graph.

Check out the [Python 3 Operator \[page 356\]](#) documentation for further information.

RClient Training Operator

Input

Input	Type	Description
message	message	A message type with all file information provided by the Read File operator.

Output

Output	Type	Description
metrics	message	A key-value pair dictionary containing the metrics and their respective values
modelBlob	blob	The generated model in binary format.

Script

The code example has three main sections:

- Model producer: you can use any R package available in your docker image to train a model and convert it to binary data;
- Metrics producer: create a JSON with one or more metrics;
- Outputs producer: list containing model binary and metrics.

3.5.17 R Producer - Training a classification model using R Example

This graph consumes data from a CSV file stored in the Local File System (vrep) to train a classification model using the method recursive partitioning from the package rpart.

As an example, we use the Iris dataset to train a model able to classify iris flowers based on a given set of features.

Prerequisites

A CSV file stored in vrep with columns separated by semicolon(';').

Configure and Run the Graph

1. In your ML scenario, go to the [Pipelines](#) tab;
2. Choose [Create](#) to create a new pipeline;
3. Input a new name for your pipeline and select the [R Producer Classification Example](#) from the drop-down list;
4. Choose [Create](#);
5. Find your pipeline on the list and select the [Deploy](#) button;
6. Optional: write a deployment description;
7. On the [Previous Configurations](#) screen, choose the [Step 4](#) button to go to next configuration step;
8. On [Pipeline Parameters](#), write a new name for your model in the **NewArtifactName** field; and choose [Save](#);
9. Execute your graph.

Training Operator

Ports

- Input
 - Input: A message type with all file information provided by the read file operator.
- Output
 - metrics: Model artifact to be saved;
 - modelBlob: Model metrics.

Dataset

The following is a sample of the CSV file with the Iris dataset. It is assumed that the target values are provided in the last column.

sepal_length	sepal_width	petal_length	petal_width	class
5.1	3.5	1.4	0.2	Iris-setosa
7.0	3.2	4.7	1.4	Iris-versicolor
7.2	6.1	6.1	2.5	Iris-virginica

This example graph can be used with any dataset that follows the above structure.

```
14. # Training a model using all columns to predict the last one.
15. cols <- colnames(df_train)
16. cm.fit<-rpart(as.formula(paste(tail(cols, 1), '~.')),data=df_train)
```

Model

The object model is converted to blob type and is output by the operator.

```
22. # Produce a blob model from your training dataset.
23. conn <- rawConnection(raw(0), "w")
24. saveRDS(cm.fit, conn)
25. modelBlob <- rawConnectionValue(conn)
```

Metrics

The second output is a message type with the accuracy of the model. To compute it, 20% of the data is used to evaluation.

3.5.18 R Producer - Training a regression model using R

This graph consumes data from a CSV file stored in the Local File System (vrep) to create a linear regression model using the `lm` function.

In this example, we use the Boston Housing dataset to train a model in order to predict the median value of owner-occupied homes in thousands of dollars.

Prerequisites

A CSV file stored in vrep with columns separated by semicolon(';').

Configure and Run the Graph

1. In your ML scenario, go to the [Pipelines](#) tab;
2. Choose [Create](#) to create a new pipeline;
3. Input a new name for your pipeline and select the [R Producer Regression Example](#) from the drop-down list;
4. Choose [Create](#);
5. Find your pipeline on the list and select the [Deploy](#) button;
6. Optional: write a deployment description;
7. On the [Previous Configurations](#) screen, choose the [Step 4](#) button to go to next configuration step;
8. On [Pipeline Parameters](#), write a new name for your model in the **NewArtifactName** field; and choose [Save](#);
9. Execute your graph.

Training Operator

Ports

- Input
 - Input: A message type with all file information provided by the Read File operator.
- Output
 - metrics: Model artifact to be saved;
 - modelBlob: Model metrics.

Dataset

The following is a sample of the CSV file with Boston Housing dataset. It is assumed that the target values are provided in the last column.

crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	black	lstat	mdev
0.006 32	18	2.31	0	0.538	6.575	65.2	4.09	1	296	15.3	396.9	4.98	24
0.0273 1	0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.9	9.14	21.6

This example graph can be used with any dataset that follows the above structure.

```
8. # Training a model using all columns to predict the last one.
9. cols <- colnames(df_train)
10. lm.fit <- lm(as.formula(paste(tail(cols, 1), '~.')), data=df_train)
```

Model

The object model is converted to blob type and is output by the operator.

```
22. # Produce a blob model from your training dataset.
23. conn <- rawConnection(raw(0), "w")
24. saveRDS(cm.fit, conn)
25. modelBlob <- rawConnectionValue(conn)
```

Metrics

The second output is a message type with the accuracy of the model. To compute it, 20% of the data is used to evaluation.

```
21. # Produce and save metrics in a JSON format.
22. json <- toJSON(data.frame(
23.   'RMSE'=toString(sqrt(mean(lm.fit$residuals^2)))
24. ))
```

3.5.19 TensorFlow MNIST Training Example

i Note

The documentation for this graph is yet to be released.

3.5.20 Tensorflow Serving Template

i Note

The documentation for this graph is yet to be released.

3.5.21 TensorFlow 2 and Scikit

The TensorFlow 2 and Scikit Structured Data Classification template illustrates how to use the Training operator to train a structured data classification model with TensorFlow 2, and to submit metrics in an ML scenario. It uses a small dataset provided by the Cleveland Clinic Foundation for Heart Disease.

Because the dataset is in the order of less than a Megabyte, it is downloaded just before use. The dataset is a single CSV file.

Pandas is used to read the file from the URL and create a dataset that TensorFlow 2 will use. Scikit (sklearn) is used with the `train_test_split` function to split the dataset into training, test, and validation parts. TensorFlow 2 is used through Keras to do the training and produce a trained model that is saved as an artifact via the HEARTMODEL output port of the operator. The [Training](#) operator uses the basic resource plan and a docker image tailored for running TensorFlow 2 on CPUs for the training. In addition, logs generated by the training script are sent via the logs output port. A Python3 operator [Metrics Extractor](#) is used to extract metrics from the training logs. The extracted metrics are delivered to the [Submit Metrics](#) operator, which registers the metrics as part of the scenario.

The [Training](#) operator outputs the model artifact only after all logs generated during training have been output. When the message representing the model artifact is delivered to the [Workflow Terminator](#), the graph will complete.

Configure and Run the Graph

1. In your ML scenario, choose the [Pipelines](#) tab.
2. Click [Create](#) to add a pipeline.
3. Enter a name for your pipeline and select the [Tensorflow 2 and scikit structured data classification](#) template from the dropdown list;
4. Click [Create](#).
5. Select your pipeline from the list and click [Execute](#).

Managing Artifacts in Training

The training script uses the Python SDK `sapdi` to manage artifacts as follows:

```
import sapdi
```

```
from sapdi.artifact.artifact import Artifact, ArtifactKind, ArtifactFileType
```

Creating Artifact for the Trained Model

```
118. out_artifact = sapdi.create_artifact(  
    "HEARTMODEL",  
    file_type=ArtifactFileType.ZIP,  
    description="Tensorflow 2 structured data classification model",  
    artifact_name="HEART_STRUCTURED_DATA_CLASSIFICATION_MODEL",  
    artifact_kind=ArtifactKind.MODEL  
)  
model_path = out_artifact.get_path()  
model.save(model_path, save_format='tf')  
move_model(model_path)
```

3.6 Others

Related Information

[Example for Vora ingestion using avro data from kafka and Go operator \[page 703\]](#)

3.6.1 Example for Vora ingestion using avro data from kafka and Go operator

From a Kafka broker, a series of messages are read, each consisting of Avro records. Each message is passed to the preingestor, which creates a message with record objects. Finally, this message is passed to the ingestor, which stores the records in the Vora disk engine and emits the commit token associated with each message.

Prerequisites

- You need a running vora instance and a kafka broker.

Configure and Run the Graph

Follow the steps below to run this graph from the Data Pipeline UI:

1. In the left panel, select the "Graphs" tab and navigate to `com/sap/demo/vora/ingestion/avro_ingestion_example_disk_go`
2. Check the configuration of the "consumer2" node: brokers and topics
3. Check the configuration of the "ingestor" node: dsn
4. In the tool bar, select "Run" (play button)
5. The "Status" panel indicates if the graph is running.
6. Use the context menu "Open UI" of the "Wiretap" node to open the wiretap.
7. Send Avro messages to the specified Kafka topic.
8. The wiretap opens and you can see the commit tokens.
9. Stop the graph and change the generator's batchSize and run the graph again.

3.7 SAP Integration

Related Information

[SAP Analytics Cloud Push Demo \[page 704\]](#)

[SAP CP EM Consumption Example \[page 706\]](#)

[SAP CP EM Production Example \[page 706\]](#)

[SAP CPI-PI iFlow Example \[page 707\]](#)

3.7.1 SAP Analytics Cloud Push Demo

This example graph shows how data can be pushed to a dataset in SAP Analytics Cloud.

The graph is configured to create an SAP Analytics Cloud dataset. However, it can easily be reconfigured to append to an existing dataset using the SAP Analytics Cloud Formatter operator.

The graph is composed of the following components:

- Read File operator: Used to read files from local storage or major cloud filestores.
- Table Decoder: Parses raw bytes provided by Read File into a CSV-style message.table format, which the next operator reads.
- [SAP Analytics Cloud Formatter \[page 403\]](#): Parses the input message and configuration details, and then produces an output format that is accepted in SAP Analytics Cloud.
- [SAP Analytics Cloud Producer \[page 406\]](#): Handles HTTP requests to the SAP Analytics Cloud application. It also provides a response to SAP Analytics Cloud Formatter, if necessary.

- Wiretaps: Allows insight to data passed between operators.

Prerequisites

To run this graph, ensure the following are available:

- An existing SAP Analytics Cloud tenant: The dataset API must be enabled for your SAP Analytics Cloud application tenant. (Contact your SAP Analytics Cloud application administrator.)
- A CSV File
- A fully configured OAuth client in the SAP Analytics Cloud application:
 - `oauthClientId`: Client ID used for the OAuth2 authentication.
 - `oauthClientSecret`: Client Secret used for the OAuth2 authentication.
 - `oauthTokenUrl`: URL for the address where the OAuth2 authentication is performed.
 - `oauth2AuthUrl`: The authorization URL used by “oauth2” security scheme when the flow is set to “accessCode”.

i Note

The SAP Analytics Cloud API requires OAuth2 Authorization Code Flow for data transfer. It is recommended that you set the refresh token lifetime longer than the expected graph runtime, so the Modeler application is authorized only once per graph run. The token is reused between graphs when the refresh token lifetime is of sufficient length. For some SAP Cloud Analytics tenants, you may need to use wildcards for the redirect URI generated by Data Intelligence. (ex. `https://My_DataIntelligence_host/**`)

Configure and Run the Graph

1. Configure the Read File operator to point to the desired file and connection.
2. Configure the SAP Analytics Cloud Formatter operator by completing the *Dataset Name* and *Description* options.
3. Configure the SAP Analytics Cloud Producer operator by completing the *hostname* and *oauth* parameters provided by the SAP Analytics Cloud application.
4. Save and run the graph.
5. On the first run, authorize the Modeler application by opening the SAP Analytics Cloud Producer operator and following the operator instructions. After the authorization is complete, restart the graph to use your new token.
6. Use the Wiretap operator to view messages returned from the SAP Analytics Cloud application. If your data is not showing up in the SAP Analytics Cloud application, check the Wiretap operator for trace files and error messages.

3.7.2 SAP CP EM Consumption Example

This graph demonstrates the consumption of messages from SAP Cloud Platform Enterprise Messaging.

Prerequisites

For this graph to run, you need

- A running instance of the SAP Cloud Platform Enterprise Messaging service.
- A connection to this service instance configured in Connection Management.

Configure and Run the Graph

1. Open the configuration of the SAP CP EM Consumer operator and
 1. Select the connection to the SAP Cloud Platform Enterprise Messaging service instance for the [CP EM Connection](#) parameter.
 2. Enter the queue from which you want to retrieve messages in the [Source Queue](#) parameter. The queue must exist on the SAP Cloud Platform Enterprise Messaging service instance.
 3. The remaining parameters are optional; if you don't specify them, a 0 value is used by default for each of them.
2. Run the graph by clicking the [Run](#) button.
3. Open the [Status Monitor](#) terminal to see any error messages that might occur during execution of the graph.
4. Open the [Payload Monitor](#) terminal to see the messages received from the SAP Cloud Platform Enterprise Messaging service instance. If you are using the `maxCount` configuration parameter, you may need to disconnect the [Graph Terminator](#) operator to be able to open a terminal before the graph completes.

3.7.3 SAP CP EM Production Example

This graph demonstrates the publication of messages via SAP Cloud Platform Enterprise Messaging.

Prerequisites

For this graph to run, you need

- A running instance of the SAP Cloud Platform Enterprise Messaging service.
- A connection to this service instance configured in Connection Management.

Configure and Run the Graph

1. Open the configuration of the SAP CP EM Consumer operator and
 1. Select the connection to the SAP Cloud Platform Enterprise Messaging service instance for the *CP EM Connection* parameter.
 2. Enter the topic to which you want to publish messages in the *Target Topic* parameter. The topic must conform to the namespace of the SAP Cloud Platform Enterprise Messaging service instance.
 3. The remaining parameters are optional; if you don't specify them, a 0 value is used by default for each of them.
2. Run the graph by clicking the *Run* button.
3. Open the *Status Monitor* terminal to see any error messages that might occur during execution of the graph.
4. Open the *Ack Monitor* terminal to see the number of acknowledged messages sent to the SAP Cloud Platform Enterprise Messaging service instance. If you are using the `maxCount` configuration parameter, you may need to disconnect the *Graph Terminator* operator to be able to open a terminal before the graph completes.

3.7.4 SAP CPI-PI iFlow Example

This is a very basic example graph demonstrating how the SAP CPI-PI iFlow operator interacts with connected operators.

Prerequisites

- A running SAP CPI instance with at least one deployed iFlow that can be triggered via HTTP.

Configure and Run the Graph

Follow the steps below to run the example:

1. In the left panel select the *Graphs* tab, navigate to *SAP Integration* and click on *SAP CPI-PI iFlow* to open the graph.
2. Optional: To not modify the original example, click on the arrow beside the save button and select *Save As*. Save a copy of this graph at a destination of your choice.
3. Before you can run the graph you need to configure it. Click on the *SAP CPI-PI iFlow* operator, in the *Operator* tab, and enter your connection details (you may also create a connection in the connection manager app first and then use this one in here) and the path of your deployed iFlow. Also check whether the set iFlow has an enabled CSRF protection and set the configuration parameter on the operator accordingly.
4. Now click on the *Constant Generator* operator and enter the payload for the *SAP CPI-PI iFlow* operator in the *Content* field.

5. In the tool bar, select [Run](#) (play button).
6. The [Status](#) panel indicates if the graph is running.
7. Use the context menu [Open UI](#) of the [Terminal](#) operator to open the terminal.
8. The terminal opens and you should see the data returned from the triggered iFlow.

3.8 SAP Vora

Related Information

[Direct Avro Ingestion \[page 709\]](#)

[Avro Ingest via Disk \[page 709\]](#)

[Avro Ingest Mod via Disk \[page 710\]](#)

[CSV Ingest2 via Disk \[page 711\]](#)

[CSV Ingest2 via Disk \[page 712\]](#)

[CSV Ingest3 via Disk \[page 712\]](#)

[Direct CSV Ingest2 \[page 713\]](#)

[JSON Ingest2 via Disk \[page 714\]](#)

[Record Ingest via Disk \[page 715\]](#)

[Vora Table Consumer \[page 716\]](#)

[Example for Vora Ingestion Using CSV Data 2 \[page 717\]](#)

[Example for Vora Ingestion Using CSV Data 3 \[page 717\]](#)

[Example for Vora Ingestion Using CSV Data 5 \[page 718\]](#)

[Example for Vora Ingestion Using CSV Data 6 \[page 719\]](#)

[Example for Vora Ingestion Using JSON Data 5 \[page 720\]](#)

[Example for Vora Ingestion Using JSON Data 6 \[page 720\]](#)

[Example for Vora Ingestion Using Record Object Data 2 in Go \[page 721\]](#)

[Example for Vora Ingestion Using Record Object Data 2 in Python \[page 722\]](#)

3.8.1 Direct Avro Ingestion

A series of messages, each consisting of Avro records are read from a Kafka broker. Each message is passed to the Avro ingestor, which directly stores the records in the Vora disk engine and emits the commit token associated with each message.

Prerequisites

You need a running SAP Vora instance and a Kafka broker.

Configure and Run the Graph

Follow the steps below to run the example from the SAP Data Intelligence Modeler UI:

1. In the left panel, select the [Graphs](#) tab and navigate to `com/sap/demo/vora/ingestion/avro_ingestion_direct_example`.
2. Check the configuration of the [consumer2](#) node: brokers and topics.
3. Check the configuration of the [ingestor](#) node: dsn.
4. In the tool bar, select [Run](#) (play button).
5. The [Status](#) panel indicates if the graph is running.
6. Use the context menu [Open UI](#) of the [Wiretap](#) node to open the Wiretap.
7. Send Avro messages to the specified Kafka topic.
8. The Wiretap opens and you see the commit tokens.
9. Stop the graph, change the generator's batchSize, and run the graph again.

3.8.2 Avro Ingest via Disk

A series of messages, each consisting of Avro records are read from a Kafka broker. Each message is passed to the preingestor, which creates a message with record objects. Finally, this message is passed to the ingestor, which stores the records in the Vora disk engine and emits the commit token associated with each message.

Prerequisites

You need a running SAP Vora instance and a Kafka broker.

Configure and Run the Graph

Follow the steps below to run the example from the SAP Data Intelligence Modeler UI:

1. In the left panel, select the [Graphs](#) tab and navigate to `com/sap/demo/vora/ingestion/avro_ingestion_example_disk`.
2. Check the configuration of the [consumer2](#) node: brokers and topics.
3. Check the configuration of the [ingestor](#) node: dsn.
4. In the tool bar, select [Run](#) (play button).
5. The [Status](#) panel indicates if the graph is running.
6. Use the context menu [Open UI](#) of the [Wiretap](#) node to open the wiretap.
7. Send Avro messages to the specified kafka topic.
8. The Wiretap opens and you see the commit tokens.
9. Stop the graph, change the generator's batchSize, and run the graph again.

3.8.3 Avro Ingest Mod via Disk

From a Kafka broker, a series of messages are read, each consisting of Avro records. Each message is passed to the preingestor, which creates a message with record objects. Finally, this message is passed to the ingestor, which stores the records in the Vora disk engine and emits the commit token associated with each message.

Prerequisites

- You need a running Vora instance and a Kafka broker.

Configure and Run the Graph

Follow the steps below to run the example from the SAP Data Intelligence Modeler UI:

1. In the left panel, select the [Graphs](#) tab and navigate to `com/sap/demo/vora/ingestion/avro_ingestion_example_disk_go`.
2. Check the configuration of the [consumer2](#) node: [brokers](#) and [topics](#)
3. Check the configuration of the [ingestor](#) node: [dsn](#)
4. In the tool bar, select [Run](#) (play button).
5. The [Status](#) panel indicates if the graph is running.
6. Use the context menu [Open UI](#) of the [Wiretap](#) node to open the wiretap.
7. Send Avro messages to the specified Kafka topic.
8. The wiretap opens and you see the commit tokens.
9. Stop the graph, change the generator's batchSize, and run the graph again.

3.8.4 CSV Ingest2 via Disk

The data generator generates a series of messages, each consisting of one or more CSV records. Each message is passed to the preingestor, which creates a message with record objects. Finally, this message is passed to the ingestor, which stores the records in the Vora disk engine and emits the commit token associated with each message.

The table definition is derived from the Avro schema configured at the preingestor, which is given as:

```
{ "name": "sample_demo_deep_record", "type": "record", "fields": [ {"name": "idx", "type": "int"}, {"name": "code", "type": "string"}, {"name": "magnitude", "type": "double"}, {"name": "name", "type": "string"}, {"name": "coordinates", "type": "record", "fields": [ {"name": "latitude", "type": "double"}, {"name": "longtitude", "type": "double"}]}, {"name": "ts", "type": "long", "logicalType": "timestamp-millis"}, {"name": "status", "type": "boolean"}] }
```

The corresponding CSV line should represent the fields with the values representing the following field values:

```
idx,code,magnitude,name,coordinates-latitude,coordinates-longtitude,ts,status
```

The datagenerator operator will transmit a message whose body consists of one or more such lines:

```
4284,9ar6S,  
6538.984505939649,f4I4XgsW7Te66n7EVLjFilM8,77.44481090477504,154.91779649284774,2  
018-05-07 2:17:42.849,true 5068,CKqKW,5027.427239619433,o9DdLXkUH,  
54.209264596298,54.42583528731214,2018-05-07 2:17:42.849,false 5783,7spVh,  
1238.6340364821854,WZzhhg3Oape8MVtTEEnLvB,-42.23289035504596,-46.40615837430687,20  
18-05-07 2:17:42.850,false ...
```

Prerequisites

You need a running SAP Vora instance.

Configure and Run the Graph

Follow the steps below to run the example from the SAP Data Intelligence Modeler UI:

1. In the left panel, select the **Graphs** tab and navigate to `com/sap/demo/vora/ingestion/csv_ingestion_example2_disk`.
2. Check the configuration of the **ingestor** node: `dsn`.
3. In the tool bar, select **Run** (play button).
4. The **Status** panel indicates if the graph is running.
5. Use the context menu **Open UI** of the **Wiretap** node to open the wiretap.
6. The wiretap opens and you see the commit tokens.
7. Stop the graph, change the generator's `batchSize`, and run the graph again.

3.8.5 CSV Ingest2 via Disk

This data generator generates a series of messages, each consisting of one or more CSV records. Each message is passed to the preingestor, which creates a message with record objects. Finally, this message is passed to the ingestor, which stores the records in the Vora disk engine and emits the commit token associated with each message.

The table definition is derived from the Avro schema configured at the avroingestor (see `csv_ingestion_example2_disk`).

Prerequisites

- You need a running Vora instance.

Configure and Run the Graph

Follow the steps below to run the example from the SAP Data Intelligence Modeler UI:

1. In the left panel, select the [Graphs](#) tab and navigate to `com/sap/demo/vora/ingestion/csv_ingestion_example2_disk_go`.
2. Check the configuration of the [ingestor](#) node: `dsn`
3. In the tool bar, select [Run](#) (play button).
4. The [Status](#) panel indicates if the graph is running.
5. Use the context menu [Open UI](#) of the [Wiretap](#) node to open the wiretap.
6. The Wiretap opens and you see the commit tokens.
7. Stop the graph, change the generator's `batchSize`, and run the graph again.

3.8.6 CSV Ingest3 via Disk

This sample graph is similar to the sample graph `csv_ingestion_example2_disk`. However, it uses an avro schema with additional extension attributes to customize the table definition.

The table definition is derived from the avro schema configured at the preingestor, which is given as:

```
{ "name": "sample_demo_deep_record3", "type": "record", "fields": [ {"name": "idx", "type": "int", "colName": "_idx"}, {"name": "code", "type": "string", "maxLength": 5}, {"name": "magnitude", "type": "double"}, {"name": "name", "type": "string", "maxLength": 32}, {"name": "coordinates", "type": "record", "fields": [ {"name": "latitude", "type": "double"}, {"name": "longitude", "type": "double"}]}, {"name": "ts", "type": "long", "logicalType": "timestamp-millis"}, {"name": "status", "type": "boolean"}] }
```


In the above avro schema, attribute `colName` is used to customize the column name and attribute `maxLength` is used to customize the size limit. With those attributes, the table columns will result in:

```
_idx INTEGER, code VARCHAR(5), magnitude DOUBLE, name VARCHAR(32),  
coordinates_latitude DOUBLE, coordinates_longitude DOUBLE, ts TIMESTAMP, status  
BOOLEAN
```

Without those attributes, the table columns would result in:

```
idx INTEGER, code VARCHAR(*), magnitude DOUBLE, name VARCHAR(*),  
coordinates_latitude DOUBLE, coordinates_longitude DOUBLE, ts TIMESTAMP, status  
BOOLEAN
```

Prerequisites

You need a running SAP Vora instance.

Configure and Run the Graph

Follow the steps below to run the example from the SAP Data Intelligence Modeler UI:

1. In the left panel, select the [Graphs](#) tab and navigate to `com/sap/demo/vora/ingestion/csv_ingestion_example3_disk`.
2. Check the configuration of the [ingestor](#) node: `dsn`.
3. In the tool bar, select [Run](#) (play button).
4. The [Status](#) panel indicates if the graph is running.
5. Use the context menu [Open UI](#) of the [Wiretap](#) node to open the wiretap.
6. The Wiretap opens and you see the commit tokens.
7. Stop the graph, change the generator's `batchSize`, and run the graph again.

3.8.7 Direct CSV Ingest2

The data generator generates a series of messages, each consisting of one or more CSV records. Each message is passed to the Avro ingestor, which stores the records in the Vora disk engine and emits the commit token associated with each message.

The table definition is derived from the Avro schema configured at the `avroingestor` (see `csv_ingestion_example2_disk`).

Prerequisites

You need a running SAP Vora instance.

Configure and Run the Graph

Follow the steps below to run the example from the SAP Data Intelligence Modeler UI:

1. In the left panel, select the **Graphs** tab and navigate to `com/sap/demo/vora/ingestion/csv_ingestion_direct_example2`.
2. Check the configuration of the **avroingestor** node: dsn.
3. In the tool bar, select **Run** (play button).
4. The **Status** panel indicates if the graph is running.
5. Use the context menu **Open UI** of the **Wiretap** node to open the Wiretap.
6. The Wiretap opens and you see the commit tokens.
7. Stop the graph, change the generator's batchSize, and run the graph again.

3.8.8 JSON Ingest2 via Disk

The data generator generates a series of messages, each consisting of one or more JSON records. Each JSON records may be represented as a JSON object or a JSON value array. If multiple records are included in a single message, these records must be of the same form and included in a JSON array. Each message is passed to the preingestor, which creates a message with record objects. Finally, this message is passed to the ingestor, which stores the records in the Vora disk engine and emits the commit token associated with each message.

The table definition is derived from the Avro schema configured at the preingestor, which is given as:

```
{ "name": "sample_demo_deep_record", "type": "record", "fields": [ {"name": "idx", "type": "int"}, {"name": "code", "type": "string"}, {"name": "magnitude", "type": "double"}, {"name": "name", "type": "string"}, {"name": "coordinates", "type": "record", "fields": [ {"name": "latitude", "type": "double"}, {"name": "longitude", "type": "double"}]}, {"name": "ts", "type": "long", "logicalType": "timestamp-millis"}, {"name": "status", "type": "boolean"}] }
```

The datagenerator operator will transmit a message whose body consists of one or more JSON records that match the above schema.

The corresponding JSON records may be represented as a single record or an array of records, where each record can be represented as a JSON object or a JSON value array.

For example, this can look either like this:

```
[{"code": "iXNWM", "coordinates": {"latitude": 55.87086007693324, "longitude": -41.24651001822619}, "idx": 6077, "magnitude": 2516.2811424004744, "name": "H2VGxtYlahjBHoLcm", "status": false, "ts": "2018-05-07 3:25:00.557"}, {"code": "IZFux", "coordinates": {"latitude": 79.13471154134257, "longitude": 71.62190615872217}, "idx": 3700, "magnitude": 8821.138614425376, "name": "WoamNWLcJAfSgm9m6rdujQ", "status": true, "ts": "2018-05-07 3:25:00.557"}, {"code": "EDa2M", "coordinates": {"latitude": 65.0152956644234, "longitude": -167.62832185615025}, "idx": 530, "magnitude": 9570.581735765665, "name": "a5DnGc7YqIsz", "status": true, "ts": "2018-05-07 3:25:00.557"}, ...]
```

or like this:

```
[[2217, "1KQ9B", 2372.7511416642337, "rRSgQyMUXqPG6UEQuMHkDLxUucg", -6.103202884364208, 63.69302212520921, "2018-05-07 3:29:46.035", false], [1624, "kLzw3",
```

```
3226.511391502805,"zbf3IqsgIXO4jt050U4AfWUI87gKK681",
6.171226722764729,-11.211557814247755,"2018-05-07 3:29:46.035",true],
[7263,"ykQG6",
9657.557856232173,"KtWSrO4eEDM9THVLS",-70.42437579521655,-83.99133796607525,"2018
-05-07 3:29:46.036",true], ... ]
```

Prerequisites

You need a running SAP Vora instance.

Configure and Run the Graph

Follow the steps below to run the example from the SAP Data Intelligence Modeler UI:

1. In the left panel, select the [Graphs](#) tab and navigate to `com/sap/demo/vora/ingestion/json_ingestion_example2_disk`.
2. Check the configuration of the [ingestor](#) node: dsn.
3. In the tool bar, select [Run](#) (play button).
4. The [Status](#) panel indicates if the graph is running.
5. Use the context menu [Open UI](#) of the [Wiretap](#) node to open the Wiretap.
6. The Wiretap opens and you see the commit tokens.
7. Stop the graph, change the generator's batchSize, and run the graph again.

3.8.9 Record Ingest via Disk

The data generator generates a series of messages, each consisting of one or more record objects. Each message is directly passed to the ingestor, which stores the records in the Vora disk engine and emits the commit token associated with each message.

The table definition is not provided as an Avro schema but instead included in the message at attribute key `vora.record.definition`.

Prerequisites

You need a running SAP Vora instance.

Configure and Run the Graph

Follow the steps below to run the example from the SAP Data Intelligence Modeler UI:

1. In the left panel, select the [Graphs](#) tab and navigate to `com/sap/demo/vora/ingestion/rec_ingestion_example2_disk`.
2. Check the configuration of the [ingestor](#) node: dsn.
3. In the tool bar, select [Run](#) (play button).
4. The [Status](#) panel indicates if the graph is running.
5. Use the context menu [Open UI](#) of the [Wiretap](#) node to open the Wiretap.
6. The Wiretap opens and you see the commit tokens.
7. Stop the graph, change the generator's batchSize, and run the graph again.

3.8.10 Vora Table Consumer

This graph demonstrates how to use Vora Table Consumer. Vora Table Consumer is a Flowagent operator and uses Flowagent subengine for execution, thus it needs to be connected to a Flowagent based consumer operator to read the data.

Table Consumer takes in `schema.tablename` as input and it will read the table from the source piece-wise (N rows per call, defined by **fetch size**) and load to next operator in line.

Componets

Defining Vora connection

- `voratableconsumer1`: provides Vora consumer connection information. A Vora connection needs to configure the HANA Wire.

Loader Definition

- `flowagentfileproducer1`: Data Services requires you to provide a CSV loader to load to Modeler operators.

Substitution variables

- `VORA_HOST`: Hostname of the Vora server
- `HANA_WIRED_PORT`: HANA wired port of the Vora server
- `VORA_DATABASE_NAME`: Database that contains the table
- `VORA_TENANT`: Tenant of the Vora instance
- `VORA_USER`: User of the Vora instance
- `VORA_PASSWORD`: User password of the Vora database
- `TABLE_NAME`: Name of the table that will be read
- `OUT_FILE_NAME`: Name of the file to which the table contents will be written

3.8.11 Example for Vora Ingestion Using CSV Data 2

The data generator generates a series of messages, each consisting of one or more CSV records. Each message is passed to the preingestor, which creates a message with record objects. Finally, this message is passed to the ingestor, which stores the records in the Vora disk engine and emits the commit token associated with each message.

The table definition is derived from the Avro schema configured at the avroingestor (see `csv_ingestion_example2_disk`).

Prerequisites

- You need a running Vora instance.

Configure and Run the Graph

Follow the steps below to run the example from the SAP Data Intelligence Modeler UI:

1. In the left panel, select the [Graphs](#) tab and navigate to `com/sap/demo/vora/ingestion/csv_ingestion_example2_disk_go`.
2. Check the configuration of the [ingestor](#) node: `dsn`
3. In the tool bar, select [Run](#) (play button).
4. The [Status](#) panel indicates if the graph is running.
5. Use the context menu [Open UI](#) of the [Wiretap](#) node to open the wiretap.
6. The wiretap opens and you see the commit tokens.
7. Stop the graph and change the generator's `batchSize` and run the graph again.

3.8.12 Example for Vora Ingestion Using CSV Data 3

This is similar to sample graph `csv_ingestion_example2_disk`, which ingests generated CSV records into Vora using the type information specified in the provided Avro schema. It uses, however, an Avro schema with additional extension attributes to customize the table definition.

The table definition is derived from the avro schema configured at the preingestor, which is given as:

```
{ "name": "sample_demo_deep_record3", "type": "record", "fields": [ {"name": "idx", "type": "int", "colName": "_idx"}, {"name": "code", "type": "string", "maxLength": 5}, {"name": "magnitude", "type": "double"}, {"name": "name", "type": "string", "maxLength": 32}, {"name": "coordinates", "type": "record", "fields": [ {"name": "latitude", "type": "double"}, {"name": "longitude", "type": "double"}]}, {"name": "ts", "type": "long", "logicalType": "timestamp-millis"}, {"name": "status", "type": "boolean"}] }
```

In the above Avro schema, attribute `colName` is used to customize the column name and attribute `maxLength` is used to customize the size limit. With those attributes, the table columns will result in:

```
_idx INTEGER, code VARCHAR(5), magnitude DOUBLE, name VARCHAR(32),  
coordinates_latitude DOUBLE, coordinates_longitude DOUBLE, ts TIMESTAMP, status  
BOOLEAN
```

Without those attributes, the table columns would result in:

```
idx INTEGER, code VARCHAR(*), magnitude DOUBLE, name VARCHAR(*),  
coordinates_latitude DOUBLE, coordinates_longitude DOUBLE, ts TIMESTAMP, status  
BOOLEAN
```

Prerequisites

You need a running SAP Vora instance.

Configure and Run the Graph

Follow the steps below to run the example from the SAP Data Intelligence Modeler UI:

1. In the left panel, select the [Graphs](#) tab and navigate to `com/sap/demo/vora/ingestion/csv_ingestion_example3_disk`.
2. Check the configuration of the [ingestor](#) node: `dsn`.
3. In the tool bar, select [Run](#) (play button).
4. The [Status](#) panel indicates if the graph is running.
5. Use the context menu [Open UI](#) of the [Wiretap](#) node to open the wiretap.
6. The wiretap opens and you see the commit tokens.
7. Stop the graph and change the generator's `batchSize` and run the graph again.

3.8.13 Example for Vora Ingestion Using CSV Data 5

This is similar to sample graph `csv_ingestion_direct_example2`. The difference is that the Avro schema is included in the message as a serialized Avro schema instead of being configured at the operator. The table name and partition criteria are customized in the provided schema.

Prerequisites

- You need a running Vora instance.

Configure and Run the Graph

Follow the steps below to run the example from the SAP Data Intelligence Modeler UI:

1. In the left panel, select the [Graphs](#) tab and navigate to `com/sap/demo/vora/ingestion/csv_ingestion_direct_example5`.
2. Check the configuration of the [avroingestor](#) node: [dsn](#)
3. In the tool bar, select [Run](#) (play button).
4. The [Status](#) panel indicates if the graph is running.
5. Use the context menu [Open UI](#) of the [Wiretap](#) node to open the wiretap.
6. The wiretap opens and you see the commit tokens.
7. Stop the graph and change the generator's `batchSize` and run the graph again.

3.8.14 Example for Vora Ingestion Using CSV Data 6

This is similar to sample graph `csv\_ingestion\_direct\_example5`. The difference is that the Avro schema is included in the message as a map representing the Avro schema.

Prerequisites

- You need a running Vora instance.

Configure and Run the Graph

Follow the steps below to run the example from the SAP Data Intelligence Modeler UI:

1. In the left panel, select the [Graphs](#) tab and navigate to `com/sap/demo/vora/ingestion/csv\_ingestion\_direct\_example6`.
2. Check the configuration of the [avroingestor](#) node: [dsn](#)
3. In the tool bar, select [Run](#) (play button).
4. The [Status](#) panel indicates if the graph is running.
5. Use the context menu [Open UI](#) of the [Wiretap](#) node to open the wiretap.
6. The wiretap opens and you see the commit tokens.
7. Stop the graph and change the generator's `batchSize` and run the graph again.

3.8.15 Example for Vora Ingestion Using JSON Data 5

This is similar to sample graph `csv_ingestion_direct_example5`, which ingests generated JSON records into Avro instead of CSV records. The table name and partition criteria are customized in the provided schema.

Prerequisites

- You need a running Vora instance.

Configure and Run the Graph

Follow the steps below to run the example from the SAP Data Intelligence Modeler UI:

1. In the left panel, select the [Graphs](#) tab and navigate to `com/sap/demo/vora/ingestion/json_ingestion_direct_example5`.
2. Check the configuration of the [avroingestor](#) node: [dsn](#)
3. In the tool bar, select [Run](#) (play button).
4. The [Status](#) panel indicates if the graph is running.
5. Use the context menu [Open UI](#) of the [Wiretap](#) node to open the wiretap.
6. The wiretap opens and you see the commit tokens.
7. Stop the graph and change the generator's `batchSize` and run the graph again.

3.8.16 Example for Vora Ingestion Using JSON Data 6

This is similar to sample graph `json\_ingestion\_direct\_example5`. The difference is that the Avro schema is included in the message as a map representing the Avro schema.

Prerequisites

- You need a running Vora instance.

Configure and Run the Graph

Follow the steps below to run the example from the SAP Data Intelligence Modeler UI:

1. In the left panel, select the [Graphs](#) tab and navigate to `com/sap/demo/vora/ingestion/json\_ingestion\_direct\_example6`.

2. Check the configuration of the *avroingestor* node: *dsn*
3. In the tool bar, select *Run* (play button).
4. The *Status* panel indicates if the graph is running.
5. Use the context menu *Open UI* of the *Wiretap* node to open the wiretap.
6. The wiretap opens and you see the commit tokens.
7. Stop the graph and change the generator's `batchSize` and run the graph again.

3.8.17 Example for Vora Ingestion Using Record Object Data 2 in Go

The data generator generates a series of messages, each consisting of one or more record objects.

Each message is directly passed to the ingestor, which stores the records in the Vora disk engine and emits the commit token associated with each message.

The table definition is not provided as an Avro schema but instead included in the message at attribute key `vora.record.definition`.

Prerequisites

- You need a running Vora instance.

Configure and Run the Graph

Follow the steps below to run the example from the SAP Data Intelligence Modeler UI:

1. In the left panel, select the *Graphs* tab and navigate to `com/sap/demo/vora/ingestion/rec_ingestion_example2_disk_go`.
2. Check the configuration of the *avroingestor* node: *dsn*
3. In the tool bar, select *Run* (play button).
4. The *Status* panel indicates if the graph is running.
5. Use the context menu *Open UI* of the *Wiretap* node to open the wiretap.
6. The wiretap opens and you see the commit tokens.
7. Stop the graph and change the generator's `batchSize` and run the graph again.

3.8.18 Example for Vora Ingestion Using Record Object Data 2 in Python

The data generator generates a series of messages, each consisting of one or more record objects. Each message is directly passed to the ingestor, which stores the records in the Vora disk engine and emits the commit token associated with each message.

The table definition is not provided as an Avro schema but instead included in the message at attribute key `vora.record.definition`.

Prerequisites

- You need a running Vora instance.

Configure and Run the Graph

Follow the steps below to run the example from the SAP Data Intelligence Modeler UI:

1. In the left panel, select the [Graphs](#) tab and navigate to `com/sap/demo/vora/ingestion/rec_ingestion_example2_disk_python`.
2. Check the configuration of the [avroingestor](#) node: `dsn`
3. In the tool bar, select [Run](#) (play button).
4. The [Status](#) panel indicates if the graph is running.
5. Use the context menu [Open UI](#) of the [Wiretap](#) node to open the wiretap.
6. The wiretap opens and you see the commit tokens.
7. Stop the graph and change the generator's `batchSize` and run the graph again.

3.9 Scenario Templates

Here, you'll find information on how to set up and run each scenario template we've prepared.

For more information, see [Using Scenario Templates](#) in the [Modeling Guide for SAP Data Intelligence](#).

Related Information

[BW ODP to Vora \(Initial + Delta Load\) \[page 723\]](#)

[BW HANA View to File \[page 724\]](#)

[Data Extraction from SAP S/4HANA CDS View to a File Store \[page 725\]](#)

[Data Extraction from SAP S4/HANA CDS View to KAFKA \[page 726\]](#)

[Data Extraction using SLT to a File Store \[page 728\]](#)
[Data Extraction using SLT to KAFKA \[page 729\]](#)
[Ingest Files into HANA \[page 730\]](#)
[Ingest Files into Vora \[page 731\]](#)
[Initial Load + Delta Extraction from AnyDB \[page 733\]](#)
[Initial Load from Oracle \(parallel\) \[page 733\]](#)
[Javascript File Data Manipulation \[page 734\]](#)
[Load Files into HANA \[page 735\]](#)
[Load Files into Vora \[page 736\]](#)
[Node.js File Data Manipulation \[page 737\]](#)
[Python File Data Manipulation with Pandas \[page 738\]](#)
[R File Data Manipulation \[page 739\]](#)
[Simple Javascript File Data Manipulation \[page 740\]](#)
[Simple Javascript File-to-DB Data Manipulation \[page 741\]](#)
[Simple Javascript DB-to-File Data Manipulation \[page 742\]](#)
[Simple Node.js File Data Manipulation \[page 743\]](#)
[Simple Python File Data Manipulation \[page 744\]](#)
[Simple Python File-to-DB Data Manipulation \[page 745\]](#)
[Simple R File Data Manipulation \[page 746\]](#)
[Simple R File-to-DB Data Manipulation \[page 747\]](#)

3.9.1 BW ODP to Vora (Initial + Delta Load)

Fetches data from a BW DataStore object and writes it into a VORA target via ODP.

In this example, the graph is configured to execute an initial load and subsequent delta loads from `OEPM_ADSO1` BW DataStore object to a VORA target.

Prerequisites

- A valid HANA_DB connection.
- A valid BW connection containing reference to the ABAP connection. The ABAP connection is needed to access the BW system via RPC for the ODP Data Transfer.
- A valid VORA connection to be used as target.
- Check and load `OEPM_ADSO1` BW Advanced DatStore object with data if it's empty.

i Note

Any unconnected output port will result in a graph failure.

Configure and Run

1. Open the graph `com.sap.scenarioTemplates.BW.bwDataTransferODP` and click the Run button (▶).
2. Provide the following configuration substitutions:
 - `BW_CONNECTION`: BW Connection ID containing reference to a ABAP connection.
 - `VORA_CONNECTION`: VORA Connection ID as target.
 - `VORA_SCHEMA`: Schema name for VORA (for example, `default\admin`).
 - `TARGET_TABLE`: Target table name.
 - `SUBSCRIPTION`: Desired name for Subscription ID (for example, `bw_odp_to_vora`).
Modifying the Subscription ID will result in a full data (initial load) transfer for the next graph execution and delta data transfer for subsequent graph executions.
3. Click OK to run the graph to execute the initial load.
4. When there are delta changes to the source dataset, run the graph again to execute the delta load.
5. The "Status" panel indicates if the graph is running.
6. Launch the Vora Tools application and view the target table.

Additional Information

Please note that delta executions via ODP works only with (advanced) DataStores on the BW side and supports Vora streaming tables as targets.

3.9.2 BW HANA View to File

Fetches data from a BW DataStore object and writes it into a file (S3) target via HANA View.

In this example, the graph is configured to execute full load from `OEPM_ADSO1` BW DataStore object to a file (S3) target.

Prerequisites

- A valid `HANA_DB` connection.
- A valid BW connection containing reference to the `HANA_DB` connection via the 'HANA DB Connection ID' field of the BW connection.
- A valid S3 connection to be used as target.
- Check and load `OEPM_ADSO1` BW Advanced DataStore object with data if it's empty and make sure that an external HANA view exists for it.

i Note

Any unconnected output port will result in a graph failure.

Configure and Run

1. Open the graph `com.sap.scenarioTemplates.BW.bwDataTransferHANAView` and click the Run button (▶).
2. Provide the following configuration substitutions: `BW_CONNECTION` : BW Connection ID containing reference to a HANA_DB connection. `S3_CONNECTION` : S3 Connection ID as target. `TARGET_FOLDER` : Target folder name. `TARGET_FILE` : Target file name (for example, `TestFile.csv`).
3. Click OK to run the graph to execute the full load.
4. The "Status" panel indicates if the graph is running.
5. Navigate to S3 to view the target files.

3.9.3 Data Extraction from SAP S/4HANA CDS View to a File Store

In this example, the source is an SAP S/4HANA.

Not mandatory, but for convenience, this example may be extended to read the data from the written files and to feed into the Terminal operator.

Prerequisites

- You need a valid connection to SAP S/4HANA. Please create a valid and checked connection to an SAP S/4HANA environment or to an SAP NetWeaver/ABAP based system that supports the CDS technology. Please use the related description of the SAP DI - Connection Management to implement and test such a connection. In this example the communication is based on HTTPS, but you may use other protocols such as HTTP, RFC or (WebSocket) WSRFC
- You need a valid connection to a file store and the ability to create, read and write files to that file store location. Please use the related description of the SAP DI - Connection Management to implement and test such a connection.
- A couple of Operators need to be in place:
 - [ABAP CDS Reader \[page 33\]](#)
 - [ABAP Converter \[page 32\]](#)
 - [Go Operator \[page 333\]](#) (→ Determine Last Batch, Limit File Size, Check Last Batch)
 - [Read File \(Deprecated\) \[page 580\]](#)
 - [Write File \(Deprecated\) \[page 603\]](#)
 - [Graph Terminator \[page 525\]](#)
- The *CDS View* must allow data extraction via the annotation: `@Analytics:{ dataExtraction: enabled: true ... for delta extraction. The delta.changeDataCapture part of the annotation needs to be maintained correctly.`

Configure and Run the Graph

1. In this example the ABAP CDS Reader is configured to execute an initial load. The CDS Reader supports 3 ways to load data: a) the straight forward Initial Load, which copies the data from source to target but without any subsequent replication. b) the Replication, which includes the Initial Load and - based on a delta handling - the subsequent replication of changed data. c) the Delta Processing, which implies that no Initial Load takes place and changed data only will get processed.
2. The ABAP Converter feeds via a Message Converter into a Go operator `Determine Last Batch`, which checks whether the SLT-based replication got stopped and is then launching a soft-kill (used at `Check Last Batch`)
3. The data keep flowing into a Go operator `Limit File Size`, which controls the file size as a parameter and the file counter.
4. The Write File Operator uses the configured 'append' mode to fill into ``/tmp/file_<header:ABAPfilenumber>.csv`` and its configured connection.
5. The Go operator `Check Last Batch` terminates the graph in case of the end of the replication; without the two operators `* Last Batch`, it can happen that the graph gets terminated before the last write gets completed.
6. (optional) To simply visualize the file, the just used and created file may be read again and sent to Terminal.
7. Use to execute this Graph.

i Note

In case of an Initial Load only, the graph will terminate, in case of Delta Mode or Replication (=Initial Load plus Delta) the graph will not stop after the initial load.

Data Integration and Data Security

Please make yourself familiar with the related Integration Guides and Security considerations such as [User Guide for ABAP Integration in SAP Data Intelligence](#), the [Administration Guide for SAP Data Intelligence](#), and SAP Note [2831756](#) .

3.9.4 Data Extraction from SAP S4/HANA CDS View to KAFKA

Extracts data from an ABAP CDS View to the KAFKA and creates related messages.

Not mandatory, but for convenience, this example re-reads the data from the KAFKA feed and redirects it to the Terminal operator.

Prerequisites

- You need a valid connection to SAP S/4HANA. Please create a valid and checked connection to an SAP S/4HANA environment or to an SAP NetWeaver/ABAP based system that supports the CDS technology. Please use the related description of the SAP DI - Connection Management to implement and test such a connection. In this example the communication is based on HTTPS, but you may use other protocols such as HTTP, RFC or (WebSocket) WSRFC
- You need a checked KAFKA connection and the ability to create, read and write messages to KAFKA. Please use the related description of the SAP DI - Connection Management to implement and test such a connection.
- A couple of Operators need to be in place:
 - [ABAP CDS Reader \[page 33\]](#)
 - [ABAP Converter \[page 32\]](#)
 - [Go Operator \[page 333\]](#)
 - [Graph Terminator \[page 525\]](#)
 - [Kafka Producer \[page 96\]](#)
 - [Kafka Consumer 2 \[page 90\]](#)
 - [Terminal \[page 539\]](#)
 - [Wiretap \[page 541\]](#)
- The CDS View must allow data extraction via the annotation: `@Analytics:{ dataExtraction: enabled: true ...}` for delta extraction. The `delta.changeDataCapture` part of the annotation needs to be maintained correctly.

Configure and Run the Graph

1. In this example the ABAP CDS Reader is configured to execute an initial load.
The CDS Reader supports 3 ways to load data:
 - The straight forward Initial Load, which copies the data from source to target but without any subsequent replication;
 - The Replication, which includes the Initial Load and - based on a delta handling - the subsequent replication of changed data;
 - The Delta Processing, which implies that no Initial Load takes place and changed data only will get processed.
2. The ABAP Converter feeds via a Message Converter into a Go Operator `Determine Last Batch`, which checks whether the SLT-based replication got stopped and is then launching a soft-kill (used at `Check Last Batch`)
The data keep flowing into KAFKA Producer using topic `test_topic`.
The KAFKA Producer converts the records to messages within the configured parameters to KAFKA Topic `test_topic`.
The Go Operator `Check Last Batch` terminates the graph in case of the end of the replication; without the two operators `* Last Batch`, it can happen that the graph gets terminated before the last write got completed.
Optional: simply to visualize the flow, the just used and created messages can be consumed by a KAFKA consumer and sent to Terminal/Wiretap.
Use to execute this Graph.

Data Integration and Data Security

Please make yourself familiar with the related Integration Guides and Security considerations such as [User Guide for ABAP Integration in SAP Data Intelligence](#), the [Administration Guide for SAP Data Intelligence](#), and SAP Note [2831756](#).

3.9.5 Data Extraction using SLT to a File Store

Extracts data from an ABAP table using SLT to a file store and creates related files in the file store.

In this example, the source is an SAP S/4HANA. Not mandatory, but for convenience, this example may be extended to read the data from the written files and to feed into the Terminal operator.

Prerequisites

- You need a valid connection to the SAP S/4HANA source. Please create a valid and checked connection to an SAP S/4HANA environment or to an SAP NetWeaver/ABAP based system that supports the SLT technology. Please use the related description of the SAP DI - Connection Management to implement and test such a connection. In this example the communication is based on HTTPS, but you may use other protocols such as HTTP, RFC or (WebSocket) WSRFC.
- You need a valid connection to a file store and the ability to create, read and write files to that file store location. Please use the related description of the SAP DI - Connection Management to implement and test such a connection. In this example we used S3 as a target file store.
- A couple of operators need to be in place:
 - [SLT Connector \[page 38\]](#)
 - [ABAP Converter \[page 32\]](#)
 - [Go Operator \[page 333\]](#) (→ Determine Last Batch, Limit File Size, Check Last Batch)
 - [Read File \(Deprecated\) \[page 580\]](#)
 - [Write File \(Deprecated\) \[page 603\]](#)
 - [Graph Terminator \[page 525\]](#)

Configure and Run

1. In this example the SLT Connector replicates the an ABAP table to S3.
2. The ABAP Converter feeds via a Message Converter into a Go Operator `Determine Last Batch`, which checks whether the SLT-based replication got stopped and is then launching a soft-kill (used at `Check Last Batch`)
3. The data keep flowing into a Go operator `Limit File Size`, which controls the file size as a parameter and the file counter.

4. The Write File operator is used to create the files until end-of-initial-load. The Write File operator uses the configured 'append' mode to fill into `/tmp/file_${ABAPfilenumber}.csv` as of Service S3 and its configured connection.
5. The Go operator `Check Last Batch` terminates the graph in case of the end of the replication; without the two operators `* Last Batch`, it can happen that the graph gets terminated before the last write got completed.
6. (optional) To simply visualize the file, the just used and created file on S3 may be read again and sent to Terminal.
7. Use to execute this Graph.

i Note

In case of an Initial Load only, the graph will terminate, in case of Delta Mode or Replication (=Initial Load plus Delta) the graph will not stop after the initial load.

Data Integration and Data Security

Please make yourself familiar with the related Integration Guides and Security considerations such as [User Guide for ABAP Integration in SAP Data Intelligence](#), the [Administration Guide for SAP Data Intelligence](#), and SAP Note [2831756](#).

3.9.6 Data Extraction using SLT to KAFKA

Extracts data from an ABAP table using SLT to the KAFKA and creates related messages.

Not mandatory, but for convenience, this example re-reads the data from the KAFKA feed and redirects it to the Terminal operator.

Prerequisites

- You need a valid connection to the SAP LT Replication Server / SLT. Please create a valid and checked connection to an the SAP LT Replication Server / SLT system that supports the necessary SLT scenarios. Please use the related description of the SAP DI - Connection Management to implement and test such a connection. In this example the communication is based on HTTPS, but you may use other protocols such as HTTP, RFC or (WebSocket) WSRFC.
- You need a checked KAFKA connection and the ability to create, read and write messages to KAFKA. Please use the related description of the SAP DI - Connection Management to implement and test such a connection. In this example we used KAFKA as the consumer of the created messages.
- A couple of Operators need to be in place: (ABAP) SLT Connector, ABAP Converter, Go Operator (→ String Operator, Determine Last Batch, Limit File Size, Check Last Batch), Python (→ splitting a portion of records to single records for messaging), Graph Terminator, KAFKA Producer and Consumer, Terminal, Wiretap.

Configure and Run

1. In this example the SLT Connector replicates the specified ABAP table to KAFKA. The CDS Reader supports 3 ways to load data: a) the straight forward Initial Load, which copies the data from source to target but without any subsequent replication. b) the Replication, which includes the Initial Load and - based on a delta handling - the subsequent replication of changed data. c) the Delta Processing, which implies that no Initial Load takes place and changed data only will get processed.
2. The ABAP Converter feeds via a Message Converter into a Go operator `Determine Last Batch`, which checks whether the SLT-based replication got stopped and is then launching a soft-kill (used at `Check Last Batch`)
3. The data keep flowing into KAFKA Producer using topic `test_topic_SLT`.
4. The KAFKA Producer converts the records to messages within the configured parameters to KAFKA Topic `test_topic_SLT`.
5. The Go operator `Check Last Batch` terminates the graph in case of the end of the replication; without the two operators `* Last Batch`, it can happen that the graph gets terminate before the last write got completed.
6. (optional) To simply visualize the flow, the just used and created messages can be consumed by a KAFKA consumer and sent to Terminal/Wiretap.
7. Use to execute this Graph.

i Note

To improve performance: implementation of a Python script that splits the portion based table extraction into individual line items to support a single-message-based-on-single-data-record messaging.

Data Integration and Data Security

Please make yourself familiar with the related Integration Guides and Security considerations such as [User Guide for ABAP Integration in SAP Data Intelligence](#), the [Administration Guide for SAP Data Intelligence](#), and SAP Note [2831756](#).

3.9.7 Ingest Files into HANA

Continuously ingests product data from CSV files into a HANA table. It lists all `product*.csv` files, reads their contents and loads into a HANA table.

After all files are listed once, the Read File operator monitors the directory for any new or modified files. The Message Key Store guarantees each file is processed at least once, even through multiple graph runs.

The graph flows as follows:

- **Monitor Files** monitors the configured path for new and modified files matching the ``products.*\*.csv`` regular expression. When the graph starts, all files are listed once, because the "Already exists" event is also enabled.

- The **Message Key Store operator** checks if each incoming file in the input port has already been acknowledged. It outputs to acked if so, and to output otherwise. When it receives a file in the ack port, it is marked as acknowledged, avoiding reprocessing it in future runs. To process all the files again, you may set its configuration Previous Content Initialization to Reset.
- **To File** converts the incoming data of type `message` to the type `message.file`.
- The **Read File operator** receives any not yet processed file, reading its contents and sending to the SAP HANA Client.
- **Clean CSV Column** removes any spaces and special characters from the first column of the CSV.
- **SAP HANA Client** takes a headerless CSV content and loads it into the configured Table name (currently using substitution parameters). It also creates the table if it doesn't exist when the graph starts, according to the Table initialization configurations. To avoid that you may change the insert mode of the HANA Client to "UPSERT" and for that create primary keys on the HANA table.
- **Extract Path** takes the path attribute value from the input and sends as an output. It is used only for visualization purposes along with the Wiretap operator. It doesn't have any productive effect in the graph.
- **Wiretap** shows any file path that has already been listed and processed (or ignored). It doesn't have any productive effect in the graph.

Prerequisites

- A running SAP HANA database with a matching registered connection.
- A running Vora database with a matching registered connection.

Configure and Run

1. Open the graph `com.sap.scenarioTemplates.datalakeToDatabase.ingestToHana` and click the run Run button (▶).
2. Fill the HANA_CONNECTION and VORA_CONNECTION values with the respective connection IDs for HANA and Vora.
3. Click "OK".
4. The "Status" panel indicates if the graph is running.

Optional Step

Connect to SAP HANA and browse the table `DEMO_STDL2DBHANAINGEST_PRODUCTS` under the schema you configured to see the produced data.

3.9.8 Ingest Files into Vora

Continuously ingests product data from CSV files into a Vora table. It lists all `product*.csv` files, reads their contents and loads into a Vora streaming table.

After all files are listed once, the Read File operator monitors the directory for any new or modified files. The Message Key Store guarantees each file is processed at least once, even through multiple graph runs. As the

Vora Avro Ingestor uses a table with a primary key and loads data via upsert, the graph also avoids any duplicates.

The graph flows as follows:

- **Monitor Files** monitors the configured path for new and modified files matching the ``products.*\.csv`` regular expression. When the graph starts, all files are listed once, because the "Already exists" event is also enabled.
- The **Message Key Store operator** checks if each incoming file in the input port has already been acknowledged. It outputs to acked if so, and to output otherwise. When it receives a file in the ack port, it is marked as acknowledged, avoiding reprocessing it in future runs. In order to process all the files again, you may set its configuration Previous Content Initialization to Reset.
- **To File** converts the incoming data of type ``message`` to the type ``message.file``.
- The **Read File operator** receives any not yet processed file, reading its contents and sending to the Vora Avro Ingestor.
- **Clean CSV Column** removes any spaces and special characters from the first column of the CSV.
- **Vora Avro Ingestor operator** receives any not yet processed file, loading it to a Vora streaming table. It uses ingestionMode as UPSERT, avoiding duplicate data.
- **Join Attributes** rebuilds the message with the proper attributes used in the Vora Avro Ingestor input. It takes the attributes from the attributes port and the body from the body port.
- **Extract Path** takes the path attribute value from the input and sends as an output. It is used only for visualization purposes along with the Wiretap operator. It doesn't have any productive effect in the graph.
- **Wiretap** shows any file path that has already been listed and processed (or ignored). It doesn't have any productive effect in the graph.

Prerequisites

- A running Vora database with a matching registered connection.

Configure and Run

1. Open the graph `com.sap.scenarioTemplates.datalakeToDatabase.ingestToVora` and click the run Run button (▶).
2. Fill the `VORA_CONNECTION` value with the Vora connection ID.
3. Click "OK".
4. The "Status" panel indicates if the graph is running.

Optional Step: Check the Data on HANA

Use the Vora Tools app and browse the table `DEMO_STDL2DBVORAINGEST_PRODUCTS` under the schema you configured to see the produced data.

3.9.9 Initial Load + Delta Extraction from AnyDB

Demonstrates how to use CDC Graph Generator Operator for replication of relational databases.

The Table Replicator operator performs replication through trigger mechanism. At the first execution, the necessary triggers and procedures are created in the source system and an initial load is performed. For consecutive executions, only delta extraction is performed.

For more information, see the Table Replicator operator documentation.

Prerequisite

Permission to create triggers, tables, and views in the source.

3.9.10 Initial Load from Oracle (parallel)

Demonstrates how to read contents from a database table and load it into a file.

In this graph, an Oracle Table Consumer is used to read an Oracle table, and then its contents are converted into CSV and written into a file by the Flowagent File Producer operator.

This graph also shows how to read from the table in parallel through the logical partitioning concept. Click [here](#) to know more about logical partitioning specification. To read the data sequentially, set the configuration substitution NUM_PARTITIONS to 1.

Prerequisites

Please see Oracle driver configuration and the connection documentation present in Connection Management.

Components

Reader definition

- `oracletableconsumer1`: operator that reads from the Oracle table.
Use the substitution variables to define the Oracle connection and desired source table.

Loader definition

- `flowagentfileproducer1`: operator that reads and writes the data into a file. Local and cloud storages are both supported.

Wait to terminate graph

- messageoperator1: operator that waits for all partitions to finish before sending a message to graph terminator.

Configuration Substitutions

- ORACLE_SOURCE_CONNECTION: ID of an Oracle connection
- ORACLE_SOURCE_TABLE: Name of the table read, in the format '.'
- OUT_FILE_NAME: Name of the file where the contents are written
- NUM_PARTITIONS: Number of generated partitions

3.9.11 Javascript File Data Manipulation

Exemplifies file data manipulation using a Javascript custom operator and writing the manipulated data back to another file.

Prerequisites

- A file containing data.
In the preconfigured example a JSON format file `test_input.json` is assumed to sit in the `/files/exampleData/scenarioTemplates` folder. It should contain an array of objects with a string type `id` property and some numerical data properties.

Configure and Run

1. Choose Run As from the Run drop-down menu and select the "default" configuration substitutions to run the graph with a preconfigured example configuration.
2. To modify the data manipulation performed by the custom operator go to the Operators tab and find the Process Data with Javascript operator, then select Edit from the operator's context menu. Please note that the operator supports ECMAScript 2009 (ES5) specifications.
3. Enter the path of the file containing the input data in the Path configuration parameter of the Read Input File operator.
The example configuration includes `/exampleData/scenarioTemplates/test_input.json` for this parameter (the operator will prepend the `/files/`, matching the folder found in System Management).
4. Enter the file path for the data manipulation result in the Path configuration parameter of the Write Results File operator.
In the preconfigured example the results will be written to a file `test_custom_js.json` in the `/files/exampleData/scenarioTemplates` folder.
5. Run the graph by clicking the Run button.

3.9.12 Load Files into HANA

Loads product data from CSV files into a HANA table. It lists all products*.csv files, reads their contents and loads into a HANA table, terminating the graph when all listed files are loaded.

The Message Key Store guarantees each file is processed at least once, even through multiple subsequent graph runs, and files that have been successfully processed and acknowledged before are not processed again.

The graph flows as follows:

- **List Files** lists all files matching the ``products.*\*.csv`` regular expression, based on the configured path. The operation takes place only once. It fails if no files are listed.
- The **Message Key Store operator** checks if each incoming file in the input port has already been acknowledged. It outputs to `acked` if so, and to output otherwise. When it receives a file in the `ack` port, it is marked as acknowledged, avoiding reprocessing it in future runs. To process all the files again, you may set its configuration Previous Content Initialization to `Reset`.
- **To File** converts the incoming data of type ``message`` to the type ``message.file``.
- The **Read File operator** receives any not yet processed file, reading its contents and sending to the SAP HANA Client.
- **From File** converts the incoming data of type ``message.file`` to the type ``message``.
- **SAP HANA Client** takes a headerless CSV content and loads it into the configured Table name (currently using substitution parameters). It also creates the table if it doesn't exist when the graph starts, according to the Table initialization configurations. Due to the at least once guarantees, it may happen that files that have not been acknowledged in the Message Key Store before are sent twice resulting in duplicate records. To avoid that you may change the insert mode of the HANA Client to "UPSERT" and for that create primary keys on the HANA table.
- **Stop Handler** checks for the `message.lastBatch` header, sending a signal to the Graph Terminator.
- **Graph Terminator** terminates the graph as completed when an input is received.

Prerequisites

- A running SAP HANA database with a matching registered connection.
- A running Vora database with a matching registered connection.

Configure and Run

1. Open the graph `com.sap.scenarioTemplates.dataLakeToDatabase.loadToHana` and click the run Run button (▶).
2. Fill the `HANA_CONNECTION` and `VORA_CONNECTION` values with the respective connection IDs for HANA and Vora.
3. Click "OK".
4. The "Status" panel indicates if the graph is running.

Optional Step

Connect to SAP HANA and browse the table DEMO_STDL2DBHANAINGEST_PRODUCTS under the schema you configured to see the produced data.

3.9.13 Load Files into Vora

Loads product data from CSV files into a Vora table. It lists all product*.csv files from S3 and loads them into a Vora table, terminating the graph when all listed files are loaded.

The Message Key Store guarantees each file is processed at least once, even through multiple subsequent graph runs, and files that have been successfully processed and acknowledged before are not processed again.

You will need an S3 connection with the products CSV files in the directory /exampleData/scenarioTemplates/products/. You can find these CSV files in the directory /files/exampleData/scenarioTemplates/products/ in the System Management Files tab, and use the Metadata Explorer to upload them to your S3 connection. If you want to use another storage service instead of S3, follow the steps from the section Changing Storage Service.

The graph flows as follows:

- **List Files** lists all files matching the ``products.*\*.csv`` regular expression, based on the configured path. The operation takes place only once. It fails if no files are listed.
- **From File** converts the incoming extracts the path from the incoming ``message.file`` to the body of the message being output.
- The **Message Key Store operator** checks if each incoming file in the input port has already been acknowledged. It outputs to `acked` if so, and to output otherwise. When it receives a file in the `ack` port, it is marked as acknowledged, avoiding reprocessing it in future runs. To process all the files again, you may set its configuration Previous Content Initialization to `Reset`.
- **1:2 Multiplexer** sends any received input to both Vora Loader and the Join Attributes operator. This is needed as the Vora Loader doesn't take message inputs, discarding its attributes.
- The **Vora Loader operator** receives any not yet processed file, loading it to the Vora table. It also creates the table if it doesn't exist when the graph, according to the Init Statements configurations.
- **Join Attributes** rebuilds the message with the proper attributes used in the Vora Loader input. It takes the attributes from the attributes port and the body from the body port.
- **Stop Handler** checks for the `message.lastBatch` header, sending a signal to the Graph Terminator.
- **Graph Terminator** terminates the graph as completed when an input is received.

Prerequisites

- A running Vora database with a matching registered connection.
- A registered S3 connection.

Configure and Run

1. Open the graph `com.sap.scenarioTemplates.datalakeToDatabase.loadToVora` and click the run Run button (▶).
2. Fill the `VORA_CONNECTION` value with the Vora connection ID.
3. Fill the `S3_CONNECTION` value with the S3 connection ID.
4. Click "OK".
5. The "Status" panel indicates if the graph is running.

Optional Step: Check the Data on Vora

Use the Vora Tools app and browse the table `DEMO_STDL2DBVORALOAD_PRODUCTS` under the schema you configured to see the produced data.

Changing Storage Service

In the Read File (List Files) operator:

1. Change the Service configuration to your desired service.
2. Update the Connection configuration with your connection value.

In the Vora Loader operator:

1. Change the Storage service configuration to your desired service.
2. Change the new connection configuration that is shown right below the Storage service with your connection value.

3.9.14 Node.js File Data Manipulation

This graph exemplifies file data manipulation using Node.js and writing the manipulated data back to another file.

Prerequisites

For this graph to run you will need:

- a file containing data.
In the preconfigured example a CSV format file ``test_large.csv`` is assumed to sit in the ``/files/exampleData/scenarioTemplates`` folder.

Configure and Run the Graph

1. To modify the data manipulation performed by the custom operator go to the [Operators](#) tab and find the [Process Data with Node.js](#) operator, then select [Edit](#) from the operator's context menu. Note that the operator supports ECMAScript 2015 (ES6) specifications.
2. Enter the path of the file containing the input data in the [Path](#) configuration parameter of the Read Input File operator.
The example configuration includes ``/exampleData/scenarioTemplates/test_input.json`` for this parameter (the operator will prepend the ``/files/``, matching the folder found in System Management).
3. Enter the file path for the data manipulation result in the [Path](#) configuration parameter of the Write Results File operator.
In the preconfigured example the results will be written to a file ``test_custom_node.csv`` in the ``/files/exampleData/scenarioTemplates`` folder.
4. Run the graph by clicking the [Run](#) button.

3.9.15 Python File Data Manipulation with Pandas

Exemplifies file data manipulation using Python with the Pandas library and writing the manipulated data back to another file.

Prerequisites

For this graph, you will need:

- A file containing data.
In the preconfigured example a CSV format file ``test_large.csv`` is assumed to sit in the ``/files/exampleData/scenarioTemplates`` folder.
The file should be:
 - in CSV format including a header row;
 - structured into four columns `t` (representing time), `v` (representing velocity in m/s), `s` (representing distance in m travelled between two data points), and `d` (representing total distance in m travelled since start of the data series);
 - sampled at a higher rate than 5 mins.

Configure and Run

1. Choose Run As from the Run drop-down menu and select the "default" configuration substitutions to run the graph with a preconfigured example configuration.
2. To modify the data manipulation performed by the custom operator go to the Operators tab and find the Process Data with Python operator, then select Edit from the operator's context menu.

3. Enter the path of the file containing the input data in the Path configuration parameter of the Read Input File operator.
The example configuration includes ``/exampleData/scenarioTemplates/test_large.csv`` for this parameter (the operator will prepend the ``/files/`` prefix, matching the folder found in System Management).
4. Enter the file path for the data manipulation result in the Path configuration parameter of the Write Results File operator.
In the preconfigured example the results will be written to a file `test_custom_py.csv` in the `files/exampleData/scenarioTemplates` folder.
5. Run the graph by clicking the Run button.

3.9.16 R File Data Manipulation

This graph exemplifies file data manipulation with an R custom operator and writing the manipulated data back to another file.

Prerequisites

For this graph to run you will need:

- a file containing data.
In the preconfigured example a CSV format file `test_products.csv` is assumed to sit in the ``/files/exampleData/scenarioTemplates`` folder. It should have a header row and contain at least the following columns "PRODUCTID", "PRODUCTNAME" and "CATEGORY".

Configure and Run the Graph

1. Choose [Run As](#) from the [Run](#) drop-down menu and select the "default" configuration substitutions to run the graph with a preconfigured example configuration.
2. To modify the data manipulation performed by the custom operator go to the [Operators](#) tab and find the [Process Data with R](#) operator, then select [Edit](#) from the operator's context menu.
3. Enter the path of the file containing the input data in the [Path](#) configuration parameter of the Read Input File operator.
The example configuration includes ``/exampleData/scenarioTemplates/test_large.csv`` for this parameter (the operator will prepend the ``/files/``, matching the folder found in System Management).
4. Enter the file path for the data manipulation result in the [Path](#) configuration parameter of the Write Results File operator.
In the preconfigured example the results will be written to a file `test_custom_r.csv` in the ``/files/exampleData/scenarioTemplates`` folder.
5. Run the graph by clicking the [Run](#) button.

3.9.17 Simple Javascript File Data Manipulation

Exemplifies file data manipulation using plain Javascript (without any libraries) and writing the manipulated data back to another file.

Prerequisites

For this graph, you will need:

- a file containing data.
In the preconfigured example a JSON format file `test_input.json` is assumed to sit in the `/files/exampleData/scenarioTemplates` folder. It should contain an array of objects with a string type `id` property and some numerical data properties.

Configure and Run

1. Choose Run As from the Run drop-down menu and select the "default" configuration substitutions to run the graph with a preconfigured example configuration.
2. Click on the Script button of the Process Data operator and enter the Javascript code that performs the data manipulation. Consult the operator documentation for details. Please note that the operator supports ECMAScript 2009 (ES5) specifications.
The preconfigured example script will insert an additional row of data between each data row in the `test_input.json` file that contains interpolated values for each numerical column.
3. Enter the path of the file containing the input data in the Path configuration parameter of the Read Input File operator.
The example configuration includes `/exampleData/scenarioTemplates/test_input.json` for this parameter (the operator will prepend the `/files/` prefix, matching the folder found in System Management).
4. Enter the file path for the data manipulation result in the Path configuration parameter of the Write Results File operator.
In the preconfigured example the results will be written to a file `test_result.js.json` in the `files/exampleData/scenarioTemplates` folder.
5. Run the graph by clicking the Run button.

3.9.18 Simple Javascript File-to-DB Data Manipulation

Exemplifies file data manipulation using plain Javascript (without any libraries) and writing the manipulated data to an SAP HANA database table.

Prerequisites

For this graph, you will need:

- a file containing data.
In the preconfigured example a JSON format file `test_input.json` is assumed to sit in the `files/exampleData/scenarioTemplates` folder. It should contain an array of objects with a string type `id` property and some numerical data properties.
- a connection with ID `CUSTDATAPROC_HANA` to an SAP HANA database.

Configure and Run

1. Click on the Script button of the Process Data operator and enter the Javascript code that performs the data manipulation. Consult the operator documentation for details. Please note that the operator supports ECMAScript 2009 (ES5) specifications.
The preconfigured example script will insert an additional row of data between each data row in the `test_input.json` file that contains interpolated values for each numerical column.
2. Enter the path of the file containing the input data in the Path configuration parameter of the Read Input File operator.
In the preconfigured example the results will be written to a file `test_db_js.json` in the `/files/exampleData/scenarioTemplates` folder.
3. Configure the database connection and the target table for the data manipulation result in the Connection, Table name, and Table columns configuration parameters of the Write to HANA DB operator.
In the preconfigured example the results will be written to the `CUSTDATAPROC_RESULT` table in the user schema pertaining to the user of the configured SAP HANA connection. The table will be created when executing the graph.
4. Run the graph by clicking the Run button.

3.9.19 Simple Javascript DB-to-File Data Manipulation

Exemplifies the manipulation of data from an SAP HANA database table using plain Javascript (without any libraries) and writing the manipulated data to a file.

Prerequisites

For this graph, you will need:

- a connection with ID CUSTDATAPROC_HANA to an SAP HANA database.
- a database table in SAP HANA containing data.
In the preconfigured example a table CUSTDATAPROC_RESULT is assumed to exist in the user schema pertaining to the user of the CUSTDATAPROC_HANA SAP HANA database connection. It should contain a string type ID column and some numerical data columns. A table like this is created by running the `com.sap.scenarioTemplates.customDataProcessing.simpleF2DBJavascript` graph.

Configure and Run

1. Choose Run As from the Run drop-down menu and select the "default" configuration substitutions to run the graph with a preconfigured example configuration.
2. Enter the SQL statement that reads the input data from the SAP HANA database table in the Content configuration parameter of the SQL Statement operator.
The example configuration contains `select * from CUSTDATAPROC_RESULT order by ID` for this parameter.
3. Choose the connection for the input data in the Connection configuration parameter of the Get Data from DB operator.
In the example configuration this parameter is set to CUSTDATAPROC_HANA.
4. Click on the Script button of the Process Data operator and enter the Javascript code that performs the data manipulation. Consult the operator documentation for details. Please note that the operator supports ECMAScript 2009 (ES5) specifications.
The preconfigured example script will insert an additional row of data between each data row in the CUSTDATAPROC_RESULT table that contains interpolated values for each numerical column.
5. Enter the file path for the data manipulation result in the Path configuration parameter of the Write Results File operator.
In the preconfigured example the results will be written to a file ``test_db_js.json`` in the ``/files/exampleData/scenarioTemplates`` folder.
Run the graph by clicking the Run button.

3.9.20 Simple Node.js File Data Manipulation

Exemplifies file data manipulation using Node.js (without any libraries) and writing the manipulated data back to another file.

Prerequisites

For this graph, you will need:

- a file containing data.
In the preconfigured example a JSON format file `test_input.json` is assumed to sit in the `/files/exampleData/scenarioTemplates` folder. It should contain an array of objects with a string type `id` property and some numerical data properties.

Configure and Run

1. Choose Run As from the Run drop-down menu and select the "default" configuration substitutions to run the graph with a preconfigured example configuration.
2. Click on the Script button of the Process Data operator and enter the Node/Javascript code that performs the data manipulation. Consult the operator documentation for details.
The preconfigured example script will insert an additional row of data between each data row in the `test_input.json` file that contains interpolated values for each numerical column.
3. Enter the path of the file containing the input data in the Path configuration parameter of the Read Input File operator.
The example configuration includes `/exampleData/scenarioTemplates/test_input.json` for this parameter (the operator will prepend the `/files/` prefix, matching the folder found in System Management).
4. Enter the file path for the data manipulation result in the Path configuration parameter of the Write Results File operator.
In the preconfigured example the results will be written to a file `test_result_node.json` in the `/files/exampleData/scenarioTemplates` folder.
5. Run the graph by clicking the Run button.

3.9.21 Simple Python File Data Manipulation

Exemplifies file data manipulation using plain Python with only built-in modules and writing the manipulated data back to another file.

Prerequisites

For this graph, you will need:

- a file containing data.
In the preconfigured example a CSV format file `test_input.csv` is assumed to sit in the `/files/exampleData/scenarioTemplates` folder. It should have a header row including an `id` column, use `,` (comma) as column separator and contain numerical data (except for the `id` column, which may be string type).

Configure and Run

1. Choose Run As from the Run drop-down menu and select the "default" configuration substitutions to run the graph with a preconfigured example configuration.
2. Click on the Script button of the Process Data operator and enter the Python code that performs the data manipulation. Consult the operator documentation for details.
The preconfigured example script will insert an additional row of data between each data row in the `test_input.csv` file that contains interpolated values for each numerical column.
3. Enter the path of the file containing the input data in the Path configuration parameter of the Read Input File operator.
The example configuration includes `/exampleData/scenarioTemplates/test_input.csv` for this parameter (it will prepend the `/files` prefix, matching the folder found in System Management).
4. Enter the file path for the data manipulation result in the Path configuration parameter of the Write Results File operator.
In the preconfigured example the results will be written to a file `test_result_py.csv` in the `/files/exampleData/scenarioTemplates` folder.
5. Run the graph by clicking the Run button.

3.9.22 Simple Python File-to-DB Data Manipulation

Exemplifies file data manipulation using plain Python with only built-in modules and writing the manipulated data to an SAP Data Hub built-in Vora database table.

Prerequisites

For this graph, you will need:

- a file containing data.
In the preconfigured example a CSV format file `test_input.csv` is assumed to sit in the `files/exampleData/scenarioTemplates` folder. It should have a header row including an id column, use "," (comma) as column separator and contain numerical data (except for the id column, which may be string type).
- a connection with ID `CUSTDATAPROC_VORA` to the SAP Data Hub built-in Vora database.

Configure and Run

1. Click on the Script button of the Process Data operator and enter the Javascript code that performs the data manipulation. Consult the operator documentation for details. Please note that the operator supports ECMAScript 2009 (ES5) specifications.
2. Click on the Script button of the Process Data operator and enter the Python code that performs the data manipulation. Consult the operator documentation for details.
The preconfigured example script will insert an additional row of data between each data row in the `test_input.csv` file that contains interpolated values for each numerical column.
3. Enter the path of the file containing the input data in the Path configuration parameter of the Read Input File operator.
The example configuration includes ``/exampleData/scenarioTemplates/test_input.csv`` for this parameter (the operator will prepend the ``/files/`` prefix, matching the folder found in System Management).
4. Configure the database connection for the data manipulation result in the `connectionType` and `Connection` configuration parameters of the Write to Vora DB operator.
In the preconfigured example the results will be written to the `CUSTDATAPROC_RESULT` table in the current user's schema. The table will be created when executing the graph.
5. Run the graph by clicking the Run button.

3.9.23 Simple R File Data Manipulation

Exemplifies file data manipulation using plain R (using only built-in libraries) and writing the manipulated data back to another file.

Prerequisites

For this graph, you will need:

- a file containing data.
In the preconfigured example a CSV format file ``test_products.csv`` is assumed to sit in the ``/files/exampleData/scenarioTemplates`` folder. It should have a header row and contain at least the following columns "PRODUCTID", "PRODUCTNAME" and "CATEGORY".

Configure and Run

1. Choose Run As from the Run drop-down menu and select the "default" configuration substitutions to run the graph with a preconfigured example configuration.
2. Click on the Script button of the Process Data operator and enter the R code that performs the data manipulation. Consult the operator documentation for details.
The preconfigured example script will filter for Notebooks on the column "CATEGORY" and return the columns "PRODUCTID", "PRODUCTNAME" and "CATEGORY" in the projection.
3. Enter the path of the file containing the input data in the Path configuration parameter of the Read Input File operator.
The example configuration includes ``/exampleData/scenarioTemplates/test_products.csv`` for this parameter (the operator will prepend the ``/files/`` prefix, matching the folder found in System Management).
4. Enter the file path for the data manipulation result in the Path configuration parameter of the Write Results File operator.
In the preconfigured example the results will be written to a file ``test_result_products_transformed.csv`` in the ``/files/exampleData/scenarioTemplates`` folder.
5. Run the graph by clicking the Run button.

3.9.24 Simple R File-to-DB Data Manipulation

Exemplifies file data manipulation using plain R (using only built-in libraries) and writing the manipulated data to an SAP HANA database table.

Prerequisites

For this graph, you will need:

- a file containing data.
In the preconfigured example a CSV format file `test_products.csv` is assumed to sit in the `files/exampleData/scenarioTemplates` folder. It should have a header row and contain at least the following columns "PRODUCTID", "PRODUCTNAME" and "CATEGORY".
- a connection with ID CUSTDATAPROC_HANA to an SAP HANA database.

Configure and Run

1. Choose Run As from the Run drop-down menu and select the "default" configuration substitutions to run the graph with a preconfigured example configuration.
2. Click on the Script button of the Process Data operator and enter the Javascript code that performs the data manipulation. Consult the operator documentation for details. Please note that the operator supports ECMAScript 2009 (ES5) specifications.
3. Click on the Script button of the Process Data operator and enter the R code that performs the data manipulation. Consult the operator documentation for details.
The preconfigured example script will filter for Notebooks on the column "CATEGORY" and return the columns "PRODUCTID", "PRODUCTNAME" and "CATEGORY" in the projection.
4. Enter the path of the file containing the input data in the Path configuration parameter of the Read Input File operator.
The example configuration includes `/exampleData/scenarioTemplates/test_products.csv` for this parameter (it will prepend the `/files` prefix, matching the folder found in System Management).
5. Configure the database connection and the target table for the data manipulation result in the Connection, Table name, and Table columns configuration parameters of the Write to HANA DB operator.
In the preconfigured example the results will be written to the PRODUCTS_RESULT table in the user schema pertaining to the user of the configured SAP HANA connection. The table will be created when executing the graph.
6. Run the graph by clicking the Run button.

3.10 Streaming Analytics

Related Information

[Using Adapters in Streaming Analytics Project CCL \[page 748\]](#)

[Streaming Analytics Freezer Monitor Demo \[page 749\]](#)

[Connecting to Kafka in Streaming Analytics Project CCL \[page 750\]](#)

[Sending Datatypes to Streaming Analytics Projects \[page 751\]](#)

3.10.1 Using Adapters in Streaming Analytics Project CCL

This example graph shows how you can send and receive data to and from a streaming analytics operator (project) using adapters. You add an adapter to the operator using Continuous Computation Language (CCL). With an adapter, all data is sent and received without using any input or output ports. However, you can always add input or output ports as they offer another option to send and receive data.

Prerequisites

The streaming analytics and SAP HANA Monitor operators in this example require a connection called HANA_DB1 of type HANA_DB. This connection is defined in the Connection Manager. Make sure your Connection Manager has this connection configured to point at your SAP HANA tenant database. If you use a different connection name, you'll need to also update the connection properties in the operators and pick your connection name from the drop-down list. Also, if you change the connection name, you'll need to also update the [service](#) property for the hana_out and db_in adapters in the project CCL to ensure they use the new connection name.

Run the Graph

In your SAP HANA tenant database, run following SQL to initialize the schema and tables required by this example:

```
CREATE SCHEMA STREAMING;
CREATE COLUMN TABLE STREAMING.TRADES
(
    ID INTEGER,
    TS BIGINT,
    SYMBOL VARCHAR(16),
    PRICE DECIMAL(20,2),
    VOLUME INTEGER,
    PRIMARY KEY ( ID )
);
CREATE COLUMN TABLE STREAMING.OLD_TRADES
(
    ID INTEGER,
    TS BIGINT,
    SYMBOL VARCHAR(16),
    PRICE DECIMAL(20,2),
    VOLUME INTEGER,
    PRIMARY KEY ( ID )
);
```

```

INSERT INTO "STREAMING"."OLD TRADES"( ID, TS, SYMBOL, PRICE, VOLUME ) VALUES
( 17701, 1543596817578, 'SAP', 30.58, 10166 );
INSERT INTO "STREAMING"."OLD TRADES"( ID, TS, SYMBOL, PRICE, VOLUME ) VALUES
( 17702, 1543596817678, 'SAP', 81.44, 13897 );
INSERT INTO "STREAMING"."OLD TRADES"( ID, TS, SYMBOL, PRICE, VOLUME ) VALUES
( 17703, 1543596817778, 'MSFT', 79.85, 10855 );
INSERT INTO "STREAMING"."OLD TRADES"( ID, TS, SYMBOL, PRICE, VOLUME ) VALUES
( 17704, 1543596817977, 'MSFT', 82.6, 10079 );
INSERT INTO "STREAMING"."OLD TRADES"( ID, TS, SYMBOL, PRICE, VOLUME ) VALUES
( 17705, 1543596818078, 'IBM', 59.45, 12622 );
INSERT INTO "STREAMING"."OLD TRADES"( ID, TS, SYMBOL, PRICE, VOLUME ) VALUES
( 17710, 1543596818978, 'MSFT', 42.53, 13409 );

```

To reset a tenant database's contents between test runs, use the following SQL:

```
delete from STREAMING.TRADES;
```

Related Information

[SAP HANA Streaming Analytics: Adapters Guide](#)

[Adding an Adapter to a Project](#)

3.10.2 Streaming Analytics Freezer Monitor Demo

This is an example of a more complex graph involving streaming analytics operators. In this demo, multiple streaming analytics operators interact to generate and process events such as power alarms and outages, and temperature readings and alarms from a fictional freezer monitor.

Prerequisites

Several streaming analytics operators in this example require a connection called HANA_DB1 of type HANA_DB. Define this connection in the Connection Manager and make sure that this connection is configured to point at your SAP HANA tenant database. If you use a different connection name than HANA_DB1, you'll need to update the [hana services](#) properties in all the affected operators and select your connection name from the drop-down list. Additionally, if you change the connection name, you'll also need to update the [service](#) property in the CCL of each of the affected operators.

Run the Graph

In your SAP HANA tenant database, run the following SQL to initialize the schema and tables required by this demo:

```

CREATE SCHEMA STREAMING;
CREATE COLUMN TABLE "STREAMING"."ACTIVITY_HIST"

```

```

(
    "MACHINEID"          VARCHAR(15),
    "EVENT_TIME"          TIMESTAMP,
    "EVENT_NAME"          VARCHAR(20),
    "EVENT_DESCRIPTION"    VARCHAR(30),
    "EVENT_VALUE"          VARCHAR(30)
);
CREATE COLUMN TABLE "STREAMING"."DASHBOARD"
(
    "MACHINEID"          VARCHAR(15) NOT NULL,
    "POWER_STATUS"        VARCHAR(30),
    "CURR_TEMP"           DECIMAL(4,2),
    PRIMARY KEY ("MACHINEID")
);
CREATE COLUMN TABLE "STREAMING"."MACHINE_REF"
(
    "MACHINEID"          VARCHAR(15) NOT NULL,
    "MACHINETYPE"         VARCHAR(15),
    "MAX_TEMP"            DECIMAL(4,2),
    "MIN_TEMP"            DECIMAL(4,2),
    "LOCATION"              VARCHAR(25),
    "TEMP_UNIT"           VARCHAR(2),
    PRIMARY KEY ("MACHINEID")
);
CREATE COLUMN TABLE "STREAMING"."POWER_OUTAGES"
(
    "MACHINEID"          VARCHAR(15),
    "POWER_OFF_TIME"      LONGDATE,
    "POWER_ON_TIME"       LONGDATE,
    "DURATION_MIN"        DOUBLE CS_DOUBLE
);
INSERT INTO "STREAMING"."MACHINE_REF" ("MACHINEID", "MACHINETYPE", "MAX_TEMP",
"MIN_TEMP", "LOCATION", "TEMP_UNIT") VALUES
( '2B', 'VEND', 50, 35, 'WALG31', 'F' );
INSERT INTO "STREAMING"."MACHINE_REF" ("MACHINEID", "MACHINETYPE", "MAX_TEMP",
"MIN_TEMP", "LOCATION", "TEMP_UNIT") VALUES
( '1A', 'COOL', 50, 35, 'FRI7', 'F' );
INSERT INTO "STREAMING"."MACHINE_REF" ("MACHINEID", "MACHINETYPE", "MAX_TEMP",
"MIN_TEMP", "LOCATION", "TEMP_UNIT") VALUES
( '4D', 'VEND', 50, 35, 'JRB235', 'F' );
INSERT INTO "STREAMING"."MACHINE_REF" ("MACHINEID", "MACHINETYPE", "MAX_TEMP",
"MIN_TEMP", "LOCATION", "TEMP_UNIT") VALUES
( '5E', 'COOL', 50, 35, 'WALG33', 'F' );
INSERT INTO "STREAMING"."MACHINE_REF" ("MACHINEID", "MACHINETYPE", "MAX_TEMP",
"MIN_TEMP", "LOCATION", "TEMP_UNIT") VALUES
( '3C', 'VEND', 10, 2, 'BP762', 'C' );
INSERT INTO "STREAMING"."MACHINE_REF" ("MACHINEID", "MACHINETYPE", "MAX_TEMP",
"MIN_TEMP", "LOCATION", "TEMP_UNIT") VALUES
( '6F', 'FREEZE', 32, 0, 'HIN88', 'F' );
INSERT INTO "STREAMING"."MACHINE_REF" ("MACHINEID", "MACHINETYPE", "MAX_TEMP",
"MIN_TEMP", "LOCATION", "TEMP_UNIT") VALUES
( '7G', 'VEND', 50, 35, 'ORD311', 'F' );

```

To clear the data generated by this demo from your tenant database, use the following SQL:

```

delete from STREAMING.ACTIVITY_HIST;
delete from STREAMING.POWER_OUTAGES;

```

3.10.3 Connecting to Kafka in Streaming Analytics Project CCL

This example graph shows how a streaming analytics project can use adapters to consume data from a Kafka topic that is produced using a Kafka Producer operator. It also shows how to publish to a Kafka topic that is then consumed using a Kafka Consumer operator. The Kafka Input and Output adapters are added to the streaming analytics operator using Continuous Computation Language (CCL).

Prerequisites

- In the project CCL, set the `kafkaBootstrapServers` property on the Kafka CSV Input and Output adapters to point to your Kafka broker(s). Do the same with the `Brokers` property on the Kafka Producer and Consumer operators.
- Upload the Kafka client jar to `/files/vflow/dockerfiles/com/sap/streaming/` using the System Management Files application. Only the Kafka 0.9.x and 0.10.x versions are supported.
- Update the streaming analytics Dockerfile to copy the Kafka client jar to `$STREAMING_HOME/adapters/framework/libj` folder. The streaming Dockerfile currently includes an example of that.

Related Information

[Setting Up the Kafka Adapters](#)

3.10.4 Sending Datatypes to Streaming Analytics Projects

This example graph shows how to send data rows containing various streaming analytics column datatypes to a streaming analytics project.

Input data can be generated in CSV or JSON format by setting the [outputFormat](#) property on the Data Generator operator and then setting the corresponding value for the 'input format' property on the attached streaming analytics operator. You can also specify the format of the data using the [output format](#) operator property. Note that when you attach two streaming analytics operators together, you'll need to ensure the output to input formats match.

This example also shows the use of string and message types for input and output ports. Both port types support CSV and JSON data formats.

Finally, the example shows how to batch input and output data. To change the batch size of incoming data, use the [batchSize](#) property in the Data Generator operator. The attached streaming analytics operator will then automatically process incoming data using this new size. To change the batch size of data being sent from the operator output ports, use the [output batch size](#) operator property.

Related Information

[Datatypes](#)

[SAP HANA Streaming Analytics: Developer Guide](#)

3.11 Utilities

Related Information

[Go Plugin Generator/Bootstrapper \[page 752\]](#)

[Wiretap \[page 753\]](#)

[Wiretap 2 \[page 754\]](#)

[Wiretap 3 \[page 754\]](#)

3.11.1 Go Plugin Generator/Bootstrapper

The Go operator allows a custom Go program to be compiled and dynamically loaded into a running Pipeline scenario.

Unfortunately, there is a limitation in the Go operator that it can only build a plugin library when there are no external package dependencies in the source code.

This limitation results in the error message:

```
plugin.Open("..."): plugin was built with a different version of package ...
```

Until this limitation is resolved, this graph can be used to build the compatible plugin libraries for a graph that contains Go programs with external package dependencies.

Prerequisites

- None.

Configure and Run the Graph

Follow the steps below to run this graph from the Data Pipeline UI:

1. Go to the configuration panel for operator Go PlugIn Gen (ID:gooperator1) and set property `graphName` to the name of the graph to generate the plugin library for. If this property is not visible in the configuration panel, open the JSON view of the configuration and look for the `config` property of operator with ID `gooperator1`. Set its property `graphName` manually. Save the graph.
2. Start the graph to generate the plugin library for the target graph.

3. After the graph successfully terminates, open and start the target graph.

In this particular graph, this operator is connected to operator Terminator(ID:graphterminator1), and as a result, this graph will terminate upon a successful plugin generation. If you wish to automatically start the target graph, connect the operator OpenAPI Client instead of Terminator. In this case, the connection properties of the OpenAPI Client must be set accordingly.

3.11.2 Wiretap

The Wiretap operator allows you to inspect messages that are transferred from one operator to another. This graph is a simple automatic game where two players, Jane and John, play against each other. At each round, having 13 games, each player picks one card out of a deck of cards numbered from 1 to 13 and the player who has the higher numbered card wins. The Wiretap operator is inserted in this graph to display both the card that each player picked and the result of each game.

Configure and Run the Graph

Follow the steps below to run the example from the SAP Data Intelligence Modeler UI:

1. Start the graph.
2. Open the UI from the wiretap operator so that the games are monitored.

For example, the following output may be displayed:

```
[2018-06-25 12:26:55.237] {"demo.card":5,"demo.game":
10,"demo.player":"donald","demo.round":1}
[2018-06-25 12:26:55.237] {"demo.card":10,"demo.game":
10,"demo.player":"hillary","demo.round":1}
[2018-06-25 12:26:55.240] {"demo.game":10,"demo.round":
1,"demo.winner":"hillary"}{"donald":5,"hillary":5}
[2018-06-25 12:26:57.236] {"demo.card":2,"demo.game":
11,"demo.player":"donald","demo.round":1}
[2018-06-25 12:26:57.236] {"demo.card":1,"demo.game":
11,"demo.player":"hillary","demo.round":1}
[2018-06-25 12:26:57.236] {"demo.game":11,"demo.round":
1,"demo.winner":"donald"}{"donald":6,"hillary":5}
[2018-06-25 12:26:59.236] {"demo.card":6,"demo.game":
12,"demo.player":"donald","demo.round":1}
[2018-06-25 12:26:59.236] {"demo.card":2,"demo.game":
12,"demo.player":"hillary","demo.round":1}
[2018-06-25 12:26:59.237] {"demo.game":12,"demo.round":
1,"demo.winner":"donald"}{"donald":7,"hillary":5}
[2018-06-25 12:27:01.237] {"demo.card":4,"demo.game":
13,"demo.player":"donald","demo.round":1}
[2018-06-25 12:27:01.237] {"demo.card":12,"demo.game":
13,"demo.player":"hillary","demo.round":1}
[2018-06-25 12:27:01.237] {"demo.game":13,"demo.round":
1,"demo.winner":"hillary"}{"donald":7,"hillary":6}
...
```

3.11.3 Wiretap 2

The Wiretap operator allows you to inspect messages that are transferred from one operator to another. This graph consists of two groups connected together and one script operator in group1 is emitting a sequence of "hallo" and two script operators in default group are emitting a sequence of "hello" and "hola", respectively. Three Wiretap operators are monitoring those messages.

More precisely speaking:

- script Hallo generates a series of hallo every 4 second in its own group of multiplicity 2, and its output is connected to wiretap3, and wiretap3 is connected to wiretap1.
- script Hello generates a series of hello every second in its default group, and its output is connected to wiretap1 and wiretap2.
- script Hola generates a series of hola every second in its default group, and its output is connected to wiretap2.

Configure and Run the Graph

Follow the steps below to run the example from the Data Pipeline UI:

1. Start the graph.
2. Open the UI from wiretap1 and active the tabs for the other wiretaps.

The following output are expected:

- wiretap1 shows a mixed series of hello and hallo.
- wiretap2 shows a mixed series of hello and hola.
- wiretap3 shows a series of hallo.

3.11.4 Wiretap 3

The Wiretap operator allows you to inspect messages that are transferred from one operator to another. This graph consists of two parts: one part with group multiplicity of 2, each generating its own series of data and monitoring the data at the first Wiretap operator Wiretap1 (ID:wiretap1), which in turn is connected to the second Wiretap operator Wiretap2 (ID:wiretap2) in the default group.

The Javascript operator Datagenerator (ID:javascrptv2operator1) is generating a series of hi-n, where n is a monotonically increasing integer number. The layout property of both Wiretap operators are set to [%i@node-%g] %m, which results in each line prefixed with the received index and the worker name.

Prerequisites

- A cluster instance to run this graph with multiple workers.

Configure and Run the Graph

Follow the steps below to run the example from the Modeler:

1. Start the graph;
2. Open the UI from Operator Wiretap1 and active the tab for wiretap2 so that both tabs wiretap1 and wiretap2 are shown.

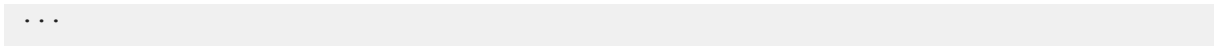
The following output is expected:

- wiretap1 is placed in the group with multiplicity of 2. Therefore, it shows two series of sequences: one with @node-group1-0 and the other with @node-group1-1, where each with its own incremental sequence of hi-n should look like:

```
[0@node-group1-0] hi-0
[1@node-group1-0] hi-1
[2@node-group1-0] hi-2
[3@node-group1-0] hi-3
[4@node-group1-0] hi-4
[5@node-group1-0] hi-5
[0@node-group1-1] hi-0
[1@node-group1-1] hi-1
[2@node-group1-1] hi-2
[3@node-group1-1] hi-3
[4@node-group1-1] hi-4
[5@node-group1-1] hi-5
[6@node-group1-1] hi-6
[6@node-group1-0] hi-6
[7@node-group1-0] hi-7
[7@node-group1-1] hi-7
[8@node-group1-0] hi-8
[8@node-group1-1] hi-8
[9@node-group1-0] hi-9
[9@node-group1-1] hi-9
[10@node-group1-0] hi-10
...
```

In contrast, wiretap2 is placed in the default group. Therefore, it shows a single sequence of data prefixed with @node-default, which should look like:

```
[0@node-default] hi-0
[1@node-default] hi-0
[2@node-default] hi-1
[3@node-default] hi-1
[4@node-default] hi-2
[5@node-default] hi-2
[6@node-default] hi-3
[7@node-default] hi-3
[8@node-default] hi-4
[9@node-default] hi-4
[10@node-default] hi-5
[11@node-default] hi-5
[12@node-default] hi-6
[13@node-default] hi-6
[14@node-default] hi-7
[15@node-default] hi-7
[16@node-default] hi-8
[17@node-default] hi-8
[18@node-default] hi-9
[19@node-default] hi-9
[20@node-default] hi-10
[21@node-default] hi-10
```



Important Disclaimers and Legal Information

Hyperlinks

Some links are classified by an icon and/or a mouseover text. These links provide additional information.

About the icons:

- Links with the icon  : You are entering a Web site that is not hosted by SAP. By using such links, you agree (unless expressly stated otherwise in your agreements with SAP) to this:
 - The content of the linked-to site is not SAP documentation. You may not infer any product claims against SAP based on this information.
 - SAP does not agree or disagree with the content on the linked-to site, nor does SAP warrant the availability and correctness. SAP shall not be liable for any damages caused by the use of such content unless damages have been caused by SAP's gross negligence or willful misconduct.
- Links with the icon  : You are leaving the documentation for that particular SAP product or service and are entering a SAP-hosted Web site. By using such links, you agree that (unless expressly stated otherwise in your agreements with SAP) you may not infer any product claims against SAP based on this information.

Videos Hosted on External Platforms

Some videos may point to third-party video hosting platforms. SAP cannot guarantee the future availability of videos stored on these platforms. Furthermore, any advertisements or other content hosted on these platforms (for example, suggested videos or by navigating to other videos hosted on the same site), are not within the control or responsibility of SAP.

Beta and Other Experimental Features

Experimental features are not part of the officially delivered scope that SAP guarantees for future releases. This means that experimental features may be changed by SAP at any time for any reason without notice. Experimental features are not for productive use. You may not demonstrate, test, examine, evaluate or otherwise use the experimental features in a live operating environment or with data that has not been sufficiently backed up.

The purpose of experimental features is to get feedback early on, allowing customers and partners to influence the future product accordingly. By providing your feedback (e.g. in the SAP Community), you accept that intellectual property rights of the contributions or derivative works shall remain the exclusive property of SAP.

Example Code

Any software coding and/or code snippets are examples. They are not for productive use. The example code is only intended to better explain and visualize the syntax and phrasing rules. SAP does not warrant the correctness and completeness of the example code. SAP shall not be liable for errors or damages caused by the use of example code unless damages have been caused by SAP's gross negligence or willful misconduct.

Bias-Free Language

SAP supports a culture of diversity and inclusion. Whenever possible, we use unbiased language in our documentation to refer to people of all cultures, ethnicities, genders, and abilities.

© 2021 SAP SE or an SAP affiliate company. All rights reserved.

No part of this publication may be reproduced or transmitted in any form or for any purpose without the express permission of SAP SE or an SAP affiliate company. The information contained herein may be changed without prior notice.

Some software products marketed by SAP SE and its distributors contain proprietary software components of other software vendors. National product specifications may vary.

These materials are provided by SAP SE or an SAP affiliate company for informational purposes only, without representation or warranty of any kind, and SAP or its affiliated companies shall not be liable for errors or omissions with respect to the materials. The only warranties for SAP or SAP affiliate company products and services are those that are set forth in the express warranty statements accompanying such products and services, if any. Nothing herein should be construed as constituting an additional warranty.

SAP and other SAP products and services mentioned herein as well as their respective logos are trademarks or registered trademarks of SAP SE (or an SAP affiliate company) in Germany and other countries. All other product and service names mentioned are the trademarks of their respective companies.

Please see <https://www.sap.com/about/legal/trademark.html> for additional trademark information and notices.