

FactoryTalk[®] Analytics[™]



FactoryTalk[®] Analytics[™] Edge User Guide Version 3.0

Contact Rockwell Automation

Customer Support Telephone - 1.440.646.3434

Copyright Notice

©2019 Rockwell Automation, Inc. All rights reserved. Printed in USA.

This document and any accompanying Rockwell Automation products are copyrighted by Rockwell Automation, Inc. Any reproduction and/or distribution without prior written consent from Rockwell Automation, Inc. is strictly prohibited. Please refer to the license agreement for details.

End User License Agreement (EULA)

To view the Rockwell Automation End-User License Agreement ("EULA"), log in to the application and click the Terms & Conditions link available on the lower-right side. You can also view the EULA by opening the EULA.pdf file located in your product's install folder on your hard drive.

Trademark Notices

FactoryTalk[®] Analytics[™] DataView, FactoryTalk[®] Analytics[™] DataFlowML, FactoryTalk[®] Analytics[™] Edge and the Rockwell Software logo are registered trademarks of Rockwell Automation, Inc.

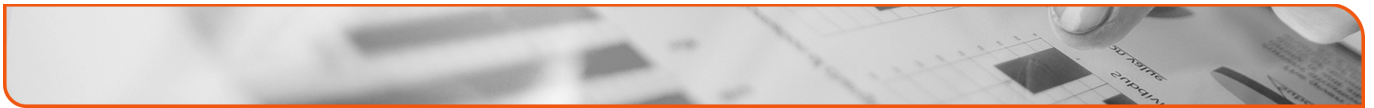
Table Of Contents

	Read Me First.....	5
	Audience	6
	Related Documents	6
	Conventions	7
	Solutions and Technical Support	7
	Security Recommendation	8
Chapter 1	Introduction.....	9
	Overview of Edge	10
	Features	11
	Getting Started	11
	Manage User Information	13
	Change Password	13
	User Dashboard.....	14
	Pipeline Configuration Summary.....	14
	Edge Nodes Deployment Summary	15
	Understanding Edge Components	16
	Collection of Data	16
	Adapter.....	16
	Ingress	16
	Transaction.....	16
	Analytics on Streaming Data	16
	Route to Next Point of Value	17
Chapter 2	Pipeline.....	19
	Pipeline	20
	Create a Pipeline	20

	Pipeline Rules	22
	Best Practices	30
	Deploy Pipeline	31
	Pipeline Options.....	33
	Pipeline Status.....	33
	Filter Pipelines.....	34
	Arrangement	34
	Filter Options	34
	Import Pipeline	35
	Import Exported (JSON) Pipeline.....	35
	Import Multiple Pipelines	36
	Import JSON-LD Pipeline	37
	Encrypt Pipeline	37
	Import JSON-LD Pipeline	38
	Additional Steps for Importing JSON-LD Pipeline	39
	Export Pipeline	41
Chapter 3	Configure Channels	45
	Channels	46
	Ingress.....	46
	Broker	50
	Transaction.....	51
	Configuration of Transaction with MQTT Ingress	53
Chapter 4	Configure Processors.....	57
	Processors	58
	Custom Processor	58
	Python	58
	PythonML	59
	Java - Class.....	60
	Java - jar	61
	JavaScript.....	62
	PMML.....	62
	Compression	63
	Decompression.....	64
	BSON Serialize.....	64
	BSON DeSerialize	65
	FileStore.....	65
	Delay.....	66

	Scheduler.....	67
	Remove	68
	Secondary Egress	69
	Group Tags.....	70
	Thinning Exception.....	72
	Thinning Compression.....	74
	Math Function (Calculation).....	76
	SqliteStore	77
	Writeback	78
	Connection Rules	78
	Data Types.....	79
	Configure Writeback.....	81
Chapter 5	Configure Emitters	85
	Emitters	86
Chapter 6	JSON Creation	93
	Sample JSON.....	94
	Channel Ingress.....	94
	FTLD.....	94
	CIP.....	95
	SQL	97
	MQTT.....	99
	DataSimulator.....	102
	JavaScript Custom Processor	103
	Sample .js File	103
	Writing a Custom Function	107
	Import Pipeline.....	107

This page has been intentionally left blank



Read Me First

In this chapter

- ❑ Audience 6
- ❑ Related Documents 6
- ❑ Conventions 7
- ❑ Solutions and Technical Support 7
- ❑ Security Recommendation 8

Audience

This document is intended for experienced professionals who understand their company's business needs as well as the technical terms and software dependencies described in this guide.

This document provides information about the User features of the product and how they can be utilized.

Related Documents

Select the specific guides for your Operating System and purpose.

Edge Installation Guides

There are two types of installation guides: Installer and Advanced Installation

☐ **Installer**

These guides will instruct a user to follow the installer step by step to install the application. Use this guide and the executable application file to install the application. The installer will unzip the files, allows you to accept the license agreement and install the application based on your selections.

- 30_FactoryTalk Analytics Edge Installer Guide (Windows)

NOTE: For first time installation and testing, using the installer version is a recommended starting point.

☐ **Advanced Installation**

These guides will instruct an experienced IT administrator step by step how to install the application. This method allows the administrator to manage a customized installation. This installer will unzip the files and allows you to accept the license agreement.

- 30_FactoryTalk Analytics Edge Advanced Installation Guide (Windows)
- 30_FactoryTalk Analytics Edge Runtime Advanced Installation Guide (Ubuntu)

Security Installation Guide

These guides are used in conjunction with the application Advanced Installation Guides. It will instruct an experienced IT administrator step by step how and when to install security.

- 30_Analytics Security Provider Advanced Installation Guide (Windows)

User Guides

These guides will provide user interface and configuration information after installation is completed.

- 30_FactoryTalk Analytics Edge User Guide
- 30_FactoryTalk Analytics Edge Administration Guide
- 30_Analytics Security Provider Administration Guide

Conventions

The following conventions are followed in this document:

- ❑ Text, labels and icon names that appear in the user interface appear in square brackets.
For example: On the Pipeline page and click the [Create New Pipeline] icon
- ❑ File names and directories appear in bold text.
For example: Open the **config.json** file.
- ❑ Placeholder text for variable values appears in angle brackets.
For example: EAP-<version>.<build>
- ❑ The cross-references appear in Green text. For example: “FTLD”
- ❑ The hypertext appears in Blue text. For example: <https://<IP address>:<port>/>
- ❑ Code and system response examples appear in Courier New font. For example:

```
{  
  "adapterName": "FTLD",  
  "bootOptions": [  
    "
```

Solutions and Technical Support

To contact technical support for issues, call (440) 646-3434 and access support using direct dial code 605.

When you call, you should be at your computer and prepared to give the following information:

- The product name and the version number, which can be found in the client.
- The type of hardware you are using.
- The exact wording of any errors or messages that appeared on your screen.
- A description of what happened and what you were doing when the problem occurred.
- A description of how you attempted to solve the problem.

Security Recommendation

We recommend you to follow your organizational security requirements and policies for setting up and operating the system. Some recommendations are below:

- Follow your organization guidelines on user management, user creation, access control etc.
- Set a complex password as per your company policy.
- Do not share your application user name, password and active sessions.
- Non-administrative users should not be allowed to change the configuration.



Chapter

1

Introduction

In this chapter:

- ☐ **Overview of Edge 10**
- ☐ **Getting Started 11**
- ☐ **Understanding Edge Components 16**

Overview of Edge

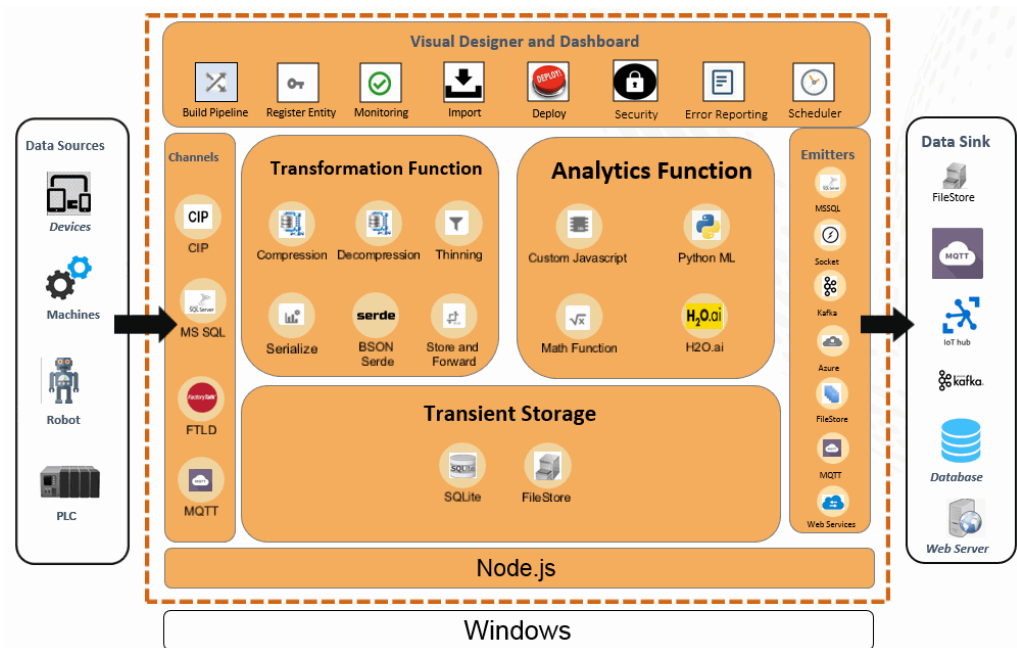
FactoryTalk[®] Analytics[™] Edge (called Edge hereafter) is a low foot-print Analytics Platform capable of Real-Time and Batch processing with low latency at the Edge layer. Edge supports data ingestion from multiple data sources such as FactoryTalk Live Data, SQL, CIP, MQTT, and can Egress processed data to various destinations like Message Queues, Cloud Sources, Socket, MSSQL, etc.

Edge comes with built in functions for Cleansing, Compression, Scheduling/Control, Temporary Storage and other capabilities. It also can extend the Processor capability by supporting the Custom Processors through which users can write custom functions.

Edge supports the custom function in Python, Machine Learning and H2O (JAVA). The user can leverage this functionality to write their own processor and ML module in Python. Edge also has the built-in capabilities to execute the PMML models generated by ThingWorx.

The Edge has a visual UI design interface and Node JS based runtime on which Pipelines are deployed. It can start, stop and monitor the status of deployment of the Pipeline through the UI itself. A Pipeline generally consists of a Channel (Data Sources), a Broker, a couple of Processors (functions) and an Emitter (data destinations).

Figure 1-1: Functional View



Features

- ❑ Designed for small Edge footprints, analytics at device.
- ❑ Visual Pipeline based configuration.
- ❑ Enable collection of data.
 - Real-time streaming data sources (CIP, OPC DA, FTLD etc.).
 - Data at Rest (SQL).
- ❑ Analytics on data in motion (Stream).
 - Data Cleansing, Refinement, Transformation, Store and Forward.
 - Machine Learning (Python ML, H2O, PMML).
- ❑ Route to next point of value.
 - Applications (Queues).
 - Datastore (RDBMS).
 - Cloud Endpoints (Azure Event Hub, IoT Hub).
 - Streaming Queue (Kafka).

Getting Started

Edge administrators are responsible for setting up and administering the instances. Other Users can access the application to build and deploy the Pipelines.

IMPORTANT: Refer to **FactoryTalk Analytics Edge Administration Guide for user roles and permissions.**

This document is a step-by-step guide for building Pipelines.

1. In the browser, use the provided URL:
 - <https://<FQDN-Hostname>:<port>>
 - **IP Address or hostname:** Server or hostname where the application is deployed. Make sure the URL is the same with that configured in Analytics Security Provider.
 - **Port:** The port is 8801 by default. The port number may vary based on installation and configuration.
2. The Edge Login page displays.
3. Enter the provided username and password and click [Log in].

Figure 1-2: Login Page



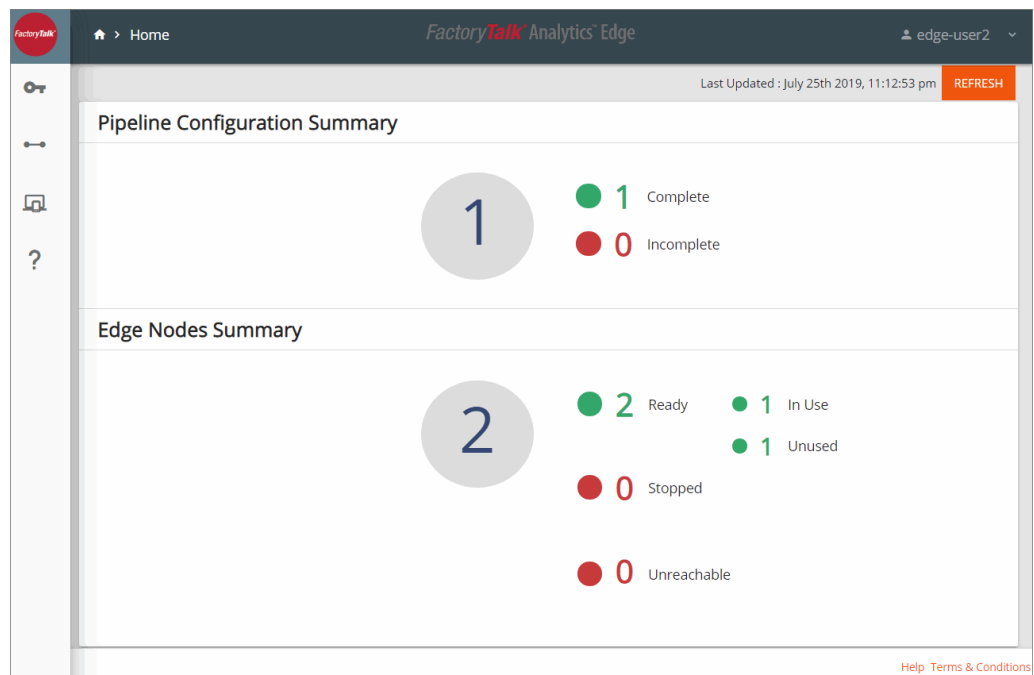
4. Depending on the setting, when the User was created, the Edge application may request to change the password. Enter the new password and click [Submit].

IMPORTANT: When changing the password, create a complex password as per your company policy.

Figure 1-3: Change Password



5. The Edge User Home page displays.

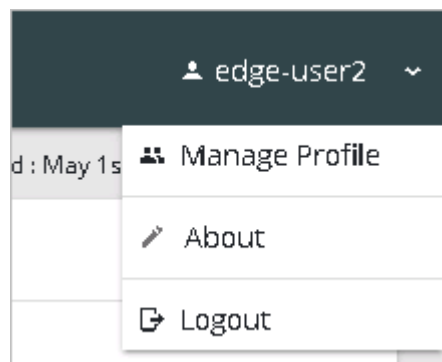
Figure 1-4: Home Page

Manage User Information

This section describes the settings of the User account.

Click the username dropdown, on the top right corner of the homepage. The following options are displayed:

- **Manage Profile:** Allows the logged-in User to change their password, view the session details and account log information.
- **About:** Displays information about the version of the application installed.
- **Logout:** Facilitates the User to quit the application.

Figure 1-5: User Information

Change Password

To change the current password:

1. Click [Manage Profile]. The Account window displays.
2. Select the [Password] tab to change the password.
3. Enter the current password and new password. Click [Save], to save new password.

IMPORTANT: When changing the password, create a complex password as per your company policy.

Figure 1-6: Change Password

User Dashboard

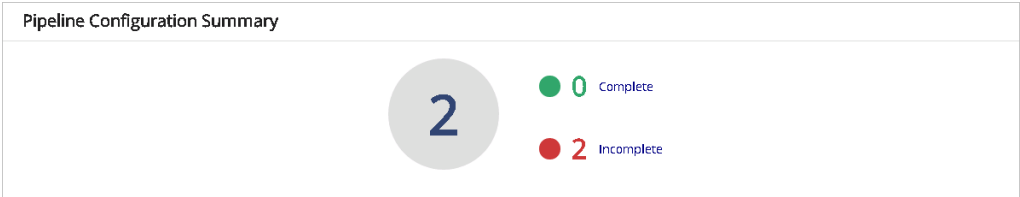
The User Dashboard is the landing page when a User logs in. The widgets on the Dashboard provide Pipeline Configuration Summary and Edge Nodes Deployment Summary.

Pipeline Configuration Summary

This widget provides a summary of Pipelines. The following table describes the Pipeline Summary and its components:

Pipeline Summary	
Pipelines	The number of Pipelines created by all the users.
Complete	Number of Pipelines configured completely.
Incomplete	Number of incomplete Pipelines.

Figure 1-7: Pipeline Configuration Summary



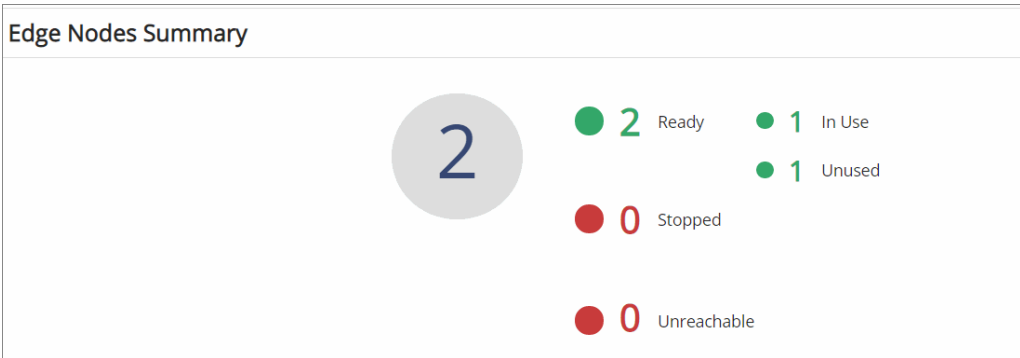
Edge Nodes Deployment Summary

This widget displays the summary of Nodes.

The following table describes the Nodes Deployment Summary and its components:

Nodes Summary		
Total Nodes		The number of Nodes configured successfully. It is split into the count of: Ready, Stopped and Unreachable
Ready	The number of Nodes on which the runtime core (CGP) and runtime agent (ShellApp) are active. The nodes are split into: In Use and Unused	
	In Use	Number of nodes on which a Pipeline has been deployed. The Pipeline may be in Active or Stopped state.
	Unused	Number of nodes that are ready but no Pipeline is deployed.
Stopped		The number of Nodes on which runtime agent (ShellApp) is active but runtime core (CGP) is inactive.
Unreachable		The number of Nodes on which the runtime core (CGP) and runtime agent (ShellApp) are inactive.

Figure 1-8: Edge Nodes Deployment Summary



Understanding Edge Components

Collection of Data

Edge collects data from the various sources and moves the data to Ingress.

- Real time streaming data source (CIP, OPC DA, FTLD, etc.).
- Data at Rest (SQL).

Adapter

Adapters are gateway components that receive actions (requests) and send reactions (responses).

Ingress

The process of Data Ingestion, to collect IoT/live data from different kinds of sources for data processing.

Edge collects IoT data from Channels like:

- **FTLD:** Data from live data sources i.e. FactoryTalk Live Data.
- **SQL:** Data from SQL database to execute inserts and store procedures.
- **CIP:** Data from PowerFlex drives and Controllers.
- **MQTT:** Data from MQTT brokers. Currently MQTT version 3.1.1 and Mosquitto version 1.5.4 are supported.

Edge also has a DataSimulator that generates test data for testing the Pipelines.

Transaction

Transaction acts as both an application and adapter on the gateway, allowing requests in the form of a trigger, an event to be processed and returned to requesting applications. The Transaction will egress both trigger and event data when the condition occurs.

Analytics on Streaming Data

Edge provides a platform to ingest real time data, process and transform it and create value in real time. The following encapsulated functions are supported which can be connected to form more complex functionality:

- Data Cleaning, Refinement and Thinning
- Transient Storage
- Scheduling and Delay
- Transformation
- Custom User Functions

- Machine Learning (Python ML, H2O and PMML-ThingWorx models)

Route to Next Point of Value

The data from an Emitter is used as input for various external platforms, stores and applications that require further analysis:

- FileStore
- MSSQL
- SQLite
- IOT Hub
- Azure Blob and Queue
- Kafka
- Socket IO
- MQTT
- Secure Webservice

This page has been intentionally left blank



Chapter

2

Pipeline

In this chapter:

- ❑ Pipeline 20
- ❑ Create a Pipeline 20
- ❑ Deploy Pipeline 31
- ❑ Filter Pipelines 34
- ❑ Import Pipeline 35
- ❑ Export Pipeline 41

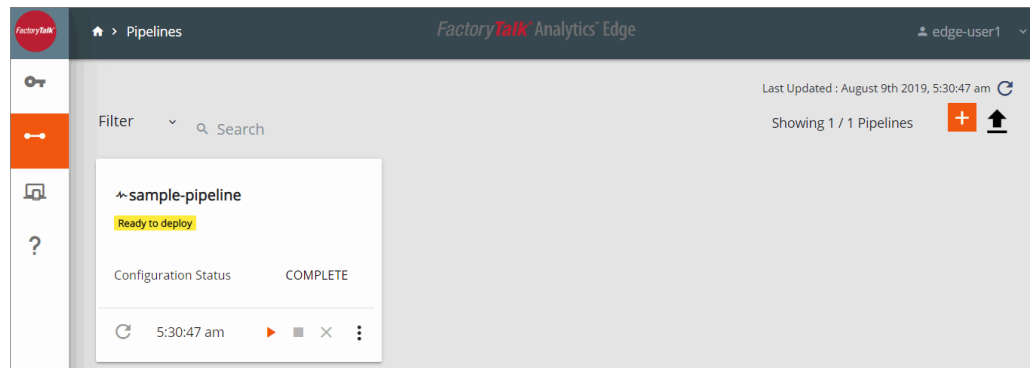
Pipeline

The Pipeline is a structured flow of data, which collects, processes, and analyzes high-volume data to generate real-time insights.

This section describes the components used and the steps to create a Pipeline.

The Pipelines menu page displays the cockpit view of the created Pipelines.

Figure 2-1: Pipeline Dashboard



Edge provides the ability to monitor the configuration and deployment status of Pipelines.


The roadmap to building a Pipeline is:

- **Pre-requisites:** Any actions required before the Pipeline can run, such as the configuration of components.
- **Data Ingestion:** Obtaining data from a source or sources by configuring Channels.
- **Data Transformation:** Manipulating the data acquired from the sources using Processors.
- **Data Analytics:** Applying mathematical or analytical including machine learning operations on the data of interest.
- **Data Persistence/Storage:** Saving the results in an external data store using Emitters.

Create a Pipeline

NOTE: Only Design Users can create and deploy the Pipelines. View Users can see the Pipeline status.
Ensure that the Adapters and the Emitters are registered before creating a Pipeline.

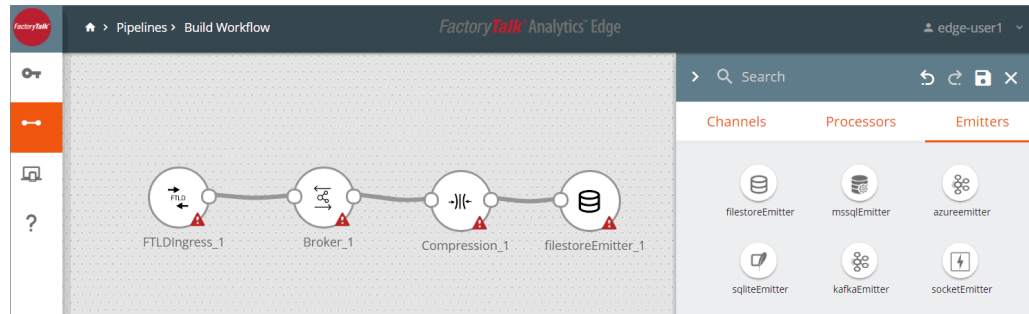
To create a new Pipeline, perform the following steps:


1. On the Pipelines page and click the [] icon.
2. Drag the components onto the Visual Designer (or canvas) or click the components from the right panel and connect the components.

NOTE: A Broker is a mandatory component for creating a Pipeline.

3. Right-click on each of the components to configure their properties.

Figure 2-2: Create Pipeline

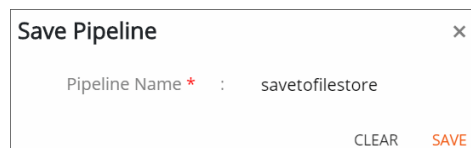


4. Click the [] icon and enter a name for the Pipeline.

NOTE: Pipeline names can contain only alphabets (a to z, A to Z), numbers (0 to 9), hyphen (-), underscore (_) and must start with a lower case alphabet.

The following Windows reserved filenames can not be used as a Pipeline name: CON, PRN, AUX, NUL, COM1, COM2, COM3, COM4, COM5, COM6, COM7, COM8, COM9, LPT1, LPT2, LPT3, LPT4, LPT5, LPT6, LPT7, LPT8, and LPT9

Figure 2-3: Save Pipeline



The screenshot shows a 'Save Pipeline' dialog box. It has a title bar with a close button (X). The main area contains the text 'Pipeline Name * : savetofilestore'. At the bottom right, there are two buttons: 'CLEAR' and 'SAVE'.

5. Click [SAVE]. The Pipeline is saved to the list.

Pipeline Rules

For a valid Pipeline, the output format of a component must match with the expected input format of the connecting component.

The Edge UI does not restrict connections between some individual components so that the custom processors can be used to fetch the actual data from Processors and do some manipulation on that data.

For example, connect an Ingress component to a Broker only. If you try to connect the Ingress directly with any Processors or Endpoints, the warning message “Wrong connection” will be displayed.

NOTE: Follow the pipeline rules described in the table as some errors will not display in the UI but will display in CGP when the Pipeline is deployed.

Component (Current point of value)	Can Connect to (Next point of value)	Can't Connect To (Next point of value)
Channels		
Ingress (FTLD, CIP, SQL, MQTT and DataSimulator)	Broker or Transaction	Any Ingress components Any Emitter components Any Processors Any Custom Processors
Broker	All Emitters All Custom Processors Processors: Calculation, Compression, FileStore, Thinning Compression, Thinning Exception, BSON Serialize, Group Tags, SqliteStore	Any Ingress components Transaction Processors: Decompression, Remove, Scheduler, Delay, BSON DeSerialize
Transaction	Broker	Any Ingress components Transaction Any Emitter components Any Processors Any Custom Processors

Component (Current point of value)	Can Connect to (Next point of value)	Can't Connect To (Next point of value)
Processors		
Compression	<p>Emitters: FileStore, SQLite, Azure Blob and Queue, IOT Hub, Kafka, Socket IO, Secure Webservice, MQTT</p> <p>Processors: Compression, Decompression, FileStore, Delay, BSON Serialize, SqliteStore</p>	<p>Emitter: MSSQL</p> <p>Any Ingress components</p> <p>Transaction</p> <p>Broker</p> <p>All Custom Processors</p> <p>Writeback</p> <p>Processors: Calculation, BSON DeSerialize, Remove, Scheduler, Thinning Compression, Thinning Exception, Group Tags</p>
Decompression	<p>Emitters: FileStore, SQLite, Azure Blob and Queue, IOT Hub, Kafka, Socket IO, Secure Webservice, MQTT</p> <p>Processors: Compression, Decompression, FileStore, Delay, BSON Serialize, SqliteStore</p>	<p>Emitter: MSSQL</p> <p>Any Ingress components</p> <p>Transaction</p> <p>Broker</p> <p>All Custom Processors</p> <p>Writeback</p> <p>Processors: Calculation, BSON DeSerialize, Remove, Scheduler, Thinning Compression, Thinning Exception, Group Tags</p>

Component (Current point of value)	Can Connect to (Next point of value)	Can't Connect To (Next point of value)
FileStore	<p>Processors: Compression, FileStore, Delay, Scheduler, Remove, BSON Serialize, SqliteStore</p> <p>All Emitters</p> <p><u>Note:</u> It is recommended to add the Delay or Scheduler processor before the endpoint. For example: Ingress -> Broker -> Store (Filestore/SQLite) -> Forward (Delay/Scheduler) -> Emitter</p>	<p>Any Ingress components</p> <p>Transaction</p> <p>Broker</p> <p>All Custom Processors</p> <p>Writeback</p> <p>Processors: Calculation, Decompression, BSON DeSerialize, Thinning Compression, Thinning Exception, Group Tags</p>
Delay	<p>All Emitters</p> <p>Processors: Compression, FileStore, Delay, Scheduler, Remove, BSON Serialize, SqliteStore</p>	<p>Any Ingress components</p> <p>Transaction</p> <p>Broker</p> <p>All Custom Processors</p> <p>Writeback</p> <p>Processors: Calculation Decompression, BSON DeSerialize, Thinning Compression, Thinning Exception, Group Tags</p>

Component (Current point of value)	Can Connect to (Next point of value)	Can't Connect To (Next point of value)
Scheduler	<p>All Emitters</p> <p>Processors: Compression, FileStore, Delay, Scheduler, Remove, BSON Serialize, SqliteStore</p>	<p>Any Ingress components</p> <p>Transaction</p> <p>Broker</p> <p>All Custom Processors</p> <p>Writeback</p> <p>Processors: Calculation, Decompression, BSON DeSerialize, Thinning Compression, Thinning Exception, Group Tags</p>
Remove	All Emitters	<p>Any Ingress components</p> <p>Broker</p> <p>Transaction</p> <p>All Custom Processors</p> <p>Writeback</p> <p>Any Processors</p>

Component (Current point of value)	Can Connect to (Next point of value)	Can't Connect To (Next point of value)
Thinning Compression	<p>All Emitters</p> <p>All Custom Processors</p> <p>Writeback</p> <p>Processors: Calculation, Compression, FileStore, Delay, Thinning Compression, Thinning Exception, BSON Serialize, Group Tags, SqliteStore</p>	<p>Any Ingress components</p> <p>Transaction</p> <p>Broker</p> <p>Processors: Decompression, Scheduler, BSON DeSerialize, Remove</p>
Thinning Exception	<p>Emitters: FileStore, SQLite, Azure Blob and Queue, IOT Hub, Kafka, Socket IO, Secure Webservice, MQTT</p> <p>All Custom Processors</p> <p>Writeback</p> <p>Processors: Calculation, Compression, FileStore, Delay, Thinning Compression, Thinning Exception, BSON Serialize, Group Tags, SqliteStore</p>	<p>Emitter: MSSQL</p> <p>Any Ingress components</p> <p>Transaction</p> <p>Broker</p> <p>Processors: Decompression, Scheduler, BSON DeSerialize, Remove</p>

Component (Current point of value)	Can Connect to (Next point of value)	Can't Connect To (Next point of value)
BSON Serialize	<p>Emitters: FileStore, SQLite, Azure Blob and Queue, IOT Hub, Kafka, Socket IO, Secure Webservice, MQTT</p> <p>Processors: Compression, FileStore, Delay, BSON Serialize, BSON DeSerialize, SqliteStore</p>	<p>Emitter: MSSQL</p> <p>Any Ingress components</p> <p>Broker</p> <p>Transaction</p> <p>All Custom Processors</p> <p>Writeback</p> <p>Processors: Calculation, Decompression, Scheduler, Remove, Thinning Compression, Thinning Exception, Group Tags</p>
BSON DeSerialize	<p>Emitters: FileStore, SQLite, Azure Blob and Queue, IOT Hub, Kafka, Socket IO, Secure Webservice, MQTT</p> <p>Processors: Compression, FileStore, Delay, BSON Serialize, BSON DeSerialize, SqliteStore</p>	<p>Emitter: MSSQL</p> <p>Any Ingress components</p> <p>Transaction</p> <p>Broker</p> <p>All Custom Processors</p> <p>Writeback</p> <p>Processors: Calculation, Decompression, Scheduler, Remove, Thinning Compression, Thinning Exception, Group Tags</p>

Component (Current point of value)	Can Connect to (Next point of value)	Can't Connect To (Next point of value)
Group Tags	<p>All Emitters</p> <p>All Custom Processors</p> <p>Writeback</p> <p>Processors: Calculation, Compression, FileStore, Delay, Thinning Compression, Thinning Exception, BSON Serialize, Group Tags, SqliteStore</p>	<p>Any Ingress components</p> <p>Transaction</p> <p>Broker</p> <p>Processors: Decompression, Scheduler, BSON DeSerialize, Remove</p>
Calculation	<p>All Emitters</p> <p>All Custom Processors</p> <p>Writeback</p> <p>Processors: Calculation, Compression, FileStore, Delay, BSON Serialize, Thinning Compression, Thinning Exception, Group Tags, SqliteStore</p>	<p>Any Ingress components</p> <p>Broker</p> <p>Transaction</p> <p>Processors: BSON DeSerialize, Decompression, Scheduler, Remove</p>

Component (Current point of value)	Can Connect to (Next point of value)	Can't Connect To (Next point of value)
SqliteStore	<p>Processors: Delay, Scheduler</p> <p>All Emitters</p> <p><u>Note</u>: It is recommended to add the Delay or Scheduler processor before the endpoint. For example: Ingress -> Broker -> Store (Filestore/SQLite) -> Forward (Delay/Scheduler) -> Emitter</p>	<p>Any Ingress components</p> <p>Transaction</p> <p>Broker</p> <p>All Custom Processors</p> <p>Writeback</p> <p>Processors: Compression, Decompression, FileStore, Calculation, BSON Serialize, BSON DeSerialize, Thinning Compression, Thinning Exception, Group Tags, Remove, SqliteStore</p>
Custom Processors		
JavaScript, Python, Python ML, Java, PMML	<p>All Emitters</p> <p>Any Custom Processors</p> <p>Writeback</p> <p>Processors: Calculation, Compression, FileStore, Delay, BSON Serialize, Thinning Compression, Thinning Exception, Group Tags, SqliteStore</p>	<p>Any Ingress components</p> <p>Transaction</p> <p>Broker</p> <p>Processors: BSON DeSerialize, Decompression, Scheduler, Remove</p>

Component (Current point of value)	Can Connect to (Next point of value)	Can't Connect To (Next point of value)
Writeback	All Emitters Any Custom Processors Writeback Processors: Calculation, Compression, FileStore, Delay, BSON Serialize, Thinning Compression, Thinning Exception, Group Tags, SqliteStore	Any Ingress components Transaction Broker Processors: BSON DeSerialize, Decompression, Scheduler, Remove
Emitters		
All Emitters		Any Ingress components Transaction Broker Any Processors Any Custom Processors Any Emitter components

Best Practices

- Do not transform the data (such as Compression, Decompression, BSONSerialize, BSONDeserialize) before the pipes are split. Split the pipes after the Broker and then add the processors that transform the data.
- If a Pipeline is built with Adaptor > Broker > Compression > Decompression > BSONSerialize > BSONDeserialize > Emitter and try to deploy, CGP will report an error.
- If any one of the following components are used in a Pipeline then MSSQL Egress cannot be used.
 - ❖ Compression
 - ❖ Decompression
 - ❖ BSON Serialized
 - ❖ BSON De-Serialized
- If the Socket client is not connected to the Socket IO Endpoint or the datastore is not ready, data will not be posted, and an error message such as

‘no clients connected’ is displayed in the event log or in the Edge error logs. It is recommended to use a Delay Processor in the Pipeline to avoid this type of error.

- The MSSQL endpoint works well for transactions where all the tags needed are together. In order to emit the tags together, use the Transaction block or the Group Tags processor in a Pipeline.

For example:

Ingress -> Broker -> Scheduler -> Group Tags -> Emitter:MSSQL

- Calculation is done based on the available tags at that moment. To wait for all the tag data and then process the result by the Calculation processor, it is recommended to use the Group Tags processor before Calculation.

For example: Ingress -> Broker -> Group Tags -> Calculation -> Emitter

- When a FileStore processor is directly connected to MSSQL, SocketIO or MQTT Emitters, error pops up in the Runtime Core (CGP). Add a Delay or Scheduler processor after the FileStore processor to avoid the error.

For example:

Ingress -> Broker -> Store (Filestore) -> Forward (Delay/Scheduler) -> Emitter

- When a SqliteStore processor is directly connected to an Emitter, data is not egressed to endpoint and only persisted in SQLite. Add a Delay or Scheduler processor after the SqliteStore processor to egress the data.

For example:

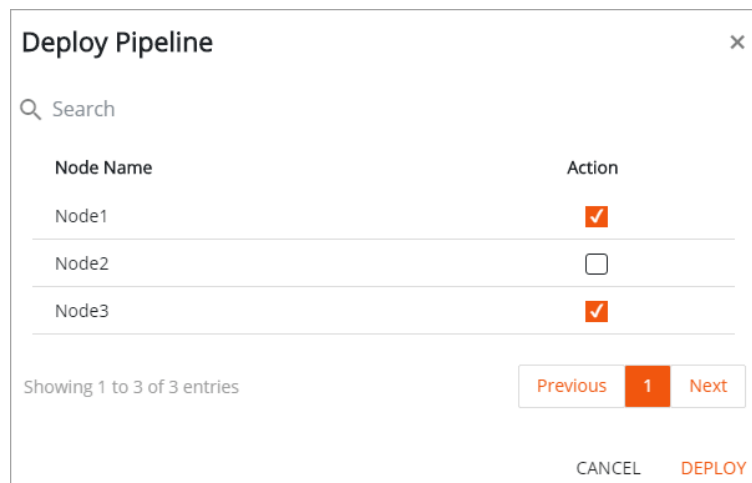
Ingress -> Broker -> Store (SqliteStore) -> Forward (Delay/Scheduler) -> Emitter

Deploy Pipeline

NOTE: Ensure that the Runtime Node is configured before deploying a Pipeline.

NOTE: Pipelines that have the FTLD ingress or the PMML Custom Processor cannot be deployed on Ubuntu Runtime Node. Deploy such Pipelines on a Windows Runtime Node.

1. Click the [▶] icon on the respective Pipeline. Deploy Pipeline dialog displays.
2. Select the Node or Nodes on which the Pipeline is to be deployed and click [DEPLOY].

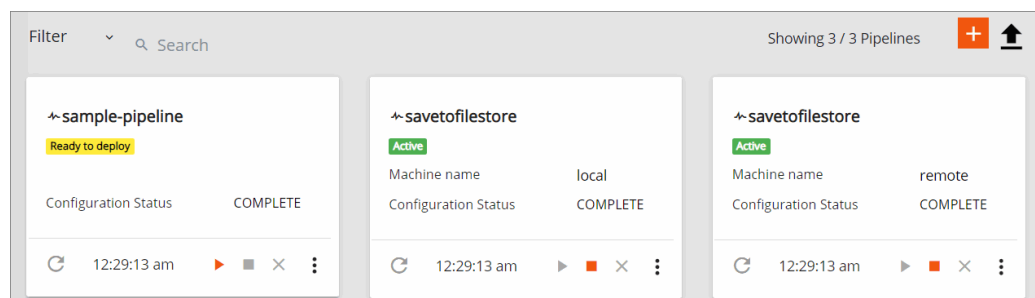
Figure 2-4: Select Nodes

3. If more than one Node is selected, the Pipeline is duplicated and deployed on the selected Nodes.
4. The Deploy success message displays. Duplicated Pipelines are displayed with the same name on the Pipelines page for each Node.

IMPORTANT: After deploying the Pipeline, check if any errors occurred in the Pipeline. Check the event viewer (Click on Windows ->Event Viewer -> Windows Logs -> Application) to view the error logs. If any errors occur, undeploy the Pipeline and correct the configuration causing the error. Now, deploy the pipeline again.

Enable Pipeline error monitoring on Edge UI, to view runtime logs.

NOTE: An active or a stopped Pipeline cannot be edited / changed. Only when the Pipeline is undeployed can it be modified.

Figure 2-5: Pipelines on Multiple Nodes

5. To stop a Pipeline, click the [■] icon.
6. To resume a stopped Pipeline, click the [▶] icon.

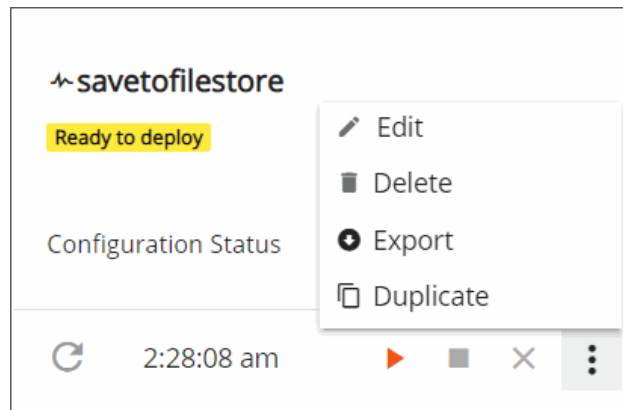
7. To undeploy a stopped Pipeline, click the [✕] icon.
8. For a Pipeline deployed on multiple Nodes, the following rules apply:
 - a. The Pipeline can be edited only if it is Undeployed from all the Nodes. Editing a Pipeline will edit all the Pipelines with the same name.
 - b. The Pipeline cannot be deleted if any of the Pipelines with the same name are in an Active or Stopped state.
 - c. The User can individually Stop, Resume or Undeploy operations on each of the Nodes.
 - d. The Pipeline will remain on the page if it is undeployed.
 - e. Undeploy and Redeploy will reuse the existing duplicate Pipeline (UI will select based on Name and IP combination).

Pipeline Options

Click the [⋮] icon shown adjacent to the Undeploy button. The Pipeline options display:

- View - To view the Pipeline (View User has this option).
- Edit - To edit the Pipeline.
- Delete - To delete the Pipeline.
- Export - To export the Pipeline.
- Duplicate - To copy the Pipeline and save it with a different name.

Figure 2-6: Pipeline Options



Pipeline Status

Different statuses of a Pipeline:

- Active - The Pipeline is currently running on the Edge Node.
- Stopped - The Pipeline is currently stopped on the Edge Node.
- Ready to Deploy - The Pipeline can be deployed on to an Edge Node.

- **Active** - When the Pipeline is deployed, the Pipeline status changes to Deploy In-progress and then to Active.
- **Stopped** - When the Pipeline is stopped, the Pipeline status changes to Stopping In-progress and then to Stopped.
- **Ready to deploy** - The Pipeline is ready to deploy. While undeploying the Pipeline, the Pipeline status changes to Undeploy In-progress and then Ready to deploy.
- **No Response from Edge Runtime Agent** - When the node server is stopped, refresh the Pipeline status. The status changes from “Active” to “Refresh In-progress” and then to “No Response from Edge Runtime Agent”.
- **No Response from Edge Runtime Core** - When the runtime core (cgp.exe) is stopped, refresh the pipeline status. The status changed from “Active” to “Refresh In-progress” and then to “No Response from Edge Runtime Core”.
- **Error** - When error occurs while running the Pipeline, the Pipeline status will change to Error.
- **Not Available** - When the pipeline is deleted, refresh the Pipeline status. The status changes to “Refresh In-progress” and then to “Not Available”.

Filter Pipelines

Arrangement

By default, all the Pipelines are shown on the Pipeline dashboard and are arranged in alphabetical order of Pipeline name.

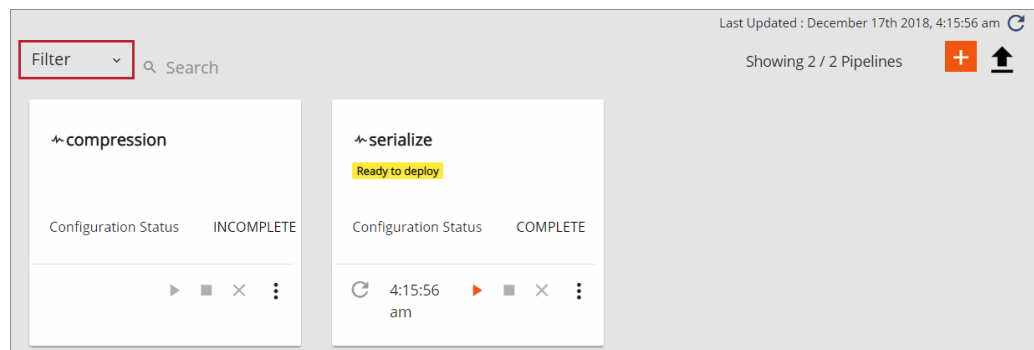
Filter Options

The Filter option allows adding the following filtering conditions in the Pipeline cards screen:

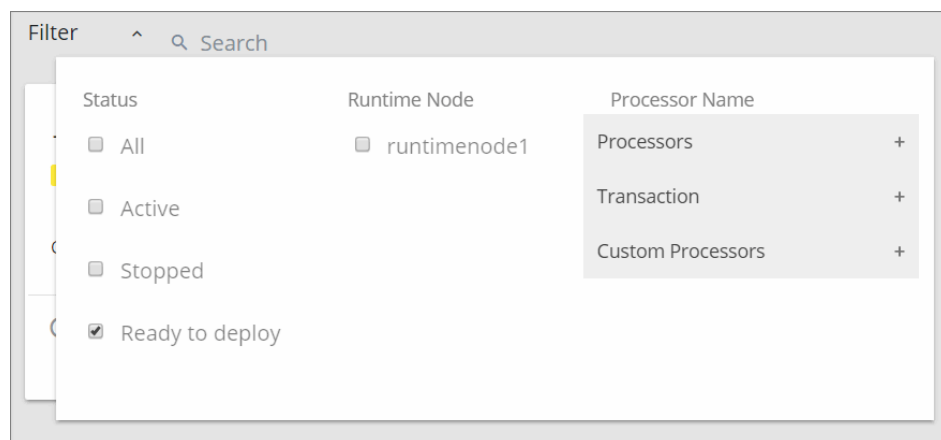
- **By Status** - Pipelines having the status condition met
- **By Runtime node**- Pipelines that are deployed or stopped using the node name
- **By Processor name** - Pipelines using the Processor name, including the pre-built and custom processors
- **By Transaction name** - Pipelines using the transaction name
- **By Custom Processor name** - Pipelines using the Custom Processor name

Perform the following steps to apply the filter:

1. Click the [Filter] dropdown.

Figure 2-7: Filter

2. Select the desired components to apply filter condition. One or more filter conditions can be selected.

Figure 2-8: Apply Filter

3. The Pipeline dashboard displays only the filtered Pipelines.

Import Pipeline

The Import Pipeline feature imports a Pipeline along with its component configurations such as Adapters, Processors and Emitters.

Import Exported (JSON) Pipeline

Perform the following steps to import a Pipeline that is exported from Edge:


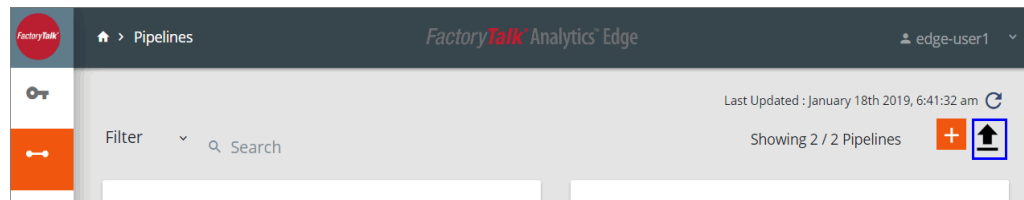
1. Click the [] icon on the Pipelines menu page.

Figure 2-9: Import Pipeline



2. The Import JSON/JSON-LD dialog displays.
3. To import the Pipeline, select [JSON] as File Type from the dropdown. Click the Upload icon and upload the **pipelines.zip** file.

The default Pipeline name contains “copy” at the end to differentiate from the original Pipeline. The name can be edited.

Figure 2-10: Import JSON Pipeline

4. Click [SAVE]. The Pipeline displays in the dashboard.

Import Multiple Pipelines

Perform the following steps to import multiple Pipelines that are exported from Edge:

1. To import the Pipelines exported from Edge, select [JSON] as File Type from the dropdown. Click the Upload icon and upload the **pipelines.zip** file. A maximum of five Pipelines can be imported at a time.

The default Pipeline name contains “copy” at the end to differentiate from the original Pipeline. The name can be edited.

Figure 2-11: Import JSON Pipeline

Import JSON/JSON-LD

Pipeline Type *: JSON

Import File *: Pipelines.zip

Pipeline Name 1*: sample-pipelinecopy

Pipeline Name 2*: savetofilestorecopy

Pipeline Name 3*: multiplicationcopy

Pipeline Name 4*: predictspeedcopy

CLEAR SAVE

2. Click [SAVE]. The Pipelines display in the dashboard.

Import JSON-LD Pipeline

There are two options to import a JSON-LD Pipeline. In the first option, user can import an encrypted Pipeline (.dat extension) to ensure security of any sensitive information. The other option is to import a non-encrypted Pipeline (.json extension). It is recommended to use the encrypted Pipeline import option.

Encrypt Pipeline

Perform the following steps to encrypt a JSON-LD file:

1. Place the JSON file in the ShellApp folder.
2. Open a Command Prompt window and change the working directory to the ShellApp folder.
3. To encrypt the JSON file using a key, run the following command:

```
cgp.exe -e <file name>.json -ek <key for encryption>
```

For Example:

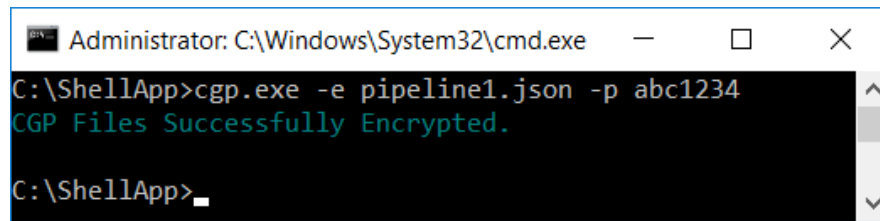
```
cgp.exe -e pipeline.json -ek
eyJhbGciOiJSU0ExXzUiLA0KICJlbnMiOiJBMjU2R0NNIiwNCiAiaXYiOiJfZXZ
```

To encrypt the JSON file using a password, run the following command:

```
cgp.exe -e <file name>.json -p <password for encryption>
```

For Example:

```
cgp.exe -e pipeline.json -p abc1234
```

Figure 2-12: Encrypt with Password

4. An encrypted configuration file called "cgp-config.dat" will be generated in the **ShellApp** folder. Rename the "cgp-config.dat" file to "<Pipeline Name>.dat". If the password method is used to encrypt, **salt.bin** will also be generated in the ShellApp folder.

Import JSON-LD Pipeline

Perform the following steps to import a JSON-LD Pipeline:

1. To import a JSON-LD pipeline, select [JSON-LD] as File Type from the dropdown. Click the Upload icon and upload the JSON-LD file. Enter the Application (Pipeline) name. Ensure that the file name matches with application name.

Figure 2-13: Define Import Pipeline

NOTE: If the JSON-LD file contains SQL, the password for the SQL server should be in clear text.

If the JSON-LD file has SSL configurations for the Channel or Emitter, the corresponding keys and certificates will not be imported. Do not import a Pipeline (JSON-LD) that has SSL configuration for the Channel or Emitter. It is recommended to configure such Pipelines through the Edge UI.

2. To upload a key encrypted Pipeline file, upload the DAT file. Select [Key] and provide the key that was used to encrypt the JSON-LD file.

IMPORTANT: While importing the Pipeline, the encryption of any sensitive data in the configuration file should be ensured by the User. We recommend using an encrypted configuration file.

Figure 2-14: Import JSON-LD

3. To upload a password encrypted Pipeline file, upload the DAT file. Select [Password] and provide the password that was used to encrypt the JSON-LD file.

Ensure that the **salt.bin** created while encrypting the Pipeline is available in the **ShellApp** folder of Runtime.

4. Click [SAVE]. The workflow creation success message displays and the Pipeline displays in the Pipelines menu page.

NOTE: An Imported JSON-LD Pipeline cannot be Edited, Duplicated, or Exported. But, an Imported JSON Pipeline can be Edited, Deleted, Exported and Duplicated.

Additional Steps for Importing JSON-LD Pipeline

NOTE: The following instructions assume that the JSON-LD file is available with User.

If the imported Pipeline contains the Custom Functions (Processors), those Custom Functions need to be copied to the target Runtime (CGP) node as follows:

1. Locate the ShellApp folder on the Runtime node where the imported pipeline would be deployed.
2. Open the .json file in a text editor.

3. Search for the String: "uri://ra/cgp/config/application"
4. Search for the item "ra-ctx#", where # starts from 0. Its value indicates where the custom function .js file resides in the ShellApp node.
5. Make sure that the value is "./shared/FILE_NAME.js", where FILE_NAME is the file name of your custom function.
6. Make sure that the JSON-LD application name, JSON-LD filename, and the Pipeline name match.
7. Copy the Custom Function .js file to the following folder:
C:\ShellApp\shared\I_pipeline\custom_processor_name
Where 'pipeline' is the name of the Pipeline and
'custom_processor_name' is the name of the custom processor.
8. If the imported JSON-LD has Java PMML (PMML) custom processor, perform following steps:
 - a. Copy the PMML model file to the following folder:
C:\ShellApp\shared\I_pipeline\custom_processor_name
Where 'pipeline' is the name of the Pipeline and
'custom_processor_name' is the name of the PMML processor.
 - b. Get the port number from each custom processor component. Search for the item "port". Its value indicates the port number.
 - c. Create a Java PMML Windows service. Update name of the service, port number and the path to custom function files in the following command and execute to install the Windows service.

```
cd C:\ShellApp\RAFunctions\JPMML

mlservice.exe //IS//<Service Name>
--Install="C:\ShellApp\RAFunctions\JPMML\mlservice.exe"
--Description="FTA Edge ML Service" --Jvm=auto
--Startup=auto
--Classpath="C:\ShellApp\RAFunctions\JPMML\ftaedgepmml-1.0.0.jar" --StartMode=jvm --StartClass=edgejpmml.mlservice
--StartMethod=main --StartParams="start#<Port Number>#c:\ShellApp\shared\I_pipeline\custom_processor_name" --StopMode=jvm --StopClass=edgejpmml.mlservice
--StopMethod=main --StopParams="stop#<Port Number>#c:\ShellApp\shared\I_pipeline\custom_processor_name" --LogPath="c:\ShellApp\RAFunctions\JPMML\logs<Port Number>" --StdOutput=auto --StdError=auto
```

For Example:

```
mlservice.exe
//IS//FTAEDGEMLService25335--Install="C:\ShellApp\RAFunctions\JPMML\mlservice.exe" --Description="FTA Edge ML Service"
--Jvm=auto --Startup=auto
--Classpath="C:\ShellApp\RAFunctions\JPMML\ftaedgepmmml-1.0.0.jar" --StartMode=jvm --StartClass=edgejpmml.mlservice
--StartMethod=main
--StartParams="start#25335#c:\ShellApp\shared\I_pmmmlpipe\pmmlcustom" --StopMode=jvm --StopClass=edgejpmml.mlservice
--StopMethod=main
--StopParams="stop#25335#c:\ShellApp\shared\I_pmmmlpipe\pmmlcustom" --LogPath="c:\ShellApp\RAFunctions\JPMML\logs25335"
--StdOutput=auto --StdError=auto
```

d. Open the Windows services console and start the Windows service.

9. Deploy the Pipeline.

NOTE: When undeploying the Java PMML Pipeline, also stop and delete the related ML service.

Export Pipeline

Existing Pipelines from an Edge server can be exported so that they can be stored for backup, to be imported back onto an Edge server or to be shared with other Design Users.

Exporting Pipelines is supported irrespective of the deployment status of the Pipeline (Active, Stopped, Ready to Deploy).

Currently, exporting the following Pipelines is not supported:

- If the Pipeline is not completely configured.
- If the Pipeline was originally created via importing JSON-LD

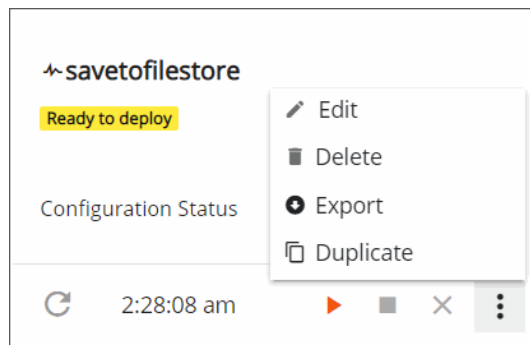
For security reasons, passwords and certificates are not exported. If a Pipeline that contains any of the following components is exported and then imported, it is saved as an "INCOMPLETE" Pipeline and the corresponding component need to be configured again.

- SQL Ingress
- MQTT Ingress, with useSSL option set to True
- MSSQL Emitter
- Kafka Emitter, with useSSL option set to True
- Secure Webservice Emitter, with useSSL option set to True
- MQTT Emitter, with useSSL option set to True

Perform the following steps to export a single Pipeline:

1. Click the [⋮] icon shown adjacent to the Undeploy button. The Pipeline options display:

Figure 2-15: Pipeline Export Option



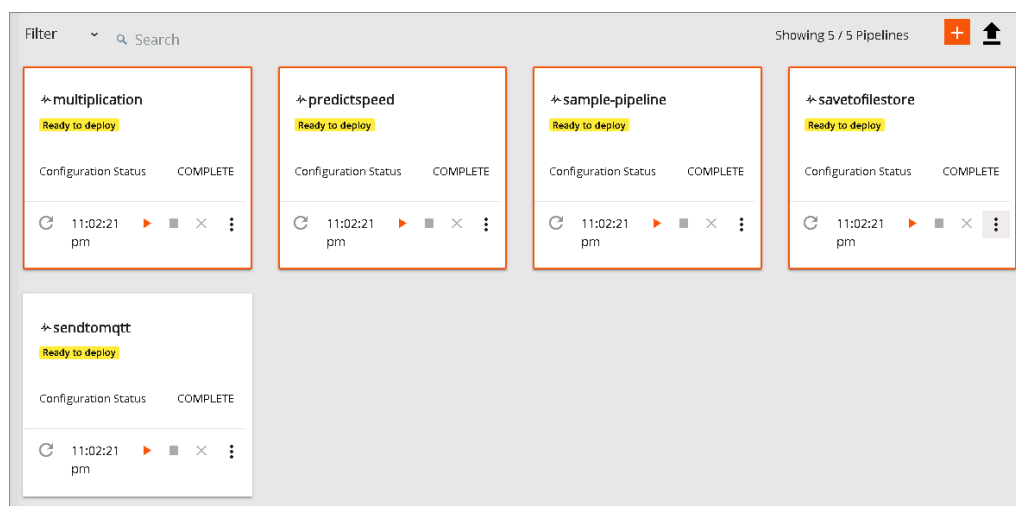
2. Select [Export]. The selected Pipeline data will be downloaded as **pipelines.zip** file and the success message (Workflow Exported Successfully) is displayed.

Perform the following steps to export multiple Pipelines:

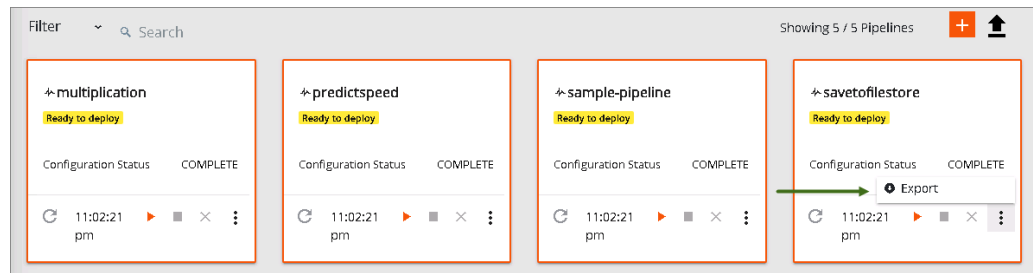
1. Press and hold the Ctrl key and click the Pipeline cards to select multiple Pipelines. The selected cards are highlighted. A maximum of five Pipelines can be exported at a time.

Use Esc key to clear all the selected cards or press and hold Ctrl key and click again the selected card to clear the selection.

Figure 2-16: Select Multiple Pipelines



2. Click the [⋮] icon on any of the selected cards. The [Export] option displays:

Figure 2-17: Export Multiple Pipelines

3. Select [Export]. The selected Pipelines and related data will be downloaded as **pipelines.zip** file and the success message (Workflow Exported Successfully) is displayed.

If the Pipelines with same name (i.e., Pipeline deployed on multiple Nodes) are exported, the downloaded .zip file contains only one JSON file.

NOTE: Refer to the “[Import Pipeline](#)” section, to import the exported Pipelines.

This page has been intentionally left blank



Chapter

3

Configure Channels

In this chapter:

- ❑ Channels 46

Channels

Data access in Edge is recognized by Channels that are built-in drag and drop operators to consume data from various data sources. Channel consists of the following components:

- ❑ “Ingress”
- ❑ “Broker”
- ❑ “Transaction”

Ingress

Ingress is the process of Data Ingestion where IoT and live data can be collected from different data processing sources.

Edge collects IoT data from Channels:

- **FTLD**: Data from live data sources i.e. FactoryTalk Live Data.
- **SQL**: Data from SQL database to execute inserts and store procedures.
- **CIP**: Data from PowerFlex drives and Controllers.
- **MQTT**: Data from MQTT brokers. Currently, MQTT version 3.1.1 and Mosquitto version 1.5.4 are supported.
- **DataSimulator**: Generates test data for testing the Pipelines.

To configure an Ingress Channel, perform the following steps:

1. Click and drag the Ingress Channel onto the Visual Designer and right-click the Ingress to configure.
2. If required, change the default properties.
3. Select the Pipe Path from the dropdown list. If there is only one Emitter in the Pipeline, Pipe Path is automatically selected. Manually select the Pipe Paths if there is more than one Emitter or Pipe Path. Click [SAVE].

NOTE: The Pipe Path field is enabled only after connecting all the components in the Pipeline.

Figure 3-1: Configure Ingress - FTLD

Configuration Settings - FTLDIngress_1

Select Source

FTLD Device Shortcut (Path) * : ra-ftld://cgp-ftld/RNA://\$Global/TestApi

Tag Name * : rawdata

Read Option * : Continuous Read

Tracking ID : Please enter value

Update Frequency (ms) * : 1000

Update Type * : Polling

Pipe Path * : Filestore_1_path-659

CANCEL SAVE

Figure 3-2: Configure Ingress - CIP

Configuration Settings - CIPIngress_1

Select Source

CIP Device Shortcut (Path) * : ra-clx://cgp-cip-adapter/192.168.1.124/

Tag Name * : cipTag1

Read Option * : Continuous Read

Tracking ID : Please enter value

Update Frequency (ms) * : 1000

Update Type * : Polling

Pipe Path * : Filestore_1_path-505

CANCEL SAVE

Figure 3-3: Configure Ingress - SQL

Configuration Settings - SQLIngress_1

Select Source

Stored Procedure * : GetName

Read Option * : Continuous Read

Domain & path of SQL Database server * : ra-sql://cgp-sql-adapter

Tracking ID : Please enter value

Update Frequency (ms) * : 1000

Update Type * : Polling

Tag Request Config * : Tags* : realRandom

CANCEL SAVE

Figure 3-4: Configure Ingress - SQL (Continued)

Configuration Settings - SQLIngress_1

Select Source

Tag Request Config * : Tags* : realRandom

Mapping * : id * : tag
v * : tagvalue
q * : status
t * : timestamp

Pipe Path * : Filestore_1_path-161

CANCEL SAVE

Figure 3-5: Configure Ingress - MQTT

Configuration Settings - mqtingress_1

Select Source

Use SSL * : ☒ true ☐ false

Import Certificate file * : cert_server.crt

Import Key file * : key_server.key

Import Certificate Authority file : ca_ca.crt

Reject Unauthorized * : ☒ true ☐ false

CANCEL SAVE

Figure 3-6: Configure Ingress - MQTT (Continued)

Configuration Settings - mqtingress_1

Select Source

Clean Session * : ☒ true ☐ false

Client ID : Please enter value

Quality of Service * : At most once

Host Name * : qadocindw16vm01.ra-int.cc

Port * : 8883

CANCEL SAVE

Figure 3-7: Configure Ingress - MQTT (Continued)

Configuration Settings - mqtingress_1

Select Source

Topic Name * : devicedata

Read Option * : Continuous Read

Tracking ID : Please enter value

Pipe Path * : datageneratorfilestore_1_pa

CANCEL SAVE

Figure 3-8: Configure Ingress - DataSimulator

Configuration Settings - DataSimulator_1

Select Source

Tag Name * : FloatTag

Read Option * : Continuous Read

Tracking ID : track-sample-pipeline-float

Data Type * : Float

Precision * : 2

Select Range * : 23.1 47.2

Update Frequency (ms) * : 1000

CANCEL SAVE

Figure 3-9: Configure Ingress - DataSimulator (Continued)

Configuration Settings - DataSimulator_1

Select Source

Update Type * : Polling

Pipe Path * : datageneratorfilestore_1_pa

Tag Name * : IntTag

Read Option * : Continuous Read

Tracking ID : track-sample-pipeline-int

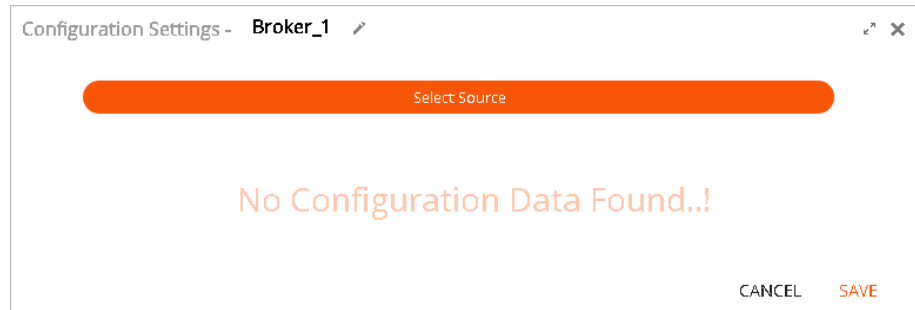
CANCEL SAVE

Broker

The Broker provides optimization of requests going down to Ingress Adapters as well as delivering requested data to requesting applications.

1. Click and drag the Broker onto the Visual Designer.
2. Right-click the Broker to configure. Click [SAVE] to enable the Broker.

NOTE: Currently, Broker has no configuration fields.

Figure 3-10: Configure Broker

Transaction

The Transaction is a gateway extension that allows applications to send transaction requests (actions). This extension acts as both an application and adapter on the gateway, allowing requests in the form of a trigger, an event to be processed and returned to requesting applications.

The output of the Transaction is always sent to the Broker. The combination of Ingress Channels and Transaction Processor is called a Transaction feature. Both Pipe Path and Configuration Parameters will be read from the Ingress Channel.

The Transaction will egress both trigger and event data when the condition occurs. The trigger will only happen if the Read option setting of the trigger tag is Continuous Read.

NOTE: A single Ingress cannot connect to multiple transaction components at same time.
Currently, DataSimulator does not support Transaction.

Following are the available trigger conditions:

- **All:** Move trigger and event data for all trigger data.
- **High:** Move trigger and event data when trigger tag is non zero.
- **Low:** Move trigger and event data when trigger tag is zero.
- **On Change:**
 - Threshold value: Move trigger and event data when difference between current and previous trigger tag value is greater than or equal to the threshold value.
 - Threshold value empty: If the threshold value is set to empty, then the condition will check for onChange Pos and onChange Neg condition. Data will egress if any change in the value and previous value (increasing trend) is greater than or equal to onChangePos value. Also Data will egress if any change in the value and previous value (decreasing trend) is greater than or equal to the onChange Neg value.

- OnChange Positive/Negative threshold value empty: Move trigger and event data when any change in current and previous trigger tag value.
- **Increasing**: Data will egress, if the value and previous value change (increasing trend) is greater than the threshold value.
- **Decreasing**: Data will egress, if the value and previous value change (decreasing trend) is greater than the threshold value.
- **Less Than**: Move trigger and event data when trigger tag value is less than the threshold value.
- **Less Than Equal**: Move trigger and event data when trigger tag value is less than equal to the threshold value.
- **Equal**: Move trigger and event data when trigger tag value is equal to the threshold value.
- **Not Equal**: Move trigger and event data when trigger tag value is not equal to the threshold value.
- **Greater Than**: Move trigger and event data when trigger tag value is greater than the threshold value.
- **Greater Than Equal**: Move trigger and event data when trigger tag value is greater than equal to the threshold value.

Event option provides all the tags from Ingress that users may select. It provides multiple options that allow the User to choose more tags in an Event.

NOTE: Only a single event tag can be selected for MQTT and SQL transaction.

The following constraints apply for the Transaction:

- Set Send Immediate to false won't work for SQL adapter tags.
- Trigger conditions Equal and Not Equal will work only if the value datatype is numeric.
- Caching will apply only when both trigger and event Read option settings are Continuous Read. If any of the events has Read option as Async Read, out of multiple event tag, then the Caching won't be applicable. Caching will be applicable only to CIP and FTLD Adapters.
- Transaction internally uses caching when all the tags read option is set to Continuous Read. Caching is not applicable to SQL adapters. Any tag set to Continuous Read in event section will egress once the trigger condition met. This cannot be stopped. This is the behavior of the transaction. Caching will egress the trigger and event data together. No matter whether the Send Immediate is true or not.
- The On Change trigger can be set using either the threshold parameter or the onChangePos and onChangeNeg parameters. If the threshold value is provided, On Change will be determined as absolute change from previous

value greater than equal to the threshold. If both onChangePos and onChangeNeg values are provided, On Change will be determined in either direction using these separate values. If no parameter is provided, the trigger will evaluate to true due to any change from the previous value.

Configuration of Transaction with MQTT Ingress

Follow the guidelines below to know which the data types of tag are appropriate for which transaction trigger type.

The default data type of MQTT is string or buffer. So a string value always performs against the condition as below:

- High - Any value other than empty string will satisfy this condition.
- Low - Only empty string will satisfy this condition.
- Equal - Never satisfy because it uses tibble == check. Since the setpointValue is always an integer. So the condition fails always. Also, when the setpointValue: "12" (string) in JSON-LD and deployed directly, then a message is published as 12 from MQTT. It is the satisfied condition and egresses data.
- Not Equal - Always satisfy. Because it is an integer (setpointValue) - string (triggerValue) comparison.

Table 3-1 MQTT Input: Trigger Value Behavior

MQTT Input: Trigger Value Examples	High	Low	Equal	Not Equal
"12", 12, '12', 0, -12	TRUE	FALSE	FALSE	TRUE
"", undefined	FALSE	TRUE	FALSE	TRUE

Perform the following steps to configure Transaction:

1. Right-click the Transaction object and it will show the static form which displays Trigger options and lists the available Ingress tags.

NOTE: When the Read Option of ingress tag is Async Read, the tag will not display in the Trigger tag drop-down list.

2. Define the following properties and click [SAVE].

Field	Description
Trigger Data	Select the Trigger data (Ingress). Select Tag: Select the trigger tag from the dropdown list
Trigger Condition	Select the condition to accept the data.
	Condition: Trigger condition to move the Trigger and Event data when Trigger Tag data meets the condition.
	Threshold Value: Threshold value for trigger conditions. All trigger conditions should have a threshold value, except the trigger conditions All, High, Low.
	Initialization Value: Initial value for evaluation of OnChange, Increasing and Decreasing trigger conditions. Initial value is considered only when previous trigger value does not exist.
	Send Immediate Yes: The trigger tag data and event tag data sent as they come in. No: The trigger tag data and the event tag data collected as a single array and sent to Pipeline.
	Ignore Initial Value Yes: The first value of trigger tag data is ignored for trigger condition evaluation. No: The first value of trigger tag data is considered for the trigger condition evaluation
	Egress Type: Set this field to indicate that trigger condition should be triggered on a true state rather than a transition from false to true. E.g.: If Egress type is set to Continuous, and the trigger condition is increasing (if the last value is greater than the second last value and if the current value is less than the last value, it will hold true). If Egress type is set to One Time, (If last value is greater than the second last value and the last value), it will not hold true. Egress Type option is not available for 'On Change' trigger condition.
Events	Select the Event. Select Tags: Select the Event tags

Figure 3-11: Configure Transaction

Configuration Settings - Transaction_1

Select Source

Trigger Data

Select Tag * : devicedata

Trigger Condition

Condition * : Greater Than

Threshold Value * : 10

Send Immediate * : ☒ Yes ☐ No

Ignore Initial Value * : ☒ Yes ☐ No

CANCEL SAVE

Figure 3-12: Configure Transaction (Continued)

Configuration Settings - Transaction_1

Select Source

Threshold Value * : 10

Send Immediate * : ☒ Yes ☐ No

Ignore Initial Value * : ☒ Yes ☐ No

Egress Type * : Continuous

Events

Select Tags * : temperature

CANCEL SAVE

This page has been intentionally left blank



Chapter

4

Configure Processors

In this chapter:

- ❑ Processors 58

Processors

Processors are the built-in operators for processing the data by performing various transformations and analytical operations.

- ☐ “Custom Processor”
 - JavaScript
 - Python
 - Python ML
 - Java
 - PMML
- ☐ “Compression”
- ☐ “Decompression”
- ☐ “BSON Serialize”
- ☐ “BSON DeSerialize”
- ☐ “FileStore”
- ☐ “Delay”
- ☐ “Scheduler”
- ☐ “Remove”
- ☐ “Group Tags”
- ☐ “Thinning Exception”
- ☐ “Thinning Compression”
- ☐ “Math Function (Calculation)”
- ☐ “SqliteStore”
- ☐ “Writeback”

Custom Processor

The Custom Processor is a feature that allows the User to create their own rules. Custom Processors that are registered can be used in building a Pipeline.

Python

NodePython is the sample Python Custom Processor supplied with the application. The sample files are available in the **<Install_Directory>\samples\python** directory.

This module supports custom python script(s) uploaded by the user to be used within a data Pipeline. The NodePython is a node module used for binding python libraries.

1. Click and drag the NodePython Processor onto the Visual Designer and right-click the Processor to configure.
2. Define the following properties and click [SAVE].

Field	Description
fileName	Name of the Python file.
tags	Array of tags (e.g., Tag1,Tag2,Tag3).
outputTag	Name of the new output tag.
passAll	Boolean (will pass all incoming tags when set to true, by default this option is true).

Figure 4-1: Configure Custom Processor - NodePython

Configuration Settings - nodePython_1

Select Source

fileName : pythonAdditic

outputTag : PredictedTag

tags : Pressure,Tem

passAll : true

CANCEL SAVE

PythonML

Sample files for Python ML Custom Processor are supplied with the application. The sample files are available in the <Install_Directory>\samples\ml and <Install_Directory>\samples\pmml directory.

There are two methods to configure the custom processor. Select either the Default method or the Custom method. Refer to FactoryTalk Analytics Edge Administration Guide.

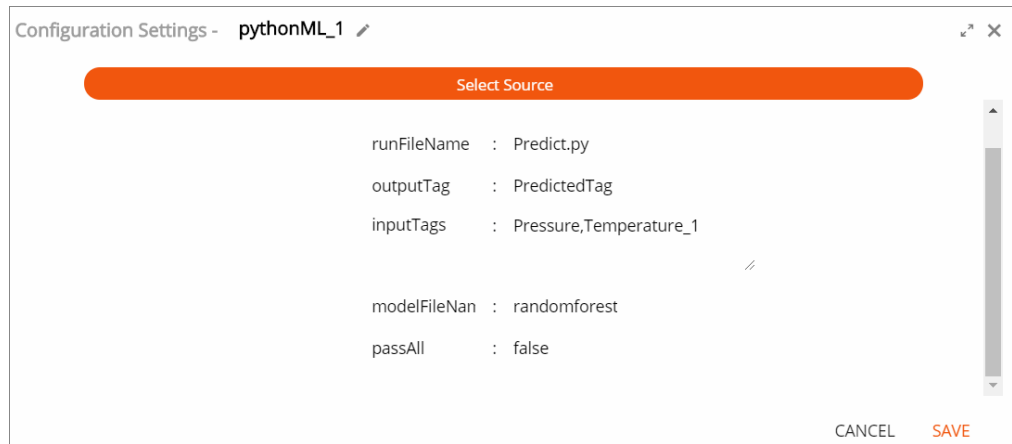
This module allows user to upload and run a pre-trained ML model as part of a data Pipeline. The PythonML module is used for Machine learning in Node Js. The preprocessor.js files are used to perform any pre-process operation.

1. Click and drag the PythonML Processor onto the Visual Designer and right-click the Processor to configure.
2. Define the following properties and click [SAVE].

Field	Description
runFileName	Name of the Python file.

Field	Description
modelFileName	Name of the prediction model file (e.g., randomforest.pkl)
inputTags	Array of tags (e.g., Tag1,Tag2,Tag3).
outputTag	Name of the new output tag.
passAll	Boolean. When set to true, passes all the incoming tags. By default this option is set to true.

Figure 4-2: Configure Custom Processor - NodeML



Java - Class

NodeH2oClass is the sample Java Custom Processor supplied with the application. The sample files are available in the <Install_Directory>\samples\h2o directory.

The NodeH2oClass node module is used for data modeling and general computing. This provides access to the H2O JVM (and extensions thereof). It uses the machine-learning algorithms, and modeling support (basic munging and feature generation).

1. Click and drag the NodeH2oClass Processor onto the Visual Designer and right-click the Processor to configure.
2. Define the following properties and click [SAVE].

Field	Description
tags	Array of tags (e.g., Tag1,Tag2,Tag3).
className	Name of the JAVA class.

Field	Description
outputTag	Name of the new output tag.
passAll	Boolean (will pass all incoming tags when set to true, by default this option is true).

Figure 4-3: Configure Custom Processor - NodeH2oClass

Configuration Settings - nodeH2oClass_1

Select Source

tags : Tag1,Tag2,Tag

className : add

outputTag : PredictedTag

passAll : true

CANCEL SAVE

Java - jar

NodeH2oJar is the sample Java Custom Processor supplied with the application. The sample files are available in the <Install_Directory>\samples\h2o directory.

1. Click and drag the NodeH2oJar Processor onto the Visual Designer and right-click the Processor to configure.
2. Define the following properties and click [SAVE].

Field	Description
jarName	Name of the jar file (e.g., abc.jar).
className	Name of the JAVA class.
outputTag	Name of the new output tag.
passAll	Boolean (will pass all incoming tags when set to true, by default this option is true).
tags	Array of tags (e.g., Tag1,Tag2,Tag3).

Figure 4-4: Configure Custom Processor - NodeH2oJar

Configuration Settings - nodeH2oJar_1

Select Source

jarName : addition.jar

className : add

outputTag : PredictedTag

tags : Tag1,Tag2,Tag

passAll : true

CANCEL SAVE

JavaScript

1. Click and drag the JavaScript Processor onto the Visual Designer and right-click the Processor to configure.
2. Define the properties and click [SAVE].

PMML

This module allows user to upload and run a pre-trained ThingWorx model as part of a data Pipeline.

1. Click and drag the PMML Processor onto the Visual Designer and right-click the Processor to configure.
2. Define the following properties and click [SAVE].

Field	Description
port	A unique port number.
inputTags	Array of tags (e.g., Tag1,Tag2,Tag3).
outputTag	Name of the new output tag.

Figure 4-5: Configure Custom Processor - PMML

Configuration Settings - predictwithpmml_1

Select Source

port : 2553

inputTags : Last_Name,DateTime,1WeekBefore,2WeekBefore,3WeekBefore,PreviousHour

outputTag : predictattend

CANCEL SAVE

Compression

Select this Processor to compress the incoming data using different compression algorithms (gzip or deflate).

1. Click and drag the Compression Processor onto the Visual Designer and right-click the Processor to configure.
2. Define the following properties and click [SAVE].

Field	Description
Output Type	Select the output type (buffer or string) from the dropdown list.
Compression Algorithm	Select the compression algorithm from the dropdown list. Available algorithms are gzip and deflate.

Figure 4-6: Configure Processor - Compress

Configuration Settings - Compression_1

Select Source

Output Type * : string

Compression Algorithm * : gzip

CANCEL SAVE

Decompression

Select this Processor to decompress the incoming compressed data based on the decompression algorithm.

1. Click and drag the Decompression Processor onto the Visual Designer and right-click the Processor to configure.
2. Define the following properties and click [SAVE].

Field	Description
Output Type	Select the same Output Type as in the Compression Processors. Select [string], if Compression Processor is set as string. Select [buffer], if Compression Processor is set as buffer.
Decompression Algorithm	Select the same Compression Algorithm as in the Compression Processors. Decompression Algorithm: gzip Decompression Algorithm: deflate

Figure 4-7: Configure Processor - Decompression

Configuration Settings - Decompression_1

Select Source

Output Type * : buffer

Decompression Algorithm * : deflate

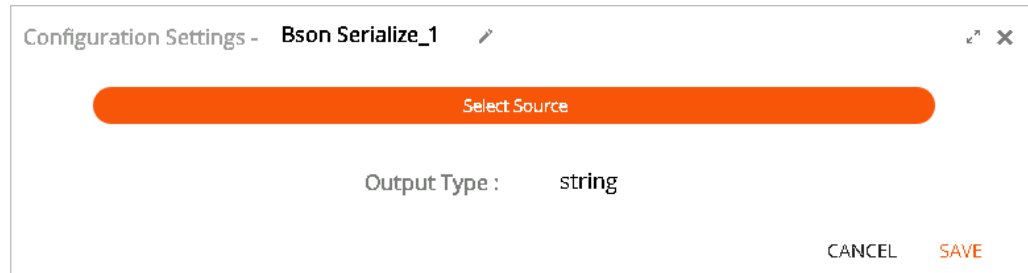
CANCEL SAVE

BSON Serialize

Select this Processor to convert data from JSON to BSON.

1. Click and drag the BSON Serialize Processor onto the Visual Designer and right-click the Processor to configure.
2. Define the following property and click [SAVE].

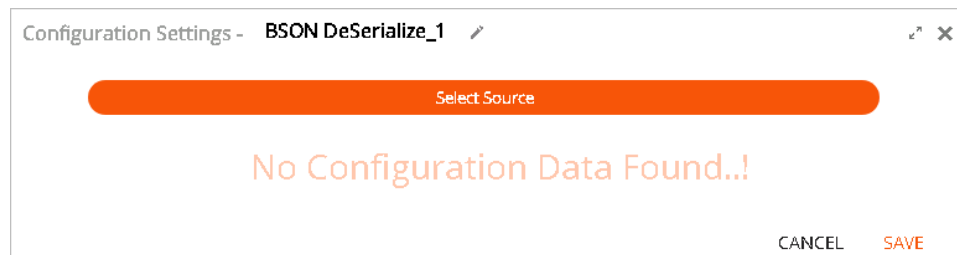
Field	Description
Output Type	Serialize to BSON as string.

Figure 4-8: Configure Processor - BSON Serialize

BSON DeSerialize

Select this Processor to convert data from BSON to JSON.

1. Click and drag the BSON DeSerialize Processor onto the Visual Designer.
2. Right-click the Processor to configure and click [SAVE].

Figure 4-9: Configure Processor - BSON DeSerialize

FileStore

Select this processor to create a transient store-forward mechanism on the runtime node.

The Scheduler or Delay forward processor, added after the FileStore processor, enables batching the data based on Time Interval (File fetch interval) and Fetch Count (Filecount or the number of records).

As a best practice, it is recommended to remove records as and when the records are forwarded, by adding the “**Remove**” processor, so that:

- The Filestore will not exhaust its maximum storage limitation
- Stale data will not exist on the Filestore

The Remove processor removes the data when egress is successful.

NOTE: If the data is not removed from the Filestore, the same data will stay forever, and when the outbound read sequence is set to fetch FIFO, it will always fetch the same data over and over again.

1. Click and drag the FileStore Processor onto the Visual Designer and right-click the Processor to configure.
2. Define the following properties and click [SAVE].

Field	Description
Store Path	Path where file will be stored. If not set will use the home directory of deployment. Example: C:\ShellApps\
Folder Name	Folder Name created in Store Path. Example: Folder1
Maximum Folder Size (Bytes)	Set maximum size of data in folder. Rollover will occur once the limit is reached.
File Extension Name	File extension for the stored data. Example: log or txt or json
Outbound Read Sequence	The next Processor will read file in selected order (FIFO/LIFO), if multiple files are read concurrently. FIFO: First In First Out LIFO: Last In First Out

Figure 4-10: Configure Processor - FileStore

Configuration Settings - FileStore_1

SF-Filestore

Store Path : C:\ShellApp

Folder Name * : tempstore

Maximum Folder Size (Bytes) * : 10000000

File Extension Name * : json

Outbound Read Sequence : FIFO

CANCEL SAVE

Delay

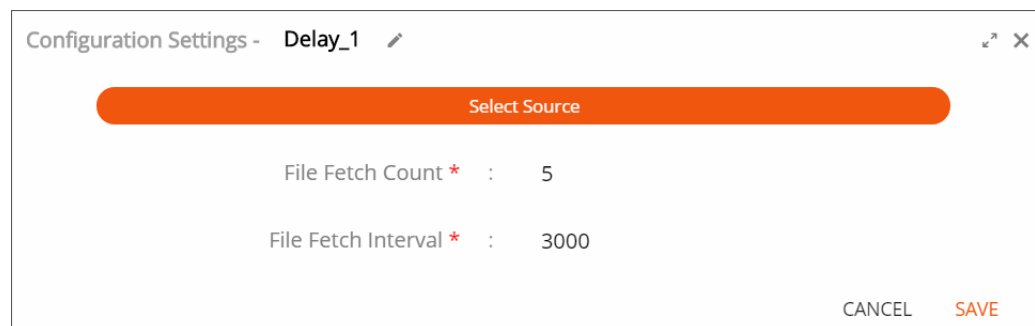
Select this Processor to fetch data and send it to the other Pipeline parts from one of the endpoints every time only when the configured interval time is expired.

NOTE: FileStore or SqliteStore Processor needs to be used before the Delay block.

1. Click and drag the Delay Processor onto the Visual Designer and right-click the Processor to configure.
2. Define the following properties and click [SAVE].

Field	Description
File Fetch Count	Define the number of files to read in each fetch. Example: if set to 3, then 3 files are read on each fetch.
File Fetch Interval	Define fetch frequency.

Figure 4-11: Configure Processor - Delay



Scheduler

Use this Processor to fetch data after every scheduled interval, unlike the Delay Processor, which holds the data and sends it after the intervalMs time is expired.

NOTE: FileStore or SqliteStore Processor needs to be used before the Scheduler block.

1. Click and drag the Scheduler Processor onto the Visual Designer and right-click the Processor to configure.

2. Define the following properties and click [SAVE].

Field	Description
Fetch Count	Define the number of files to read in each fetch. Example: if set to 3, then 3 files are read on each fetch
File Fetch Interval	Define fetch frequency.

Figure 4-12: Configure Processor - Scheduler

Configuration Settings - Scheduler_1

Select Source

Fetch Count * : 5

Fetch Interval (ms) * : 3000

CANCEL SAVE

Remove

Select this Processor to remove data from store endpoint when data is emitted successfully.

NOTE: FileStore or SqliteStore Processor needs to be used before the Remove block.
Delay or Scheduler also needs to be used before the Remove block.

1. Drag the Remove Processor onto the Visual Designer and right-click on the Processor.
2. After connecting each component and configuring the FileStore Processor, the name of the FileStore will be configured in Remove processor automatically. Click [Save] to finish the configuration.

Field	Description
Name of Filestore to remove from	Name of the Store and Forward Filestore to remove files from.

Figure 4-13: Configure Processor - Remove

Configuration Settings - Remove_1

Select Source

Name of FileStore to remove from : FileStore_1

CANCEL SAVE

Secondary Egress

Secondary Emitter can be connected to the Remove processor to support one of the following configurations:

- a. Move: Move data to both the primary and secondary endpoint.
- b. Move on Error: Move data to secondary if an error occurs while egressing data to the primary. If the error is resolved, then it will start moving data to the primary endpoint only.
- c. Delete on Error: If an error occurs at the primary block, then it will delete the data from the store (temporary) block. Data is not egressed to the primary or secondary block until the error is resolved.

To enable the secondary egress block, configure the Pipeline in the order below.

- Both Egress (Primary and Secondary)
- Remove processor
- Other processors
- Ingress

NOTE: Secondary Emitter selection in the configuration of Remove processor is displayed only after both the primary and secondary Emitters are configured and saved.

Figure 4-14: Configure Processor - Secondary Egress on Remove

Configuration Settings - Remove_1

Select Source

Secondary Endpoint : * Filestore_2 ▼

Move : ☐ Yes

Move on Error : ☒ Yes

Delete on Error : ☐ Yes

CANCEL SAVE

TIP: While editing a Pipeline:

If the primary egress is deleted, the pipe info on the Ingress channel automatically changes to show the secondary egress as the new egress.

If any of the egress blocks are deleted, the Remove config options do not show the secondary move options (Move, Move on Error, Delete on Error).

If a second egress block is added again, options for secondary egress are shown again.

Group Tags

Use this Processor to group a set of different tags and send them as one response. For example if you want to read 3 tags on an OnChange condition (Ingress), group the 3 tags. On a change of any tag, the current data of all 3 tags will be sent to end point.

Group Tags processor does not allow the tags which are not defined in it. For example there are three tags in Ingress: TagA, TagB, and TagC and defined tags in the Group Tags processor are TagA and TagB then it will only send the TagA and TagB and will not allow TagC go further.

1. Click and drag the Group Tags Processor onto the Visual Designer and right-click the Processor to configure.

2. Define the following properties and click [SAVE].

Field	Description
Tags to be Grouped	Tag Names separated by comma. Example: <Tag1>,<Tag2>,<Tag3>,<Tag4>
Send Immediate	<p>Define the Tags grouping behavior:</p> <p>False: Send the group of tags as a set when all the tags in the set have an updated value.</p> <ul style="list-style-type: none"> If Tags are configured as polling then they are considered updated based on polling frequency. If tags are configured as Onchange, then they are considered updated only when value of tags have changed at source. <p>True: Send the group of tags as a set when any one tag in the set have an updated value.</p> <ul style="list-style-type: none"> If Tags are configured as polling then they are considered updated based on polling frequency. If tags are configured as on change, then they are considered updated only when value of tags have changed at source.

Figure 4-15: Configure Processor - Group Tags

Configuration Settings - Group Tags_1

Select Source

Tags to be Grouped * : Tag1,Tag2

Send Immediate : ☒ true ☐ false

CANCEL SAVE

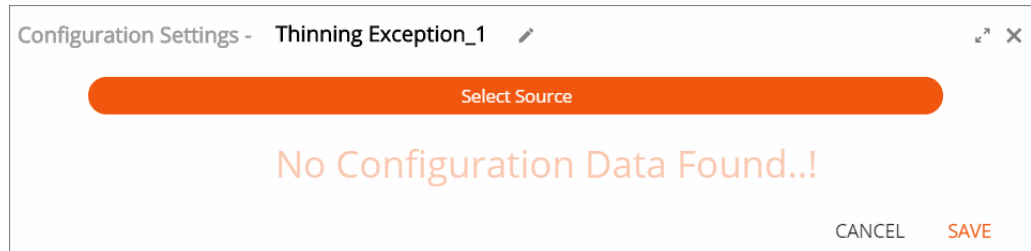
NOTE: Group Tags processor does not push the tags which are not defined in it.

Thinning Exception

Select this Processor to thin out all the data and capture information if it goes about a particular deviation value (exceptDev) in particular time interval (timeLimit).

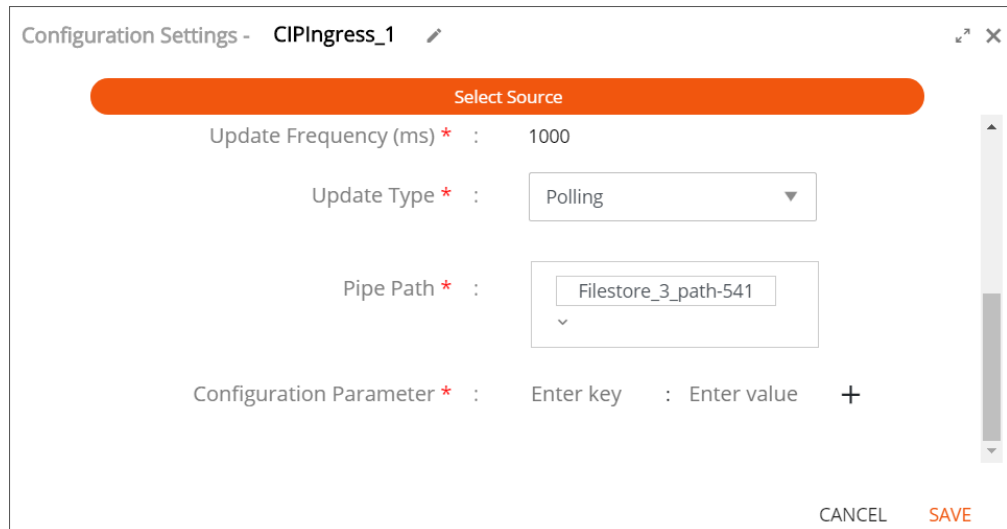
1. Click and drag the Thinning Processor onto the Visual Designer and right-click the Processor to configure.
2. Click [SAVE] to enable Thinning Exception.

Figure 4-16: Configure Processor - Thinning Exception



After, all the components in the Pipeline are connected, right-click the Ingress Channel and select the Pipe Path. Configuration Parameter label displays.

Figure 4-17: Select Pipe Path



Define the following Thinning Exception properties and click [SAVE].

Field	Description
exceptDev	Enter the Exception Deviation Limit parameter (exceptDev) in the Tag Subscription Config of the adapter. <u>NOTE:</u> Parameter is case sensitive Ex: exceptDev : 5. Enter “exceptDev” in the “Enter Key” field and the exception deviation value in the “Enter Value” field.
timeLimit	Enter the time Limit parameter (timeLimit) a in the Tag Subscription Config of the adapter. <u>NOTE:</u> Parameter is case sensitive. Ex: timeLimit : 10. Enter “timeLimit” in the “Enter Key” field and the time limit value in the “Enter Value” field. <u>NOTE:</u> The timeLimit is measured in milliseconds.

NOTE: Configure both the parameters (exceptDev and timeLimit). These parameters cannot be used alone.

Figure 4-18: Thinning Exception Properties

Configuration Settings - CIPIngress_1

Select Source

Update Frequency (ms) * : 1000

Update Type * : Polling

Pipe Path * : Filestore_3_path-541

Configuration Parameter * :

exceptDev	: 10
timeLimit	: 2000

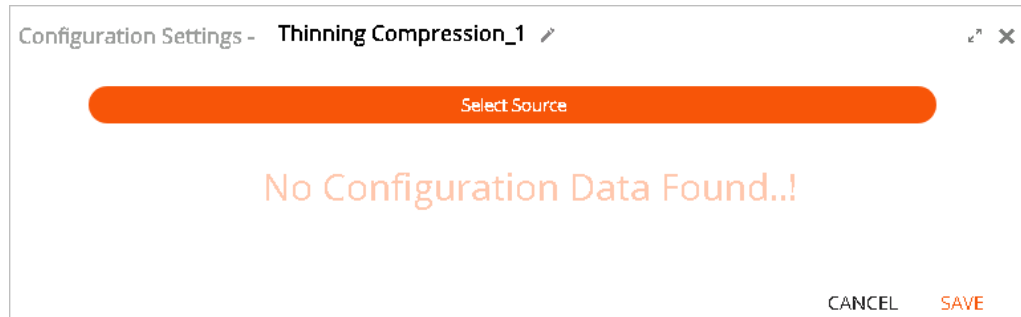
CANCEL SAVE

Thinning Compression

Select this Processor to thin out all the data using revolving door algorithm to remove unnecessary data and provide the data values that cross the deviation value (compDev) in particular time (timeLimit).

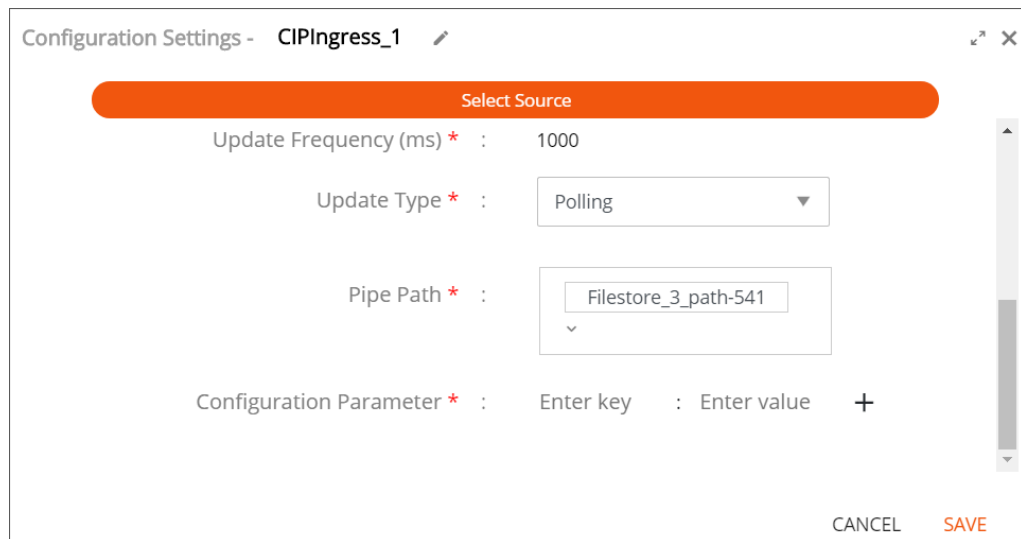
1. Click and drag the Thinning Compression Processor onto the Visual Designer and right-click the Processor to configure.
2. Click [SAVE] to enable Thinning Compression.

Figure 4-19: Configure Processor - Thinning Compression



3. After, all the components in the Pipeline are connected, right-click the Ingress Channel and select the Pipe Path. Configuration Parameter label displays.

Figure 4-20: Select Pipe Path



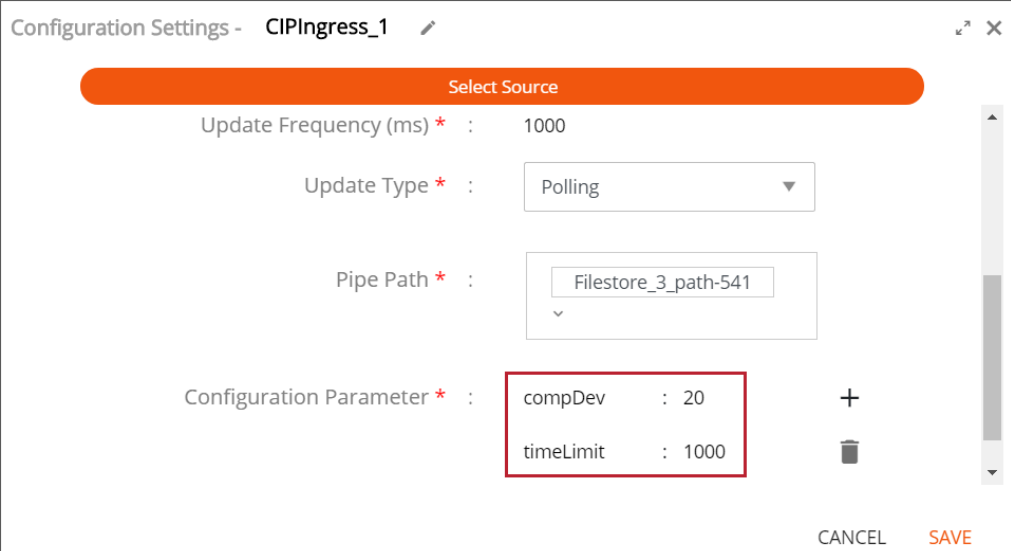
4. Click the [+] icon to add more Parameters.

5. Define the following Thinning Compression properties and click [SAVE].

Field	Description
compDev	Enter the Compression Deviation Limit parameter (compDev) in the Tag Subscription Config of the adapter. <u>NOTE:</u> Parameter is case sensitive. Ex: compDev :10 Enter “compDev” in the “Enter Key” field and the compression deviation value in the “Enter Value” field.
timeLimit	Enter the time Limit parameter (timeLimit) a in the Tag Subscription Config of the adapter. <u>NOTE:</u> Parameter is case sensitive. Ex: timeLimit: 10. Enter “timeLimit” in the “Enter Key” field and the time limit value in the “Enter Value” field. <u>NOTE:</u> The timeLimit is measured in milliseconds.

NOTE: Configure both the parameters (compDev and timeLimit). These parameters cannot be used alone.

Figure 4-21: Thinning Compression Properties



Configuration Settings - CIPIngress_1

Select Source

Update Frequency (ms) * : 1000

Update Type * : Polling

Pipe Path * : Filestore_3_path-541

Configuration Parameter * :

- compDev : 20
- timeLimit : 1000

CANCEL SAVE

Math Function (Calculation)

The Math function (Calculation) processor allows users to create calculation expressions while building Pipeline using tags. The Inputs are defined by Ingress tags and the outputs are defined by the Ingress tags and a Calculation Tag which is the Calculation value.

- Users can select the tags according to the ingresses that connect to the component.
 - Users can create the expression using the following operators:
 - Sum (+)
 - Difference (-)
 - Product (*)
 - Quotient (/)
 - Percentage (%)
1. Click and drag the Calculation Processor onto the Visual Designer and right click the Processor to configure.
 2. Define the following properties and click [SAVE].

Field	Description
Expression	The expression for the calculation.
Tags	The Tags that users selected for expression.
Output Tag Name	Name of the output Calculation Tag.

Figure 4-22: Configure - Calculation

Configuration Settings - Calculation_2

Expression		Tags							
Expression: <input type="text" value="(#{Tag1})+#{Tag2})*5"/>		Output Tag Name: <input type="text" value="CalculationTag"/>							
Arithmetic: +, -, *, /, % Logical: (,)		Search: <input type="text"/>							
Numeric keypad: 1, 2, 3, 4, 5, 6, 7, 8, 9, C, 0, .		<table border="1"> <thead> <tr> <th>Tag ID</th> <th>Tag Path</th> </tr> </thead> <tbody> <tr> <td>Tag1</td> <td>ra-ftld://cgp-ftld/RNA://\$Global/TestApp/TestArea:: [110Slot0]</td> </tr> <tr> <td>Tag2</td> <td>ra-ftld://cgp-ftld/RNA://\$Global/TestApp/TestArea:: [110Slot0]</td> </tr> </tbody> </table>		Tag ID	Tag Path	Tag1	ra-ftld://cgp-ftld/RNA://\$Global/TestApp/TestArea:: [110Slot0]	Tag2	ra-ftld://cgp-ftld/RNA://\$Global/TestApp/TestArea:: [110Slot0]
Tag ID	Tag Path								
Tag1	ra-ftld://cgp-ftld/RNA://\$Global/TestApp/TestArea:: [110Slot0]								
Tag2	ra-ftld://cgp-ftld/RNA://\$Global/TestApp/TestArea:: [110Slot0]								
		Showing 1 to 2 of 2 entries							
		Previous 1 Next							
		CANCEL SAVE							

SqliteStore

SqliteStore Processor can be used when an SQLite Database is required as a store-forward mechanism. SQLite database must be installed on the Runtime node.

The SqliteStore processor does not forward the records automatically. The Processor needs a forward block right after it to fetch records to be forwarded to the next processing block. Also, as a best practice, it is recommended to remove records as and when the record(s) are forwarded, by adding the Remove processor so that:

- The SQLite database will not exhaust its maximum storage limitation
- Stale data will not exist on the SQLite database

Use case 1:

Ingress -> Broker -> SqliteStore -> Endpoint

When the SqliteStore is used alone, data is not egressed to the endpoint. Data is persistent in the SQLite store.

Use case 2:

Ingress -> Broker -> SqliteStore -> Forward (Delay) -> Endpoint

Add the Delay Processor after SqliteStore to fetch data and send it to the endpoint only for one-time after the configured interval time is expired.

Use case 3:

Ingress -> Broker -> SqliteStore -> Forward (Scheduler) -> Endpoint

Add the Scheduler Processor after SqliteStore to fetch data only for one-time after the scheduled interval and send it to the endpoint.

Use case 4:

Ingress -> Broker -> SqliteStore -> Forward (Scheduler / Delay) -> Remove -> Endpoint

Add the Remove Processor as shown below to remove data from the SqliteStore when data is emitted successfully.

The Scheduler or Delay forward processor, added after the SqliteStore processor, enables batching the data based on Time Interval (File fetch interval) and Fetch Count (Filecount or the number of records).

To configure the SqliteStore Processor:

1. Click and drag the SqliteStore Processor onto the Visual Designer and right click the Processor to configure.

2. Define the following properties and click [SAVE].

Field	Description
Store Path	Path where file will be stored. If not set then will use the home directory of deployment. Example: C:\ShellApp\
Database Name	Enter the name of the Database.
Table Name	Enter the name of the Table.
Database Max Size (Bytes)	Maximum size for the Table in Database. One Table in each Database is recommended.

Figure 4-23: SqliteStore Processor

Configuration Settings - SqliteStore_1

SF-SqliteStore

Store Path : C:\SQLiteStore

Database Name * : TagData

Table Name * : fromController

Database Max Size(Bytes) * : 10000000

CANCEL SAVE

Writeback

The Writeback Processor writes the value of the defined "Read Tag" (incoming tag) to the defined "Write Tag" in the PLC. Edge supports write back to the PLC through the FTL D and CIP adapters. The value of the "Writeback Shortcut (Path)" option should have a path to FTL D or CIP.

Connection Rules

The Writeback Processor can receive the response from the Ingress. The "mimeTypeIn" and "mimeTypeOut" will be "x-ra/cgp-reaction". This means it can connect as input to any component whose "mimeTypeOut" is "x-ra/cgp-reaction". All the Endpoints and Processors having "mimeTypeIn" as "x-ra/cgp-reaction" can be connected to this component.

The following components cannot be connected before the Writeback component when creating the Pipeline:

Compression, Decompression, BSON Serialized, BSON Deserialized, Filestore, Sqlitestore, Delay, Scheduler, and Remove.

As the output data from the above components is not in the expected input format (“x-ra/cgp-reaction”) of Writeback.

However, the above-mentioned components can connect after the Writeback component in the Pipeline.

NOTE: CGP will not throw any error even if any option(s) are defined incorrectly in the Writeback Processor.
The user needs to make sure that all options are correct.

Data Types

The following write types are currently supported by the toolkit:

SINT, INT, DINT, LINT, REAL, STRING, BOOL

Table 4-1 Data Type Range

Atomic Data Type	Definition	Range
BOOL	1 bit (boolean)	0 (False), 1 (True)
SINT	8 bits (short integer)	-128 to 127
INT	16 bits (signed integer)	-32,768 to 32,767
DINT	32 bits (signed double integer)	-2,147,483,648 to 2,147,483,647
LINT	64 bits (signed long integer)	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
REAL	32 bits (IEEE 754 single precision float)	-3.40282346E+38 to 3.40282346E+38

NOTE: CGP will take the value in the regular form up to the number of 20 digits, however, if the length of the number goes beyond 20 digits then it will convert the number into the scientific notation form.

Whereas RSLogix (PLC) will take the value in the regular form up to number less than 10 digits. If the length of the number goes beyond 9 digits then it will convert the number into the scientific notation form.

The User needs to select the correct data type for the input and output tag. CGP do not know the data type of the tag (from FTLD, CIP, SQL, or MQTT).

The CGP and Adapters don't register the value or length of the data type. CGP and Adapters directly send values to the PLC to write specific tags defined by the Users.

When writing the value to the tag, the PLC (RSLogix Emulator) will check the data type of the tag to which it has to write. If the value entering the PLC from CGP has a greater range compared to the write tag, then it breaks down the value and writes it.

For instance, if the input tag data type is INT (range -32,768 to 32,767) and the write tag data type is SINT (range -128 to 127), and the value of the input tag is 1000, which is much higher than the range of the write tag data type, then RSLogix will write an incorrect value to the input tag.

To overcome this type of error, follow the specified selection of the data type. Based on the selection of the input tag data type, the write tag data type will be displayed.

Refer the following tables for supported data type selection for read and write for FTLD, CIP, SQL, and MQTT:

'True' in the table indicates that the combination can be selected.

Table 4-2 Data Type Selection for FTLD and CIP Writeback - Adapters FTLD or CIP

		Input Tag Data Type						
		SINT	INT	DINT	LINT	REAL	STRING	BOOL
Write back Tag Data Type	SINT	True						
	INT	True	True					
	DINT	True	True	True				
	LINT	True	True	True	True			
	REAL	True	True	True	True	True		
	STRING	True	True	True	True	True	True	
	BOOL							True

Table 4-3 Data Type Selection for FTL D and CIP Writeback - Adapter SQL

		Input Tag Data Type					
		SMALL INT	INT	REAL	CHAR	VAR CHAR	Bit
Write back Tag Data Type	INT	True					
	DINT	True	True				
	LINT	True	True				
	REAL	True	True	True			
	STRING	True	True	True	True	True	
	BOOL						True

Table 4-4 Data Type Selection for FTL D and CIP Writeback - Adapter MQTT

		Input Tag Data Type
		STRING
Writeback Tag Data Type	STRING	True

Configure Writeback

WARNING: The Writeback Processor can modify the source tag values. Be sure to configure the Processor.

1. Click and drag the Writeback Processor onto the Visual Designer and right click the Processor to configure.
2. Define the following fields:

Field	Description
Input Tag Name	The incoming tag name to read data.
Select Adapter	Select the Adapter from the dropdown list.
Input Tag Data Type	Select the data type of the Input Tag.
Writeback To	Select the PLC type to writeback to (FTLD or CIP).
Writeback Tag Name	Enter tag name to Writeback data.

Field	Description
Writeback Shortcut (Path)	<p>Select the Adapter from the dropdown list.</p> <p>To write back through FTL D, enter the Device path in FT Admin Console as given below: Device path example: ra-ftld://cgp-ftld/< Device Shortcut> Device Shortcut example: RNA://\$Global/EAPTest/EAPArea::[Physical_PLC]</p> <p>To write back through CIP, enter the Device path in FT Admin Console as given below: Device path format: ra-dpi://cgp-cip-adapter/<IP of device> /<Backplane Number/Slot> Device shortcut example for a PLC: ra-clx://cgp-cip-adapter/xxx.xxx.xxx.xxx/1:0</p>
Writeback Tag Data Type	Select the data type of the Writeback tag. The dropdown list displays only the supported data types based on the selected Adapter and the Input Tag data type.
Tracking ID	<p>This ID will be available in data response json object as an additional field for tracking.</p> <p>Response object format for Tracking Id: "trackingId":<ID specified></p>

Figure 4-24: Configure - Writeback

Configuration Settings - WritebackProcessor_1

Select Source

Input Tag Name * : realRandom

Select Adapter * : FTL D

Input Tag DataType * : SINT

Writeback To * : CIP

Writeback Tag name * : newTag

Writeback Shortcut (Path) * : ra-clx://cgp-cip-adapter/10.168.10.1

Writeback Tag DataType * : LINT

CANCEL SAVE

WARNING: Whenever User edits the Ingress channel on an already registered Writeback entity, they have to revisit the Writeback configuration and update the same information.

NOTE: If the configured Input Tag Name doesn't match with any of the available Ingress Tags, a warning message is displayed. If you are sure click [OK] or else click [CANCEL] and change the Input Tag Name.

This page has been intentionally left blank



Chapter

5

Configure Emitters

In this chapter:

- Emitters 86

Emitters

Emitters define the destination stage of a Pipeline. The available Emitters are FileStore, MSSQL, SQLite, IOT Hub, Azure Blob and Queue, Kafka, Socket IO, Secure Webservice and MQTT.

Refer to Chapter 2 of FactoryTalk Analytics Edge Administration Guide for registering the Emitters.

To add an Emitter into your Pipeline:


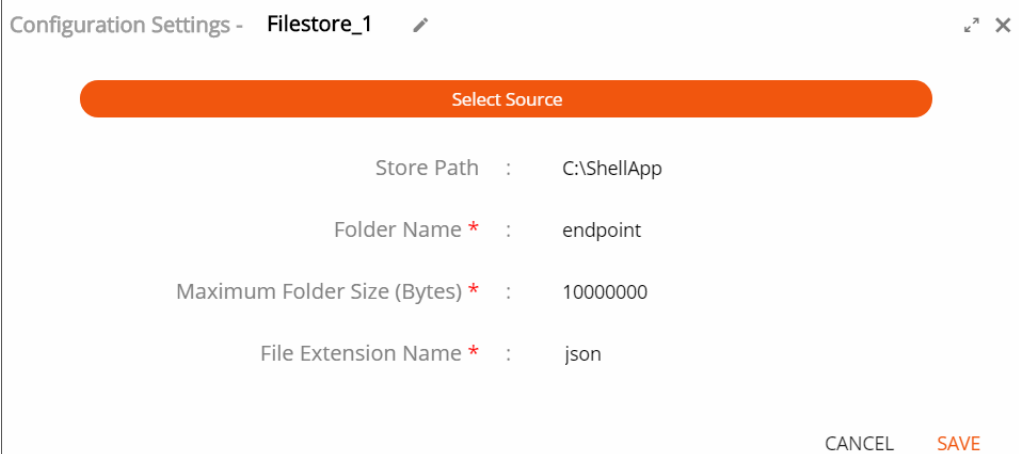
1. Click and drag the Emitter onto the Visual Designer, connect it to a Channel or Processor and right-click the Emitter to configure it.
2. If required, click the [] icon to change the Component Name of the Emitter from the Pipeline canvas. The edited or updated Emitter component name will be reflected in Pipe Path section of Ingress component. The emitter component name will also be reflected in the Secondary Egress section of Remove processor, if available.
3. If required, change the default configuration and click [SAVE].

Figure 5-1: Configure Emitter - FileStore



Configuration Settings - Filestore_1

Select Source

Store Path : C:\ShellApp

Folder Name * : endpoint

Maximum Folder Size (Bytes) * : 10000000

File Extension Name * : json

CANCEL SAVE

Figure 5-2: Configure Emitter - MSSQL

Configuration Settings - mssqlEmitter_1

Select Source

Server Name * : 192.168.10.12

Database Name * : database1

Username * : root

Password * :

Select Destination Type * : ☐ table ☒ storedProcedure

Name * : getName

Mapping Flow * : Tag-->Column/Storec ▼

CANCEL SAVE

Figure 5-3: Configure Emitter - MSSQL (Continued)

Configuration Settings - mssqlEmitter_1

Select Source

Name * : getName

Mapping Flow * : Tag-->Column/Storec ▼

Mapping * : tag0 : tagname1 +

TimeStamp * : Insert ▼

TimeStamp Column or Parameter * : timecolumn

Encrypt Connection * : ☒ true ☐ false

CANCEL SAVE

Figure 5-4: Configure Emitter - SQLite

Configuration Settings - sqliteEmitter_1

Select Source

Store Path : C:\ShellApp

Database Name * : production

Table Name * : fromedge

Database Max Size(Bytes) * : 10000000

CANCEL SAVE

Figure 5-5: Configure Emitter - Azure Blob and Queue

Configuration Settings - azureemitter_1

Select Source

Blob Service Endpoint * : http://rockwellstorage.blob.

Blob Token * : ?sv=2019-03-29&ss=bf

Blob Container Name * : BlobContainer1

To Queue * : ☒ true ☐ false

Queue Service Endpoint * : https://rockwellstorage.que

Queue Token * : ?sv=2018-03-29&ss=bf

Queue Name * : Queue1

CANCEL SAVE

Figure 5-6: Configure Emitter - IOT Hub

Configuration Settings - IoTHubEmitter_1

Select Source

Device ID * : device1

Host Name * : rockwelhub.azure.devices.n

Connection String * : HostName=rockwellhub.azi

CANCEL SAVE

Figure 5-7: Configure Emitter - Kafka

Configuration Settings - kafkaEmitter_1

Select Source

Post Topic Name * : kafkaTopic

Post IP * : 192.168.10.128

Post Port * : 803

Post Partitions * : 1

Use SSL * : ☒ true ☐ false

Import Certificate file * : cert_apinbanisepqa2.ra-...

Import Key file * : key_apinbanisepqa2.ra-i...

CANCEL SAVE

Figure 5-8: Configure Socket IO Source

Configuration Settings - SocketIOEmitter_1

Select Source

Select Source Gateway Server

Room Name * : Room1

Post Event Name * : Event1

CANCEL SAVE

Figure 5-9: Configure Socket IO GatewayServer

Configuration Settings - SocketIOEmitter_1

Select Source

Select Source Gateway Server

Port * : 8080

CANCEL SAVE

Figure 5-10: Configure Emitter - Secure Webservice

Configuration Settings - httpsclient_1 ✎

Select Source

Webserver Host Name/IP Address* : 192.168.10.128

Webserver Port* : 3000

Webserver Path* : /handle

Headers : Content-Type : application/x- +

Reject Unauthorized* : ☒ true ☐ false

Import Certificate file* : cert_server-crt.pem

Import Key file* : key_server-key.pem

CANCEL SAVE

Figure 5-11: Configure Emitter MQTT

Configuration Settings - mqttEmitter_1 ✎

Select Source

Post Host Name : apinbanisepqa2.ra-int.com

Post Port* : 8883

Topic Name* : emitterTopic

Clean Session* : ☒ true ☐ false

Client ID : Please enter value

Quality of Service* : At most once ▼

Use SSL* : ☒ true ☐ false

CANCEL SAVE

Figure 5-12: Configure Emitter MQTT (Continued)

Configuration Settings - mqttEmitter_1

Select Source

Quality of Service * : At most once

Use SSL * : ☒ true ☐ false

Reject Unauthorized * : ☒ true ☐ false

Import Certificate file * : cert_server-crt.pem

Import Key file * : key_server-key.pem

Import Certificate Authority file * : ca_ca.crt

CANCEL SAVE

This page has been intentionally left blank



Chapter

6

JSON Creation

In this chapter:

- ❑ **Sample JSON** 94
- ❑ **Channel Ingress** 94
- ❑ **JavaScript Custom Processor** 103
- ❑ **Import Pipeline** 107

Sample JSON

Channel Ingress

FTLD

Figure 6-1: FTLD Configuration Type - JSON

Channel Ingress : FTLD

Component Name * : FTLD1

Configuration Type : ☐ Fields ☒ Import (Json)

▼ Tag: 1

Enter Action Data: *

```
{
  "id" : "rawdata",
  "actionType" : "0",
  "contextUri" : "ra-ftld://cgp-
ftld/RNA://$Global/TestApp/TestArea::
[110Slot0]",
  "contextPrefix" : "a",
  "trackingId" : "a",
  "updateRateMs" : 1000,
  "updateType" : "0"
}
```

CLEAR REGISTER

```
{
  "id": "Tag1",
  "actionType": 0,
  "contextUri": "ra-ftld://cgp-ftld/RNA://$Global/TestApp/TestArea::[110Slot0]",
  "trackingId": "a",
  "contextPrefix" : "a",
  "updateRateMs": 1000,
  "updateType": 0
}
```


The following table helps in understanding the JSON file in relation with the user interface:

UI Label	JSON
Tag Name	ID
Read Option	actionType
Continuous Read	0
Async Read	4
FTLD Device Shortcut (Path)	contextUri
Tracking ID	trackingId
Update Frequency (ms)	updateRateMs
Update Type	updateType
Polling	0
OnChange	1

CIP

Figure 6-2: CIP Configuration Type - JSON

Channel Ingress : CIP

+

Component Name * : CIP2

Configuration Type : ☐ Fields ☒ Import (Json)

Tag: 1

Enter Action Data: *

```
{
  "id" : "rawdata",
  "actionType" : "0",
  "contextUri" : "ra-clx://cgp-cip-adapter/192.168.1.124/1:2",
  "contextPrefix" : "a",
  "trackingId" : "a",
  "updateRateMs" : 1000,
  "updateType" : "0"
}
```

CLEAR
REGISTER

```

{
  "id": "rawdata",
  "actionType": 0,
  "contextUri": "ra-clx://cgp-cip-adapter/192.168.1.124/1:2",
  "contextPrefix": "a",
  "trackingId": "track-rawdata",
  "updateRateMs": 1000,
  "updateType": 0
}

```

The following table helps in understanding the JSON file in relation with the User Interface:

UI Label	JSON
Tag Name	id
Read Option	actionType
Continuous Read	0
Async Read	4
CIP Device Shortcut (Path)	contextUri
Tracking ID	trackingId
Update Frequency (ms)	updateRateMs
Update Type	updateType
Polling	0
OnChange	1

SQL

Figure 6-3: SQL Configuration Type - JSON

Channel Ingress : SQL

Component Name *

 : SQLChannel

Configuration Type

 :

Fields

Import (json)

Server *

 : 192.168.10.110

Database *

 : plant1data

Username *

 : smith

Password *

 :

▼ Tag: 1

Enter Action Data: *

{
 "contextUri": "ra-sql://sql_test_cgp-sql-
adapter1524690375397",
 "updateRateMs": 1000,
 "updateType": 0,
 "groupActionOptions": {
 "qualityType": "csm",
 "credentialsId": "1",
 "mapping": {
 "q": "status",

CLEAR

REGISTER

```

{
  "contextUri": "ra-sql://sql_test_cgpr-sql-adapter1524690375397",
  "updateRateMs": 1000,
  "updateType": 0,
  "groupActionOptions": {
    "qualityType": "csm",
    "credentialsId": "1",
    "mapping": {
      "q": "status",
      "id": "tag",
      "t": "DataTimeStamp",
      "v": "tagvalue"
    },
    "tags": [
      "Tag1"
    ]
  },
  "actionType": 0,
  "trackingId": "dghfghf",
  "id": "GetHistoricalDataAllTags",
  "groupReactionOptions": {
    "pipe": [
    ]
  }
}

```

The following table helps in understanding the JSON file in relation with the User Interface:

UI Label	JSON
Stored Procedure	id
Read option	actionType
Continuous Read	0
Async Read	4
Domain & path of SQL Database server	contextUri
Tracking ID	trackingId
Update Frequency (ms)	updateRateMs
Update Type	updateType

Polling	0
OnChange	1
Tag Request Config	groupActionOptions
Mapping	mapping
q	q
id	id
t	t
v	v

MQTT

Figure 6-4: MQTT Configuration Type - JSON

Channel Ingress : MQTT

Component Name * :
mqttimportjson

Configuration Type :
☐ Fields
☒ Import (json)

Use SSL * :
☒ true
☐ false

Import Certificate file * :

cert_server.crt

Import Key file * :

key_server.key

Import Certificate Authority file :

ca_ca.crt

Reject Unauthorized * :
☒ true
☐ false

Clean Session * :
☒ true
☐ false

Client ID :
Please enter value

Quality of Service * :

At most once

Host Name * :
qadocindw16vm01.ra-int.

Port * :
8883

Tag: 1

Enter Action Data: *

{
"q": "test1",
"actionType": 0,
"contextUri": "ra-mqtt://cgp-mqtt-adapter/mqtt/qadocindw16vm01.ra-int.com:8883",
"trackingId": "123",
"updateRateMs": 1000,
"updateType": 0,
"groupActionOptions": {}
}

CLEAR
REGISTER

```

{
    "id": "test1",
    "actionType": 0,
    "contextUri":
"ra-mqtt://cgp-mqtt-adapter/mqtt://qadocindw16vm01.ra-int.com:8883",
    "trackingId": "123",
    "updateRateMs": 1000,
    "updateType": 0,
    "groupActionOptions": {},
    "groupReactionOptions": {},
    "itemActionOptions": {
        "connectionString":
"mqtt://qadocindw16vm01.ra-int.com:8883",
        "subscribeOptions": {
            "qos": 0
        },
        "connectionOptions": {
            "useSSL": true,
            "sslOptions": {
                "cert":
"./shared/mqttimportjson/cert_server.crt",
                "key":
"./shared/mqttimportjson/key_server.key",
                "ca": "./shared/mqttimportjson/ca_ca.crt",
                "rejectUnauthorized": true
            },
            "clean": true,
            "clientId": "client123"
        }
    },
    "itemReactionOptions": {}
}

```

The following points help in understanding the JSON file in relation with the user interface:

- id is Topic Name.
- actionType should be given as 0 (zero)
// For MQTT, only action Type 0 is available.
- Tracking id should be given as some value.
- contextUri should be given as "ra.mqtt://cgp-mqtt-adapter/mqtt:// <value which was entered in the Host Name or IP textbox> : < value which was entered in the Port textbox >"

- updateRateMs should be given as some number
// For MQTT, update rate is not required. However, it cannot be passed empty. Data only come in when publisher publishes any data
- updateType should be given as 0
// For MQTT, Pooling or Onchange is not supported. Data only come in when publisher publish any data. However, as it cannot be left empty so default value 0 (zero) should be passed.
- groupActionOptions should be { }
- groupReactionOptions should be { }
- "itemActionOptions": {
 "connectionString": "mqtt://qadocindw16vm01.ra-int.com:8883",
 "publishOptions": {
 "qos": "0"
 },
 "connectionOptions": {
 "useSSL": true,
 "sslOptions": {
 "cert": "./shared/mqttimportjson/cert_server.crt",
 "key": "./shared/mqttimportjson/key_server.key",
 "ca": "./shared/mqttimportjson/ca_ca.crt",
 "rejectUnauthorized": true
 }
 }
}
- ConnectionString should be mqtt://<value which was entered in the IP textbox>:< value which was entered in the Port textbox >
- "qos" should be given as '0'
- useSSL is the option selected in Use SSL radio button
- cert should be "./shared/mqttimportjson<This is the component name>/cert_server.crt<This is the file name which is showed below the upload icon of import Certificate file after uploading>"
- key should be "./shared/mqttimportjson<This is the component name>/key_server.key<This is the file name which is showed below the upload icon of import Key file after uploading >"
- ca should be "./shared/mqttimportjson<This is the component name>/ca_ca.crt<This is the file name which is showed below the upload icon of import Certificate Authority file after uploading >"
- rejectUnauthorized should be the value of Reject Unauthorized radio button.
- itemReactionOptions should be given as { }

DataSimulator

Figure 6-5: DataSimulator Configuration Type - JSON

The screenshot shows the 'Channel Ingress : DataSimulator' configuration window. The 'Component Name' is 'simulatefloat'. The 'Configuration Type' is set to 'Import (json)'. A 'Tag: 1' section is expanded, showing the 'Enter Action Data:' field with a JSON object. The JSON object contains the following fields: 'contextUri', 'updateRateMs', 'itemActionOptions', 'updateType', 'actionType', 'trackingId', and 'id'. At the bottom right, there are 'CLEAR' and 'REGISTER' buttons.

```

{
  "contextUri": "ra-cgp://eap-gen-data-adapter",
  "updateRateMs": 1000,
  "itemActionOptions": {},
  "updateType": 0,
  "actionType": 0,
  "trackingId": "track-sample-pipeline-float",
  "id": "FloatTag;Float-23.1,47.2:2"
}

```

```

{
  "contextUri": "ra-cgp://eap-gen-data-adapter",
  "updateRateMs": 1000,
  "itemActionOptions": {},
  "updateType": 0,
  "actionType": 0,
  "trackingId": "track-sample-pipeline-float",
  "id": "FloatTag;Float-23.1,47.2:2"
}

```

The following table helps in understanding the JSON file in relation with the user interface:

UI Label	JSON
Tag Name	id
Read Option	actionType
Continuous Read	0
Tracking ID	trackingId
Update Frequency (ms)	updateRateMs
Update Type	updateType
Polling	0
OnChange	1

JavaScript Custom Processor

Custom Processor can be added by uploading a .js file.

Figure 6-6: Custom Processor Tab

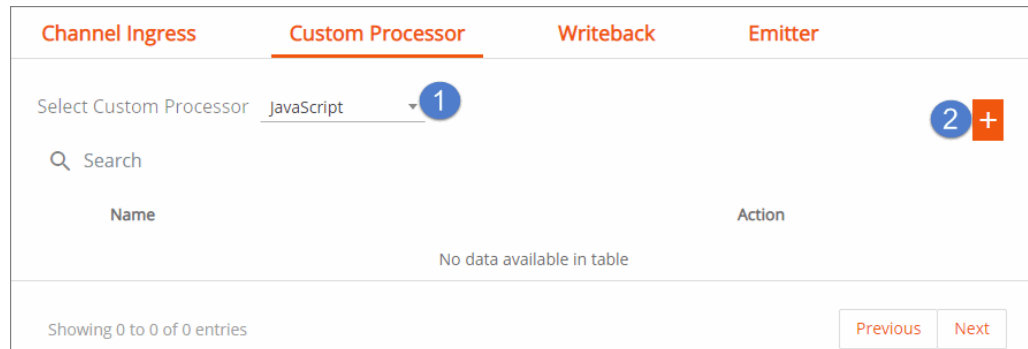
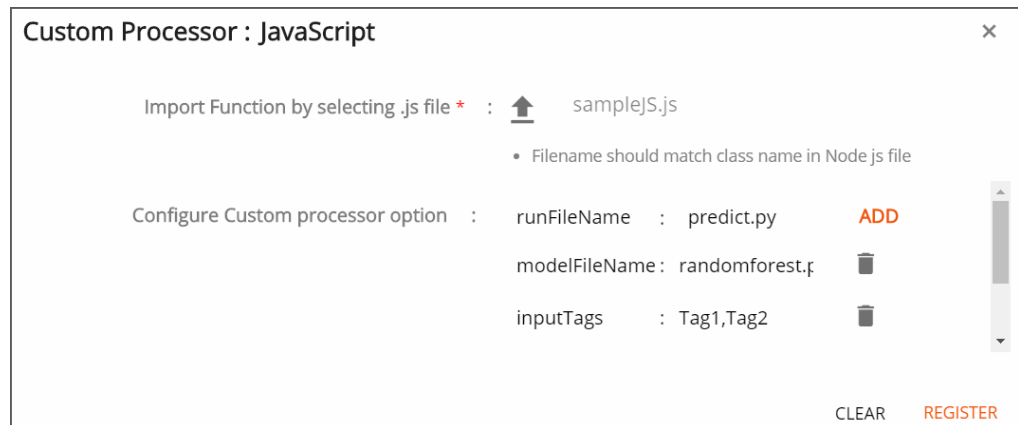


Figure 6-7: Custom Processor



Sample .js File

The following example shows a Custom Processor. Options of the sample are as follow:

```
runFileName: "", // Name of the Python file. // required
modelName: "", // Name of the prediction model file (e.g.,
               randomforest.pkl) // required
inputTags: "", // Array of tags (e.g., Tag1,Tag2,Tag3). // required
outputTag: "", // Name of the new output tag. // required
passAll: "", // Boolean. When set to true, passes all the incoming tags.
              By default this option is set to true.
              // optional
```

```

"use strict";
Object.defineProperty(exports, "__esModule", { value: true });
var PythonShell = require('python-shell');
var path = require('path');
const cgp_core_1 = cgp_require("cgp-core");
//-----
class sampleJS {
    constructor() {
        this.allInOne_Responses = [];
        this.options = {
            runFileName: '',
            modelFileName: '',
            inputTags: [],
            outputTag: '',
            passAll: false,
        };
        this.sendData = [];
        this.appData = {};
        this.reactionOptions = { pipe: [] };
    }
    execute(appData, next) {
        this.appData = appData;
        let inputLength = 1;
        this.reactionOptions.pipe = this.options._pipeInformation;
        this.appData.data.forEach((response, index) => {
            if (response.id) {
                if (!Array.isArray(this.options.inputTags)) {
                    if (response.id == this.options.inputTags) {
                        this.allInOne_Responses[0] = response;
                    }
                }
            }
            else {
                this.options.inputTags.forEach((tag, idx) => {
                    if (response.id == tag) {
                        this.allInOne_Responses[idx] = response;
                    }
                });
            }
        });
    }
}

```

```

        });
        inputLength = this.options.inputTags.length;
    }
    if (this.allInOne_Responses.length == inputLength)
    {
        let inputData = [];
        let allTagsReceived = true;
        for (let i = 0; i <
this.allInOne_Responses.length; i++) {
            if (this.allInOne_Responses[i] &&
this.allInOne_Responses[i].id) {
                inputData[i] = [
                    this.allInOne_Responses[i].id,
                    this.allInOne_Responses[i].vqts[0].v,
                    this.allInOne_Responses[i].vqts[0].q,
                    this.allInOne_Responses[i].vqts[0].t
                ];
            }
            else {
                allTagsReceived = false;
            }
        }
        inputData.unshift(this.options.modelFileName);
        if (allTagsReceived) {
            let that = this;
            let shellpath =
__dirname.substring(__dirname.indexOf("ShellApp") + 9,
__dirname.length);
            let filepath = path.join(shellpath,
this.options.runFileName);
            let pyshell = new PythonShell(filepath);
            pyshell.send(JSON.stringify(inputData));
            pyshell.on('message', function (message) {
                message =
message.replace(/(\r\n|\n|\r)/gm, "");
                let newTag = {
                    "@type": "reaction",
                    id: that.options.outputTag,
                    vqts: [{
                        v: message,

```

```

        q: 192,
        t: Date()
    }],
    reactionType:
cgp_core_1.Ia.ReactionType.Value,
    mimeType: "",
    trackingId: "",
    reactionOptions: {},
    metadata: {
        "@type": "metadata",
        "@id": "",
        updateRateMs: 0,
        updateType:
cgp_core_1.Ia.UpdateType.Polling,
        reactionOptions:
that.reactionOptions
    },
    links: [{
        "@id": ""
    }],
    contextPrefix: "",
    contextUri: ""
}

if (that.options.passAll == true) {
    that.appData.data =
that.allInOne_Responses;
    that.appData.data.push(newTag);
}
else {
    that.appData.data = []
    that.appData.data.push(newTag);
}

that.allInOne_Responses = [];
next(null, that.appData)
});

pyshell.end(function (err) {

```

```

                if (err) {
                    throw err;
                };
            });
        }
    }
}

pushData(response) {
    this.sendData.push(response);
}

}

exports.sampleJS = sampleJS;

```

Writing a Custom Function

Few things to keep in mind when writing a custom function:

1. If using any cgp node modules then it should be used given as `cgp_require`. See the example that follows:

```
const cgp_core_1 = cgp_require("cgp-core");
```

2. Logic should be written inside `execute` method. Example follows:

```
const cgp_core_1 = cgp_require("cgp-core");
execute(appData, next) {
    // custom function logic
    next(null, appData);
});
```

3. Following write types are supported by the toolkit:
SINT, INT, DINT, LINT, REAL, DREAL, STRING, BOOL

Import Pipeline

IMPORTANT: As you are importing the Pipeline, the encryption of any sensitive data in the configuration file should be ensured by the User. We recommend using an encrypted configuration file.

To import a Pipeline:


1. Click the [] icon on the Pipeline page.

Figure 6-8: Import Pipeline



2. Enter the Application (Pipeline) name.

Figure 6-9: Define Import Pipeline


Import JSON/JSON-LD

×

Pipeline Type * :

JSON-LD ▼

Import File * :



pipeline1.json

- Filename must match with the Pipeline name
- Only one Pipeline in a JSON/dat file uploaded

Pipeline Name*:

pipeline1

CLEAR

SAVE

3. Click the Upload icon and upload the JSON-LD file. Ensure that the file name matches with the application name.
4. Click [SAVE].

Sample JSON for Pipeline

```
{
  "@context": [
    "uri://ra/cgp/config/gateway",
    {
      "cgp-conductor-manager-zmq": "cgp-conductor-manager-zmq",
      "cgp-sql-adapter": "",
      "cgp-application-sdk": ""
    }
  ],
  "@graph": [
    {
      "@id": "_:application",
      "@graph": [
        {
          "@type": "Application",

```

```

        "class": "Application",
        "name": "application1",
        "options": {
"shellEnabled": true,
        "cgp-config": {
            "@context": [
                "uri://ra/cgp/config/application",
                {}
            ],
            "@graph": [
                {
                    "@id": "_:startware",
                    "@graph": []
                },
                {
                    "@id": "_:middleware",
                    "@graph": [
                        {
                            "@id": "ra-config:egress",
                            "@type": "Middleware",
                            "class": "EgressMw",
                            "name": "egress",
                            "options": {
                                "endPoint": "ra-config:endpoint",
                                "isPost": true
                            },
                            "contextPrefix": "cgp-application-sdk"
                        }
                    ]
                }
            ],
            {
                "@id": "_:dependency",
                "@graph": [
                    {
                        "@id": "ra-config:endpoint",
                        "@type": "Dependency",
                        "class": "FileStore",
                        "name": "endpoint",
                        "options": {
                            "storeDir": "./dir",

```

```

        "maxSizeBytes": 10000000000,
        "getPolicy": "FIFO",
        "returnFiles": true,
        "extName": "data",
        "cleanOnRemove": false
    },
    "contextPrefix": "cgp-file-store"
}
]
},
{
    "@id": "_:path",
    "@graph": [
        {
            "@id": "_:b0",
            "@type": "Path",
            "name": "Pipe1",
            "path": [
                "ra-config:egress"
            ]
        }
    ]
},
{
    "@id": "_:library",
    "@graph": []
}
]
},
"request":
"{\"@context\":[\"uri://ra/cgp/action\",{\"ra-ctxjflrpeyg\":{\"ra-sql://cgp-sql-adapt
er\"}],\"@graph\":[{\"@id\":\"_:metadata\", \"@graph\":[{\"@type\":\"metadata\", \"@id
\":\"_:bjflrpeyh\", \"updateRateMs\":1000, \"updateType\":1, \"actionOptions\":{\"crede
ntialsId\":\"1\", \"tags\":[\"Tag2\"], \"mapping\":{\"id\":\"tag\", \"v\":\"tagvalue\",
\"q\":\"status\", \"t\":\"DateTimeStamp\"}, \"qualityType\":\"csm\"}, \"reactionOptions
\":{\"pipe\":[\"pipe1\"]}}]}, {\"@id\":\"_:actions\", \"@graph\":[{\"@type\":\"action\
\", \"id\":\"ra-ctxjflrpeyg:dbo.GetHistoricalDataAllTags\", \"actionType\":0, \"actionOp
tions\":{\"actionOptions\", \"reactionOptions\":{\"reactionOptions\", \"trackingId\":\"t
rack1x\", \"metadata\":[{\"@id\":\"_:bjflrpeyh\"}]}}]}]}",
    "name": "application1"
},
"contextPrefix": "cgp-application-sdk"
}

```



```

    ],
    {
      "@id": "_:adapter",
      "@graph": [
        {
          "@type": "Adapter",
          "class": "MsSqlAdapter",
          "name": "cgp-sql-adapter",
          "options": {
            "credentials": {
              "1": {
                "user": "username",
                "password": "password",
                "server": "server_name",
                "database": "database_name"
              }
            },
            "name": "cgp-sql-adapter"
          },
          "contextPrefix": "cgp-sql-adapter"
        }
      ]
    },
    {
      "@id": "_:concerto",
      "@graph": []
    },
    {
      "@id": "_:conductor",
      "@graph": [
        {
          "@type": "Conductor",
          "class": "Conductor",
          "name": "condZmq",
          "options": {
            "name": "cgp-manager",
            "ip": "127.0.0.1",
            "port": "50000",
            "commsEnabled": true,

```

```

        "brokerEnabled": true,
        "passthroughEnabled": false,
        "passthroughMultiEnabled": false
    },
    "contextPrefix": "cgp-conductor-manager-zmq"
}

]

},
{
    "@id": "_:opus",
    "@graph": []
},
{
    "@id": "_:library",
    "@graph": []
},
{
    "@id": "_:logger",
    "@graph": [
        {
            "syslog": {
                "type": "log4js-syslog-appender",
                "tag": "RACCommonGateway",
                "facility": "local0",
                "hostname": "localhost",
                "port": 514
            },
            "levels": {},
            "verbose": true,
            "logFile": true
        }
    ]
}

]
}

```