

# FactoryTalk<sup>®</sup> Analytics<sup>™</sup>



## FactoryTalk<sup>®</sup> Analytics<sup>™</sup> Edge User Guide Version 2.2

**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.

**Trademark Notices**

FactoryTalk<sup>®</sup> Analytics<sup>™</sup> DataView, FactoryTalk<sup>®</sup> Analytics<sup>™</sup> DataFlowML, FactoryTalk<sup>®</sup> Analytics<sup>™</sup> Edge and the Rockwell Software logo are registered trademarks of Rockwell Automation, Inc.

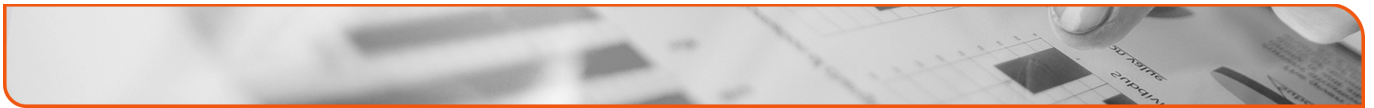
# Table Of Contents

|                  |                                       |           |
|------------------|---------------------------------------|-----------|
|                  | <b>Read Me First.....</b>             | <b>5</b>  |
|                  | Audience .....                        | 6         |
|                  | Related Documents .....               | 6         |
|                  | Conventions .....                     | 7         |
|                  | Solutions and Technical Support ..... | 7         |
|                  | Security Recommendation .....         | 8         |
| <b>Chapter 1</b> | <b>Introduction.....</b>              | <b>9</b>  |
|                  | 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 .....   | 14        |
|                  | Understanding Edge Components .....   | 15        |
|                  | Collection of Data .....              | 15        |
|                  | Adapter.....                          | 15        |
|                  | Ingress .....                         | 15        |
|                  | Analytics on Streaming Data .....     | 16        |
|                  | Route to Next Point of Value .....    | 16        |
| <b>Chapter 2</b> | <b>Pipeline.....</b>                  | <b>17</b> |
|                  | Pipeline .....                        | 18        |
|                  | Create a Pipeline .....               | 18        |
|                  | Pipeline Rules .....                  | 20        |

|                  |  |           |
|------------------|--|-----------|
|                  | Best Practices: .....                                | 29        |
|                  | Deploy Pipeline .....                                | 29        |
|                  | Pipeline Options.....                                | 31        |
|                  | Pipeline Status.....                                 | 32        |
|                  | Filter Pipelines.....                                | 32        |
|                  | Arrangement .....                                    | 32        |
|                  | Filter Options .....                                 | 32        |
|                  | Import Pipeline .....                                | 33        |
|                  | Encrypt Pipeline.....                                | 33        |
|                  | Copy Custom Processor to Runtime .....               | 36        |
| <b>Chapter 3</b> | <b>Configure Channels .....</b>                      | <b>37</b> |
|                  | Channels .....                                       | 38        |
|                  | Ingress.....   | 38        |
|                  | Broker .....   | 42        |
|                  | Transaction.....                                     | 42        |
|                  | Configuration of Transaction with MQTT Ingress ..... | 44        |
| <b>Chapter 4</b> | <b>Configure Processors.....</b>                     | <b>47</b> |
|                  | Processors .....                                     | 48        |
|                  | Custom Processor .....                               | 48        |
|                  | Python .....   | 48        |
|                  | PythonML .....                                       | 49        |
|                  | Java.....  | 50        |
|                  | Java.....  | 51        |
|                  | JavaScript.....                                      | 52        |
|                  | Compression .....                                    | 52        |
|                  | Decompression.....                                   | 52        |
|                  | BSON Serialize.....                                  | 53        |
|                  | BSON DeSerialize .....                               | 54        |
|                  | FileStore.....                                       | 54        |
|                  | Delay .....  | 55        |
|                  | Scheduler .....                                      | 56        |
|                  | Remove .....   | 57        |
|                  | Secondary Egress .....                               | 58        |
|                  | Group Tags.....                                      | 59        |
|                  | Thinning Exception.....                              | 61        |
|                  | Thinning Compression.....                            | 63        |
|                  | Math Function (Calculation).....                     | 65        |

|                  |  |           |
|------------------|--|-----------|
|                  | SqliteStore .....                              | 66        |
|                  | Writeback .....                                | 67        |
|                  | Connection Rules .....                         | 67        |
|                  | Data Types .....                               | 68        |
|                  | Configure Writeback .....                      | 70        |
| <b>Chapter 5</b> | <b>Configure Emitters .....</b>                | <b>73</b> |
|                  | Emitters .....                                 | 74        |
| <b>Chapter 6</b> | <b>JSON Creation .....</b>                     | <b>79</b> |
|                  | Sample JSON .....                              | 80        |
|                  | Channel Ingress .....                          | 80        |
|                  | FTLD .....                                     | 80        |
|                  | CIP .....                                      | 81        |
|                  | SQL .....                                      | 83        |
|                  | MQTT .....                                     | 85        |
|                  | Custom Processor .....                         | 88        |
|                  | Sample .js File .....                          | 88        |
|                  | Writing a Custom Function .....                | 91        |
|                  | Import Pipeline .....                          | 92        |
| <b>Chapter 7</b> | <b>Read Data from Edge in DataFlowML .....</b> | <b>99</b> |
|                  | Overview .....                                 | 100       |
|                  | Create and Deploy Pipeline in Edge .....       | 101       |
|                  | Create and Start Pipeline in DataFlowML .....  | 102       |

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.

- 22\_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.

- 22\_FactoryTalk Analytics Edge Advanced Installation Guide - Windows
- 22\_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.

- 22\_Analytics Security Provider Advanced Installation Guide - Windows



## User Guides

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

- 22\_FactoryTalk Analytics Edge User Guide
- 22\_FactoryTalk Analytics Edge Administration Guide
- 21\_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
- ❑ Commands that the user enters from keyboard appear in bold text.  
For example: Execute the command **node server**
- ❑ 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 15**

## Overview of Edge

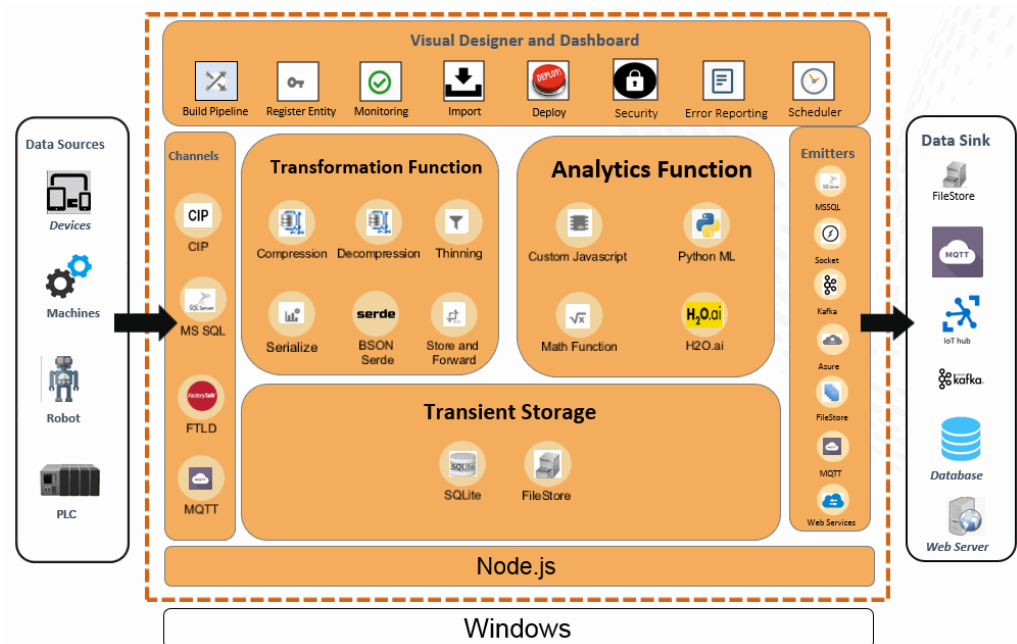
FactoryTalk® Analytics Edge™ (called Edge hereafter) is a low foot-print Analytics Platform capable of Real-Time and Batch processing with low latency at the Edge layer. Edge supports data ingestion from multiple data sources such as FactoryTalk Live Data, SQL, CIP, MQTT, and can Egress processed data 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.

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, FTLTD etc.).
  - Data at Rest (SQL).
- ❑ Analytics on data in motion (Stream).
  - Data Cleansing, Refinement, Transformation, Store and Forward.
  - Machine Learning (Python ML, H2O).
- ❑ 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:** Currently, Admin User, Design User, and View User have the same role permissions.

---

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

1. In the browser, use the provided URL:
  - <https://<FQDN>:<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 880 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].

---

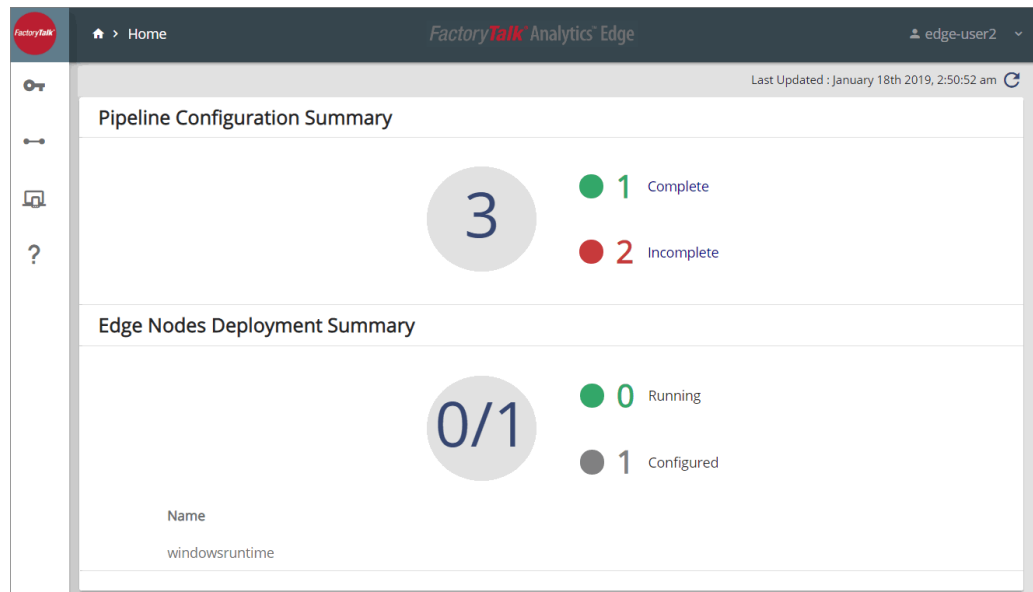
**IMPORTANT:** When changing the password, create a complex password as per your company policy.

---

Figure 1-3: Change Password



5. The Edge User Home page displays.

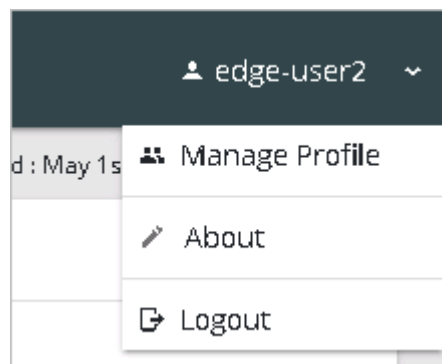
**Figure 1-4: Home Page**

## 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 per your company policy.

---

**Figure 1-6: Change Password**

## User Dashboard

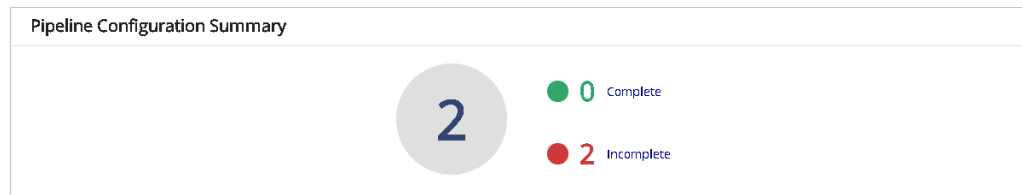
The User Dashboard is the landing page when a User logs in. The widgets on the Dashboard provide Pipeline Configuration Summary and Edge Nodes Deployment Summary.

### Pipeline Configuration Summary

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

| Pipeline Summary  |   |
|-------------------|---|
| <b>Pipelines</b>  | The number of Pipelines created by all the users. |
| <b>Complete</b>   | Number of Pipelines configured completely.        |
| <b>Incomplete</b> | Number of incomplete Pipelines.                   |

**Figure 1-7: Pipeline Configuration Summary**



### Edge Nodes Deployment Summary

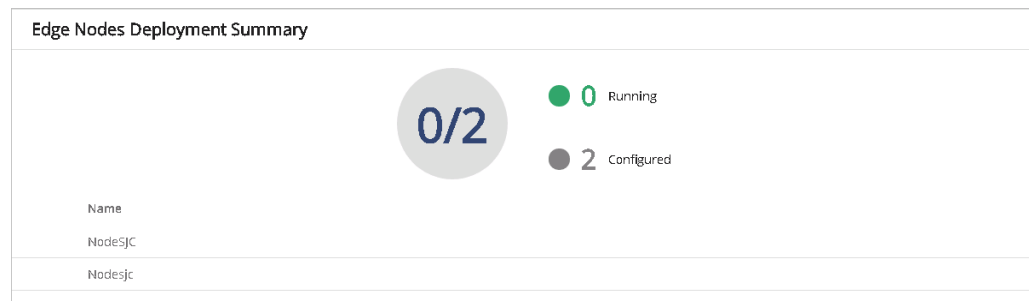


This widget displays the summary of Nodes.

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

| Nodes Summary     |  |
|-------------------|--|
| <b>Nodes</b>      | The number of Nodes connected with Gateway versus the number of Nodes configured successfully. |
| <b>Running</b>    | Number of Nodes connected with Gateway.  |
| <b>Configured</b> | Number of Nodes configured successfully.   |

**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, Mosquitto version 1.4.14 is supported.

## **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)

## **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



# Chapter

# 2

## Pipeline

### In this chapter:

- ❑ Pipeline 18
- ❑ Create a Pipeline 18
- ❑ Deploy Pipeline 29
- ❑ Import Pipeline 33

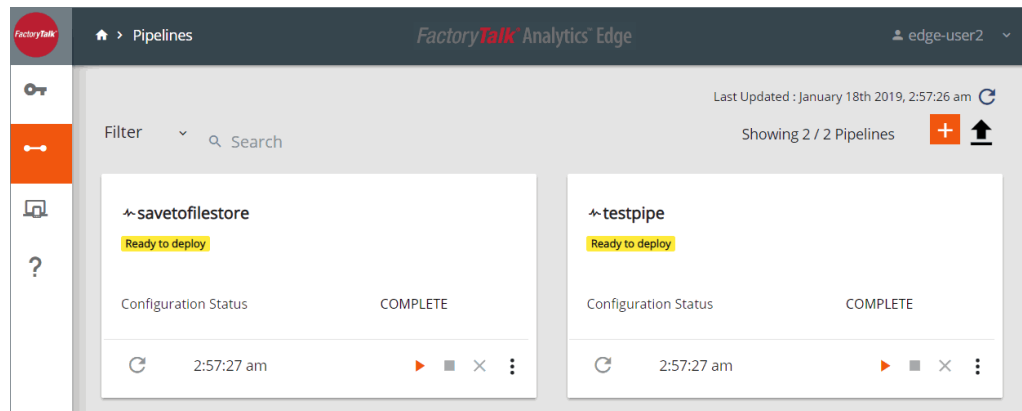
## Pipeline

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:** 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.

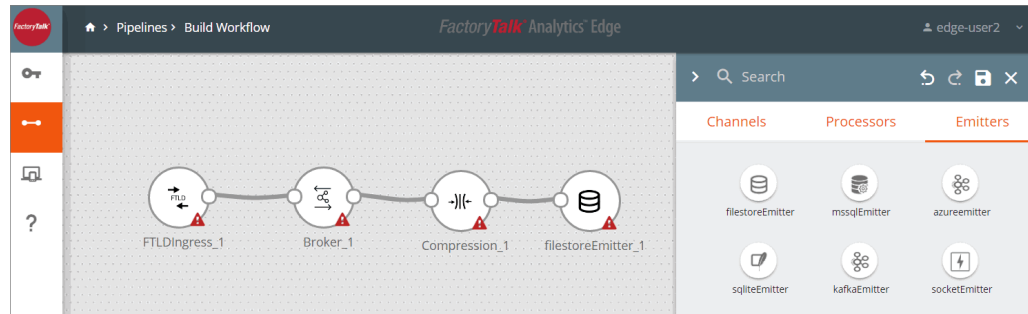
---


**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.

---

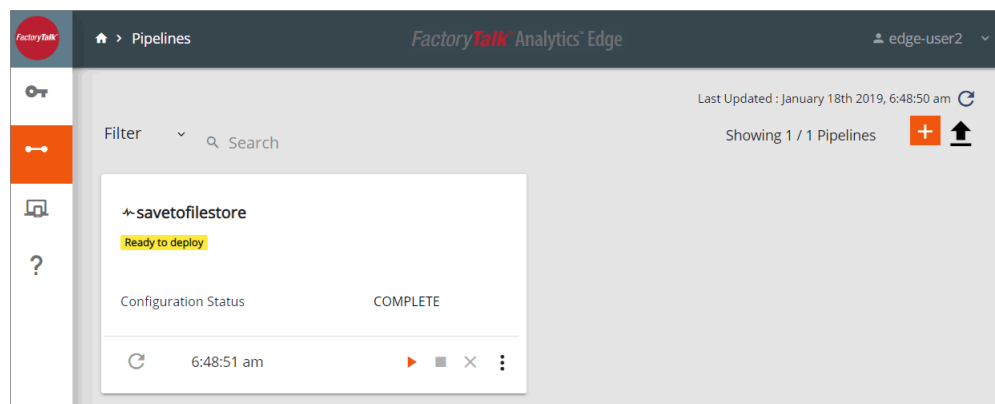
**NOTE:** Pipeline names can contain only lower case alphabets (a to z), numbers (0 to 9) or special characters (\_, @, -) and must start with an alphabet.

---

**Figure 2-3: Save Pipeline**

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

**Figure 2-4: Pipeline List**



## Pipeline Rules

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

Follow the rules in the table that follows to build a valid Pipeline. 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.

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.

---

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

| Component<br>(Current point<br>of value) | Can Connect to<br>(Next point of value)   | Can't Connect To<br>(Next point of value)  |
|--|---|--|
| 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<br>(Current point<br>of value) | Can Connect to<br>(Next point of value)  | Can't Connect To<br>(Next point of value)  |
|--|--|--|
| FileStore                                | <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> |
| 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<br>(Current point<br>of value) | Can Connect to<br>(Next point of value)  | Can't Connect To<br>(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<br>(Current point<br>of value) | Can Connect to<br>(Next point of value)   | Can't Connect To<br>(Next point of value)  |
|--|---|--|
| Thinning<br>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<br>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<br>(Current point<br>of value) | Can Connect to<br>(Next point of value)  | Can't Connect To<br>(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<br/>Compression, Thinning<br/>Exception, Group Tags</p> |
| BSON<br>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<br/>Compression, Thinning<br/>Exception, Group Tags</p> |

| Component<br>(Current point<br>of value) | Can Connect to<br>(Next point of value)  | Can't Connect To<br>(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> |

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

| Component<br>(Current point<br>of value)                             | Can Connect to<br>(Next point of value)  | Can't Connect To<br>(Next point of value)   |
|--|--|---|
| <b>Custom Processors</b>   |  |   |
| NodePython,<br>NodeML,<br>NodeH2oClass,<br>NodeH2oJar,<br>JavaScript | All Emitters<br><br>Any Custom Processors<br><br>Writeback<br><br>Processors: Calculation,<br>Compression, FileStore,<br>Delay, BSON Serialize,<br>Thinning Compression,<br>Thinning Exception, Group<br>Tags, SqliteStore | Any Ingress components<br><br>Transaction<br><br>Broker<br><br>Processors: BSON<br>DeSerialize, Decompression,<br>Scheduler, Remove |
| Writeback  | All Emitters<br><br>Any Custom Processors<br><br>Writeback<br><br>Processors: Calculation,<br>Compression, FileStore,<br>Delay, BSON Serialize,<br>Thinning Compression,<br>Thinning Exception, Group<br>Tags, SqliteStore | Any Ingress components<br><br>Transaction<br><br>Broker<br><br>Processors: BSON<br>DeSerialize, Decompression,<br>Scheduler, Remove |
| <b>Emitters</b>  |  |   |
| All Emitters   |  | Any Ingress components<br>Transaction<br>Broker<br>Any Processors<br>Any Custom Processors<br>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.
- When you build a Pipeline 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
- There will be no data for the output tag that is defined in the Writeback Processor if there are multiple Egresses in the Pipeline.
- 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 CGP. It is recommended to use a Delay Processor in the Pipeline to avoid this type of error.

## Deploy Pipeline

---

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

---

---

**NOTE:** Pipelines that have FTLD ingress 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.

**Figure 2-5: Deploy Pipeline**

| Node Name | Action                   |
|-----------|--------------------------|
| Node1     | <input type="checkbox"/> |
| Node2     | <input type="checkbox"/> |
| Node3     | <input type="checkbox"/> |

Showing 1 to 3 of 3 entries

Previous 1 Next

CANCEL DEPLOY

2. Select the Node or Nodes on which the Pipeline is to be deployed and click [DEPLOY].

**Figure 2-6: Select Nodes**

| Node Name | Action                              |
|-----------|-------------------------------------|
| Node1     | <input checked="" type="checkbox"/> |
| Node2     | <input type="checkbox"/>            |
| Node3     | <input checked="" type="checkbox"/> |

Showing 1 to 3 of 3 entries

Previous 1 Next

CANCEL DEPLOY

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.

---

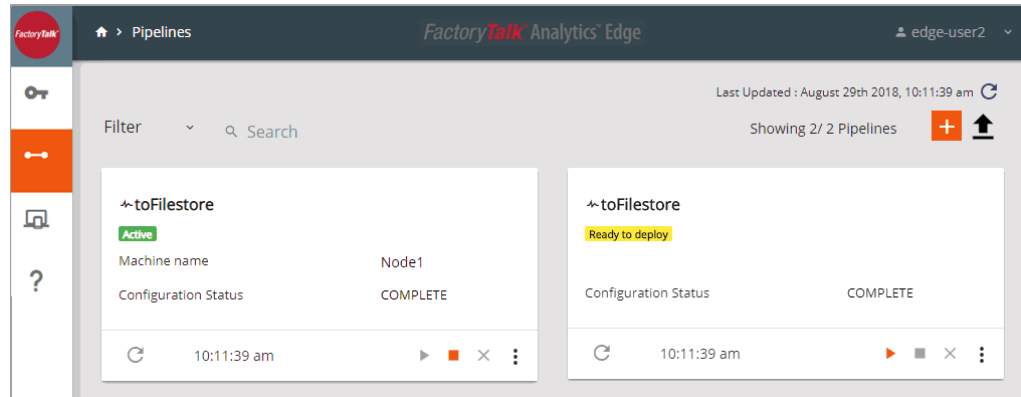


---

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

---

**Figure 2-7: 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

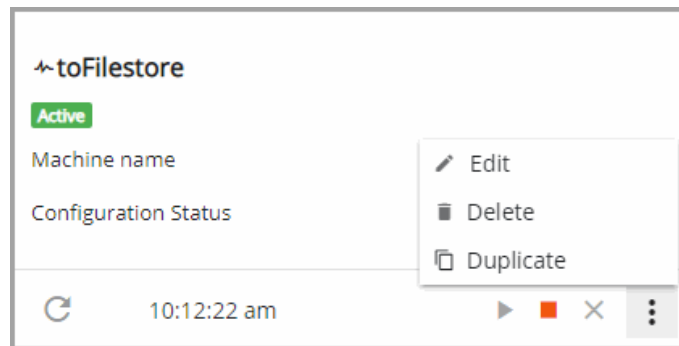
Click the [ ⋮ ] icon shown adjacent to the Undeploy button. The Pipeline options display:

- Edit - To edit the Pipeline.
- Delete - To delete the Pipeline.
- Duplicate - To copy the Pipeline and save it with a different name.

---

**NOTE:** A Pipeline cannot be edited or deleted until it is undeployed.

---

**Figure 2-8: 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.

## Filter Pipelines

### Arrangement

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

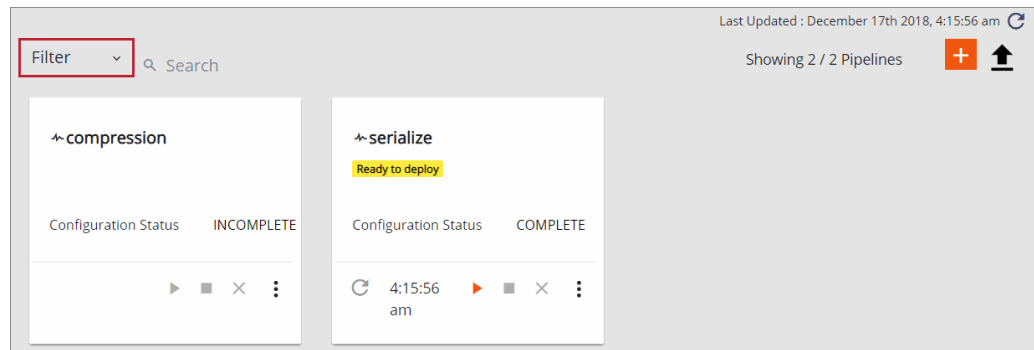
### Filter Options

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

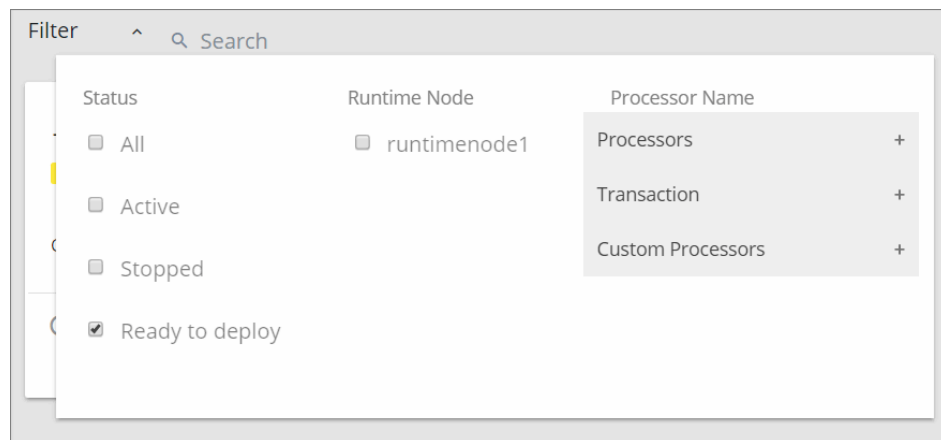
- By Runtime node- Pipelines that are deployed or stopped using the node name
- By Processor name - Pipelines using the Processor name, including the pre-built and custom processors
- By Transaction name - Pipelines using the transaction name
- By FTLD WriteBack name - Pipelines using the Writeback Processor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

Perform the following steps to apply the filter:

1. Click the [Filter] dropdown.

**Figure 2-9: Filter**

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

**Figure 2-10: Apply Filter**

3. The Pipeline dashboard displays only the filtered Pipelines.

## Import Pipeline

The Import Pipeline feature enables you to import a Pipeline along with its component configurations such as Adapters, Processors and Emitters.

There are two options to import a 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
eyJhbGciOiJSU0ExXzUiLA0KICJlbnMiOiJBMjU2R0NNIiwNCiAiaXYiOiJfXz
```

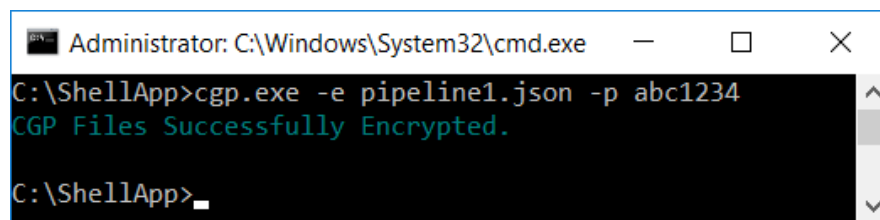
To encrypt the JSON file using a password, run the following command:

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

For Example:


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

**Figure 2-11: 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 password method is used to encrypt, **salt.bin** will also be generated the ShellApp folder.

To import a Pipeline:

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

**Figure 2-12: Import Pipeline**



2. Enter the Application (Pipeline) name.

**Figure 2-13: Define Import Pipeline**

Import JSON-LD

Application Name : \* pipeline1

Import JSON-LD by selecting file : \* pipeline1.json

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

CLEAR SAVE

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

---

**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.

---

4. 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**

Import JSON-LD

Application Name : \* pipeline1

Import JSON-LD by selecting file : \* pipeline1.dat

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

Encryption : \* ☒ Key ☐ Password

eyJhbGciOiJSU0ExXzUiLA

CLEAR SAVE

5. 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.

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

---

**NOTE:** An imported Pipeline cannot be edited or duplicated.

---

## Copy Custom Processor to Runtime

---

**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 "./CustomFunctions/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\CustomFunctions\I\_pipelineName  
Where pipelineName is the name of the Pipeline.
8. Deploy the Pipeline.



# Chapter

# 3

## Configure Channels

### In this chapter:

- ❑ Channels 38

## 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

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, Mosquitto version 1.4.14 is supported.

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. Change the default properties, if required and select the Pipe Path. Click [SAVE].

---

**NOTE:** The Pipe Path field is enabled only after connecting all the components in the Pipeline. Select the pipe from the dropdown list.

---



**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 : \* FilestoreEmitter\_1\_path-202

CANCEL SAVE

**Figure 3-2: Configure Ingress - CIP**

Configuration Settings - CIPIngress\_1

Select Source

Tag Name : \* CIPTag1

Read Option : \* Continuous Read

CIP Device Shortcut (Path) : \* ra-clx://cgp-cip-adapter/xxx.xxx.xxx.xx

Tracking ID : Please enter value

Update Frequency (ms) : \* 1000

Update Type : \* Polling

Pipe Path : \* FilestoreEmitter\_1\_path-860

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 : \* tag1

Mapping : \* id : \* tag

CANCEL SAVE

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

Configuration Settings - SQLIngress\_1

Select Source

Tag Request Config : \* Tags : \* tag1

Mapping : \* id : \* tag

v : \* tagvalue

q : \* status

t : \* timestamp

Pipe Path : \* FilestoreEmitter\_1\_path-860

CANCEL SAVE

**Figure 3-5: Configure Ingress - MQTT**

Configuration Settings - mqttIngress\_1

Select Source

Use SSL : \* ☐ true ☒ false

Quality of Service : \* At most once

IP : \* 10.85.116.59

Port : \* 1883

Topic Name : \* devicedata

CANCEL SAVE

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

Configuration Settings - mqttIngress\_1

Select Source

Port : \* 1883

Topic Name : \* devicedata

Read Option : \* Continuous Read

Tracking ID : Please enter value

Pipe Path : \* Select Pipes

file\_1\_path-687

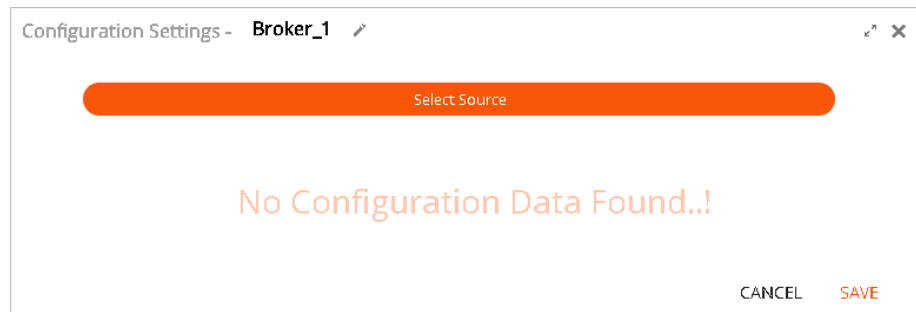
CANCEL SAVE

## 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.

**Figure 3-7: 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.

---

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.

The following constraints apply for the Transaction:

- Set SendImmediate to false won't work for SQL adapter tags.
- Trigger conditions Equal and NotEqual 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 one event Read option is Async Read, out of multiple event tag, then the Caching won't applicable. Caching will be applicable only to CIP and FTL D 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. You cannot stop this. This is the behavior of the transaction. Caching will egress the trigger and event data together. No matter send immediate is true or not.
- The onChange trigger can be set using either the threshold parameter or the onChangePos and onChangeNeg parameters. If the threshold value is provided, onChange will be determined as absolute change from previous value greater than equal to the threshold. If both onChangePos and onChangeNeg values are provided, onChange 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 `tipple === 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**

| <b>MQTT Input:<br/>Trigger Value<br/>Examples</b> | <b>High</b> | <b>Low</b> | <b>Equal</b> | <b>Not<br/>Equal</b> |
|---|-------------|------------|--------------|----------------------|
| "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).<br><b>Select Tag:</b> Select the trigger tag from the dropdown list   |
| Trigger Condition | Select the condition to accept the data.   |
|                   | <b>Condition:</b> Trigger condition to move the Trigger and Event data when Trigger Tag data meets the condition.  |
|                   | <b>Threshold Value:</b> Threshold value for trigger conditions. All trigger conditions should have a threshold value, except the trigger conditions All, High, Low.  |
|                   | <b>Initialization Value:</b> Initial value for evaluation of OnChange, Increasing and Decreasing trigger conditions. Initial value is considered only when previous trigger value does not exist.  |
|                   | <b>Send Immediate</b><br><b>Yes:</b> The trigger tag data and event tag data sent as they come in.<br><b>No:</b> The trigger tag data and the event tag data collected as a single array and sent to Pipeline.   |
|                   | <b>Ignore Initial Value</b><br><b>Yes:</b> The first value of trigger tag data is ignored for trigger condition evaluation.<br><b>No:</b> The first value of trigger tag data is considered for the trigger condition evaluation   |
|                   | <b>Egress Type:</b> Set this field to indicate that trigger condition should be triggered on a true state rather than a transition from false to true.<br>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. |

| Field  | Description  |
|--------|--|
| Events | Select the Event.<br><b>Select Tags:</b> Select the Event tags |

Figure 3-8: Configure Transaction

Configuration Settings - Transaction\_1

Select Source

Trigger Data

Select Tag : \* rawdata

Trigger Condition

Condition : \* Greater Than

Threshold Value : \* 10

Send Immediate : \* ☒ Yes ☐ No

Ignore Initial Value : \* ☒ Yes ☐ No

CANCEL SAVE

Figure 3-9: 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 : \* rawdata

CANCEL SAVE





# Chapter

# 4

## Configure Processors

### In this chapter:

- ❑ Processors 48

## 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
- ☐ “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   |
|------------------|---|
| <b>fileName</b>  | Name of the Python file.  |
| <b>tags</b>      | Array of tags (e.g., Tag1,Tag2,Tag3).   |
| <b>outputTag</b> | Name of the new output tag.   |
| <b>passAll</b>   | 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

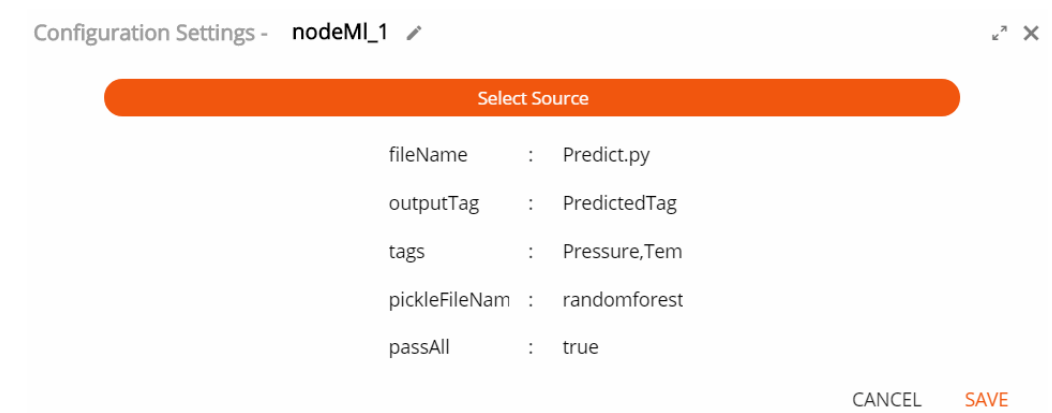
## PythonML

NodeML is the sample Python ML Custom Processor supplied with the application. The sample files are available in the <Install\_Directory>\samples\ml directory.

This module allows user to upload and run a pre-trained ML model as part of a data Pipeline. The NodeML module is used for Machine learning in Node Js.

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

| Field                 | Description   |
|-----------------------|---|
| <b>fileName</b>       | Name of the Python file.  |
| <b>outputTag</b>      | Name of the new output tag.   |
| <b>tags</b>           | Array of tags (e.g., Tag1,Tag2,Tag3).   |
| <b>pickleFileName</b> | Name of the prediction model file (e.g., randomforest.pkl)                              |
| <b>passAll</b>        | Boolean (will pass all incoming tags when set to true, by default this option is true). |

**Figure 4-2: Configure Custom Processor - NodeML**

## Java

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   |
|------------------|---|
| <b>tags</b>      | Array of tags (e.g., Tag1,Tag2,Tag3).   |
| <b>className</b> | Name of the JAVA class.   |
| <b>outputTag</b> | Name of the new output tag.   |
| <b>passAll</b>   | 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

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   |
|------------------|---|
| <b>JarName</b>   | Name of the jar file (e.g., abc.jar).   |
| <b>className</b> | Name of the JAVA class.   |
| <b>outputTag</b> | Name of the new output tag.   |
| <b>passAll</b>   | Boolean (will pass all incoming tags when set to true, by default this option is true). |
| <b>tags</b>      | 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]

## 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   |
|------------------------------|---|
| <b>Output Type</b>           | Select the output type (buffer or string) from the dropdown list.                                   |
| <b>Compression Algorithm</b> | Select the compression algorithm from the dropdown list. Available algorithms are gzip and deflate. |

**Figure 4-5: 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   |
|--------------------------------|---|
| <b>Output Type</b>             | Select the same Output Type as in the Compression Processors.<br>Select [string], if Compression Processor is set as string.<br>Select [buffer], if Compression Processor is set as buffer. |
| <b>Decompression Algorithm</b> | Select the same Compression Algorithm as in the Compression Processors.<br>Decompression Algorithm: gzip<br>Decompression Algorithm: deflate  |

**Figure 4-6: Configure Processor - Decompression**

Configuration Settings - Decompression\_1

Select Source

Output Type : \* string ▼

Decompression Algorithm : \* gzip ▼

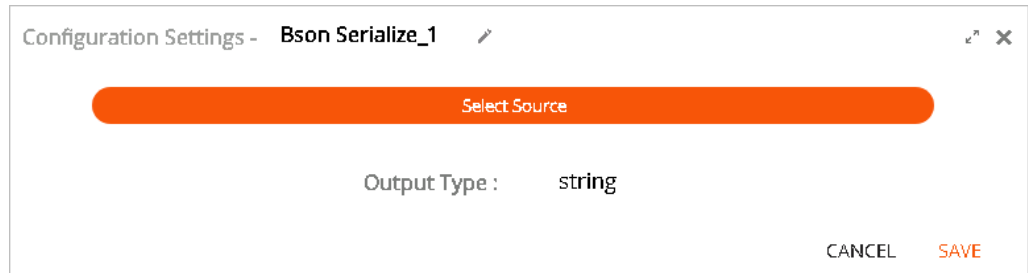
CANCEL SAVE

## 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                  |
|--------------------|------------------------------|
| <b>Output Type</b> | Serialize to BSON as string. |

**Figure 4-7: 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 4-8: 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  |
|-------------------------------------|--|
| <b>Store Path</b>                   | Path where file will be stored. If not set will use the home directory of deployment. Example: C:\ShellApps\   |
| <b>Folder Name</b>                  | Folder Name created in Store Path.<br>Example: Folder1   |
| <b>Maximize Folder Size (Bytes)</b> | Set maximum size of data in folder. Rollover will occur once the limit is reached.   |
| <b>File Extension Name</b>          | File extension for the stored data.<br>Example: log or txt or json   |
| <b>Outbound Read Sequence</b>       | The next Processor will read file in selected order (FIFO/LIFO), if multiple files are read concurrently.<br><br><b>FIFO:</b> First In First Out<br><b>LIFO:</b> Last In First Out |

**Figure 4-9: Configure Processor - FileStore**

Configuration Settings - FileStore\_1

SF-Filestore

Store Path : C:\ShellApp\

Folder Name : \* ProcessorFolder

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   |
|---------------------------------|---|
| <b>File Fetch Count</b>         | Define the number of files to read in each fetch.<br>Example: if set to 3, then 3 files are read on each fetch. |
| <b>File Fetch Interval (ms)</b> | Define fetch frequency.   |

**Figure 4-10: Configure Processor - Delay**

Configuration Settings - Delay\_2

Select Source

File Fetch Count : \* 5

File Fetch Interval : \* 3000

CANCEL SAVE

## 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  |
|----------------------------|--|
| <b>Fetch Count</b>         | Define the number of files to read in each fetch.<br>Example: if set to 3, then 3 files are read on each fetch |
| <b>File Fetch Interval</b> | Define fetch frequency.  |

**Figure 4-11: Configure Processor - Scheduler**

## 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   |
|---|---|
| <b>Name of Filestore to remove from</b> | Name of the Store and Forward Filestore to remove files from. |

**Figure 4-12: Configure Processor - Remove**

Configuration Settings - Remove\_1

Select Source

Name of FileStore to remove from : \* FileStore\_2

CANCEL SAVE

## Secondary Egress

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

- Move: Move data to both the primary and secondary endpoint
- 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.
- 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 4-13: 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  |
|---------------------------|--|
| <b>Tags to be Grouped</b> | Tag Names separated by comma.<br>Example: <Tag1>,<Tag2>,<Tag3>,<Tag4>  |
| <b>Send Immediate</b>     | <p>Define the Tags grouping behavior:</p> <p><b>False:</b> Send the group of tags as a set when <b>all the tags</b> in the set have an updated value.</p> <ul style="list-style-type: none"> <li>If Tags are configured as polling then they are considered updated based on polling frequency.</li> <li>If tags are configured as Onchange, then they are considered updated only when value of tags have changed at source.</li> </ul> <p><b>True:</b> Send the group of tags as a set when <b>any one tag</b> in the set have an updated value.</p> <ul style="list-style-type: none"> <li>If Tags are configured as polling then they are considered updated based on polling frequency.</li> <li>If tags are configured as on change, then they are considered updated only when value of tags have changed at source.</li> </ul> |

**Figure 4-14: Configure Processor - Group Tags**

Configuration Settings - Group Tags\_1

Select Source

Tags to be Grouped : \* Tag1,Tag2

Send Immediate : ☐ true ☒ false

CANCEL SAVE

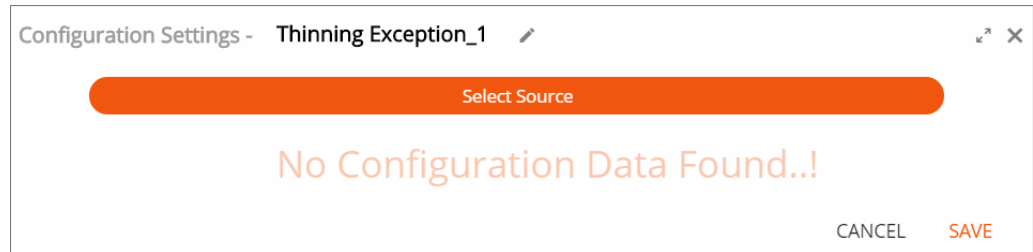
**NOTE:** Group Tags processor does not push the tags which are not defined in it.

## 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).

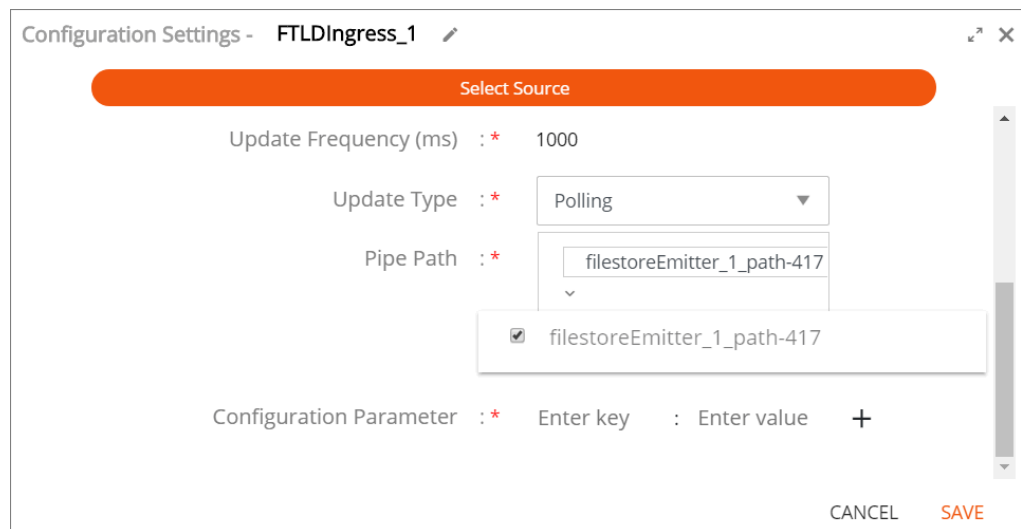
1. Click and drag the Thinning Processor onto the Visual Designer and right-click the Processor to configure.
2. Click [SAVE] to enable Thinning Exception.

**Figure 4-15: 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 4-16: Select Pipe Path**

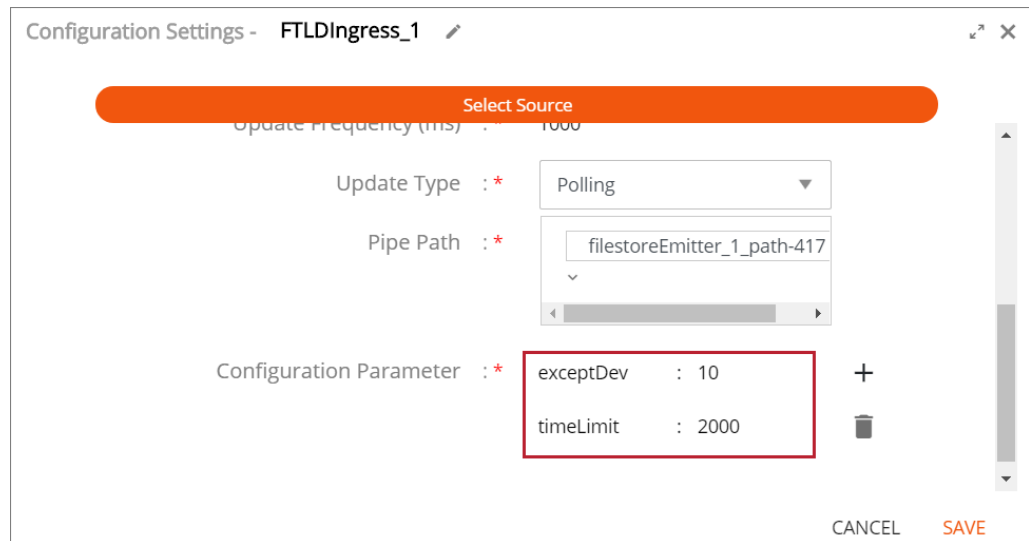


Define the following Thinning Exception properties and click [SAVE].

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

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

**Figure 4-17: Thinning Exception Properties**



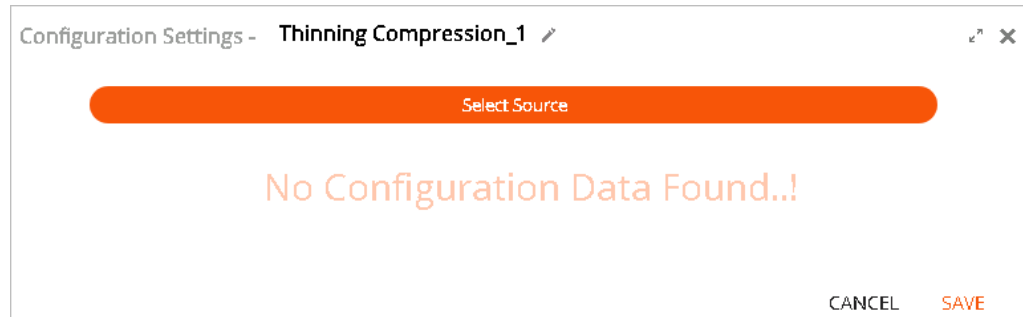


## Thinning Compression

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).

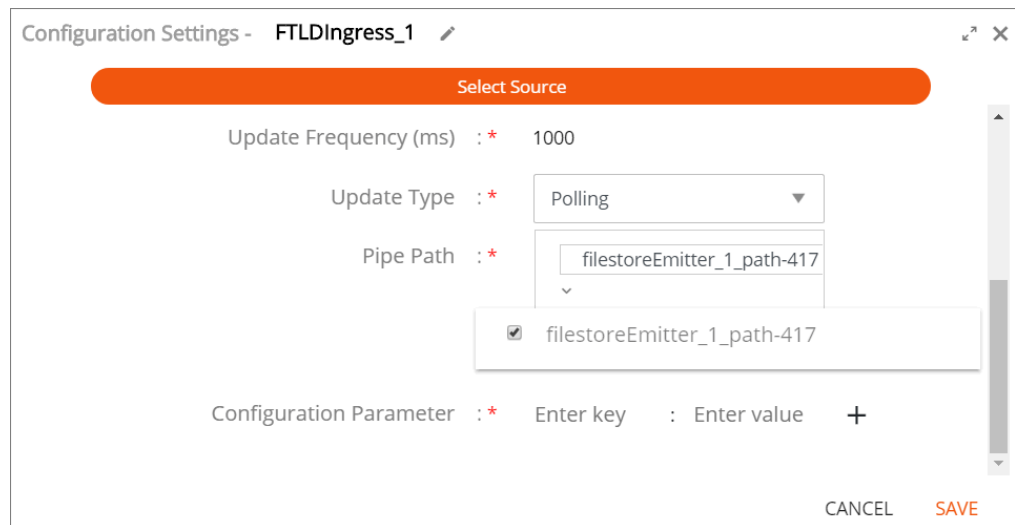
1. Click and drag the Thinning Compression Processor onto the Visual Designer and right-click the Processor to configure.
2. Click [SAVE] to enable Thinning Compression.

**Figure 4-18: 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 4-19: Select Pipe Path**



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

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

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

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

**Figure 4-20: Thinning Compression Properties**

Configuration Settings - FTLIngress\_1

Select Source

Update Frequency (ms) : \* 1000

Update Type : \* Polling

Pipe Path : \* filestoreEmitter\_1\_path-417

Configuration Parameter : \*

- compDev : 10
- 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                                  |
|------------------------|--|
| <b>Expression</b>      | The expression for the calculation.          |
| <b>Tags</b>            | The Tags that users selected for expression. |
| <b>Output Tag Name</b> | Name of the output Calculation Tag.          |

**Figure 4-21: 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: +, -, *, /, %<br>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   |
|----------------------------------|---|
| <b>Store Path</b>                | Path where file will be stored. If not set then will use the home directory of deployment.<br>Example: C:\ShellApp\ |
| <b>Database Name</b>             | Enter the name of the Database  |
| <b>Table Name</b>                | Enter the name of the Table   |
| <b>Database Max Size (Bytes)</b> | Maximum size for the Table in Database. One Table in each Database is recommended.                                  |

**Figure 4-22: SqliteStore Processor**

Configuration Settings - SqliteStore\_1

SF-Sqlite

Store Path : C:\SQLiteStore

Database Name : \* TagDatabase

Table Name : \* TagTable

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. Currently, the toolkit supports write back to the PLC only through the FTL D adapter, which means the value of the "Writeback Shortcut (Path)" option should have a path to FTL D.

## 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, 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 (FTLD, CIP, SQL and 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**

|                          |        | 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 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 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  |
|----------------------------------|--|
| <b>Input Tag Name</b>            | The incoming tag name to read data.  |
| <b>Select Adapter</b>            | Select the Adapter from the dropdown list.   |
| <b>Input Tag Data Type</b>       | Select the data type of the Input Tag.   |
| <b>Writeback Tag Name</b>        | Enter tag name to Writeback data.  |
| <b>Writeback Shortcut (Path)</b> | Device path in FT Admin Console.<br>Device path example: ra-ftld://cgp-ftld/< Device Shortcut><br>Device Shortcut example:<br>RNA://\$Global/EAPTest/EAPArea::[Physical_PLC] |



| Field                          | Description   |
|--------------------------------|---|
| <b>Writeback Tag Data Type</b> | 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.        |
| <b>Tracking ID</b>             | This ID will be available in data response json object as an additional field for tracking.<br>Response object format for Tracking Id:<br>"trackingId":<ID specified> |

**Figure 4-23: Configure - Writeback**

Configuration Settings - WritebackProcessor\_1

Select Source

Input Tag Name : \* realRandom

Select Adapter : \* CIP

Input Tag DataType : \* SINT

Writeback Tag name : \* newTag

Writeback Shortcut (Path) : \* ra-ftld://cgp-ftld/RNA://\$Global/Test

Writeback Tag DataType : \* SINT

Tracking ID : TrackrealRandom

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 74

## 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, change the default configuration and click [SAVE].

**Figure 5-1: Configure Emitter FileStore**

Configuration Settings - Filestore\_2

Select Source

Store Path : C:\ShellApps

Folder Name : \* endpoint

Maximum Folder Size(Bytes) : \* 10000000

File Extension Name : \* json

CANCEL SAVE

---

**NOTE:** Do not change the Component Name of the Emitter from the Pipeline canvas. It is recommended to register another Emitter and replace the existing emitter in the Pipeline.

---

**Figure 5-2: Configure Emitter MSSQL**

Configuration Settings - mssqlEmitter\_1

Select Source

Server Name : \* 192.168.20.12

Database Name : \* database1

Username : \* root

Password : \* .....

Select Destination Type : \* ☐ table ☒ storedProcedure

Name : \* getName

Mapping : \* tag0 : tagname1 +

CANCEL SAVE

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

Configuration Settings - mssqlEmitter\_1

Select Source

Name : \* getName

Mapping : \* tag0 : tagname1 +

Mapping Flow : \* key-->Value ▼

Insert 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 : \* https://rastorageaccount.bl

Blob Token : \* ?sv=2018-09-12&st=2015-04

Blob Container Name : \* racontaineredge

To Queue : \* ☒ true ☐ false

Queue Service Endpoint : \* https://raedge.queue.core.v

Queue Token : \* ?sv=2019-10-11&st=2015-04

Queue Name : \* edgeque

CANCEL SAVE

**Figure 5-6: Configure Emitter IOT Hub**

Configuration Settings - IoTHub\_1

Select Source

Device ID : \* IOTdevice1

Host Name : \* IOTHubName1.azure-devices.net

Connection String : \* IOTHubName1.azure-devices.net;SharedAc

CANCEL SAVE

**Figure 5-7: Configure Emitter Kafka**

Configuration Settings - kafkaEmitter\_1

Select Source

Post Topic Name : \* edgeTopic

Post IP : \* 192.168.10.23

Post Port : \* 803

Post Partitions : \* 1

Use SSL : \* ☒ true ☐ false

Import Certificate file : \* cert\_apinbanisepqa2.ra-...

Import Key file : \* key\_apinbanisepqa2.ra-i...

Import Certificate Authority file : \* ca ca.crt

CANCEL SAVE

**Figure 5-8: Configure Socket IO Source**

Configuration Settings - socketEmitter\_1

Select Source

Select Source Gateway Server

Room Name : \* edgeroom

Post Event Name : \* tagdata

CANCEL SAVE

**Figure 5-9: Configure Socket IO GatewayServer**

Configuration Settings - socketEmitter\_1

Select Source

Select Source Gateway Server

Port : \* 8080

CANCEL SAVE

**Figure 5-10: Configure Emitter MQTT**

Configuration Settings - mqttEmitter\_1

Select Source

Post Host Name : \* apinbanisepqa2.ra-int.com

Post Port : \* 8883

Topic Name : \* egressTopic

Clean Session : \* ☒ true ☐ false

Client ID : Please enter value

Quality of Service : \* At most once ▼

Use SSL : \* ☒ true ☐ false

CANCEL SAVE

**Figure 5-11: Configure Emitter MQTT (Continued)**

Configuration Settings - MqttChannel\_2

Select Source

IP : \* apinbanisepqa2.ra-int.com

Port : \* 8883

Topic Name : \* ingressTopic

Read Option : \* Continuous Read ▼

Tracking ID : track1

Pipe Path : \* filestoreEmitter\_1\_path-181

CANCEL SAVE





# Chapter

# 6

## JSON Creation

### In this chapter:

- ❑ **Sample JSON** 80
- ❑ **Channel Ingress** 80
- ❑ **Custom Processor** 88
- ❑ **Import Pipeline** 92

# 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,
}
```

CLEAR REGISTER

```
{
  "id"      : "rawdata",
  "actionType" : "0",
  "contextUri" :
    "ra-ftld://cgp-ftld/RNA://$Global/TestApp/TestArea::[110Slot0]"
  ,
  "contextPrefix" : "a",
  "trackingId" : "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" : "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            |
| 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 : \* SQL1

Configuration Type : ☐ Fields ☒ Import (Json)

Server : \* 127.0.0.1

Database : \* factory

Username : \* user1

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_cgp-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

Quality of Service : \*

At most once ▼

IP : \*

apinbanisepqa2.ra-int.com

Port : \*

8083

Tag: 1

Enter Action Data: \*

```
{
  "id": "test1",
  "actionType": 0,
  "contextUri": "ra-mqtt://cgp-mqtt-adapter/mqtt://apinbanisepqa2.ra-int.com:8083",
  "trackingId": "123",
  "updateRateMs": 1000,
  "updateType": 0,
  "groupActionOptions": {},
}
```

CLEAR

REGISTER

```

{
  "id": "test1",
  "actionType": 0,
  "contextUri":
"ra-mqtt://cgp-mqtt-adapter/mqtt://apinbanisepqa2.ra-int.com:80
83",
  "trackingId": "123",
  "updateRateMs": 1000,
  "updateType": 0,
  "groupActionOptions": {},
  "groupReactionOptions": {},
  "itemActionOptions": {
    "connectionString":
"mqtt://apinbanisepqa2.ra-int.com:8083",
    "publishOptions": {
      "qos": "0"
    },
    "connectionOptions": {
      "useSSL": true,
      "sslOptions": {
        "cert":
"./shared/mqtttest/cert_server-crt.pem",
        "key":
"./shared/mqtttest/key_server-key.pem",
        "ca": "./shared/mqtttest/ca_ca-crt.pem",
        "rejectUnauthorized": true
      }
    }
  },
  "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 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://127.0.0.1:8083",  
  "publishOptions": {  
    "qos": "0"  
  },  
  "connectionOptions": {  
    "useSSL": true,  
    "sslOptions": {  
      "cert": "./shared/mqtttest/cert\_server-crt.pem",  
      "key": "./shared/mqtttest/key\_server-key.pem",  
      "ca": "./shared/mqtttest/ca\_ca-crt.pem",  
      "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/mqtttest<This is the component name>/cert\_server-crt.pem<This is the file name which is showed below the upload icon of import Certificate file after uploading>"
- key should be "./shared/mqtttest<This is the component name>/key\_server-key.pem<This is the file name which is showed below the upload icon of import Key file after uploading >"
- ca should be "./shared/mqtttest<This is the component name>/ca\_ca-crt.pem<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 { }

## Custom Processor

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

**Figure 6-5: Custom Processor Tab**

**Figure 6-6: Custom Processor**

## Sample .js File

The following example shows a Custom Processor called writeToPlc. Options of writeToPlc are as follow:

```
readTag: “”, // name of tag whose value you want to write to plc ex:
              Tag coming from sql adapter // required

writeTag: “”, // name of tag present in plc to which you want to write
              the value // required

writeUrl: “”, // path to the plc ex:
              'ra-ftld://cgp-ftld/RNA://$Global/TestApps/TestApps::[tes
              t]' // required

writeType: “”, // these are the data type present in plc, data types
               specified/selected when creating a tag in plc. // required

contextPrefix: “”, // context prefix , not really required
                  // optional

trackingId: “” // tracking id
              // optional
```

```

"use strict";
Object.defineProperty(exports, "__esModule", {value: true });
const cgp_core_1 = cgp_require("cgp-core");

class writeToPlc {
    /**
     * Constructor for writeto plc Class.
     */
    constructor() {
        this.options = {
            readTag: '',
            writeTag: '',
            writeUrl: '',
writeType: '',
            contextPrefix: '',
            trackingId: ''

        };
    }
    /**
     * This function is the middleware function.
     *
     * @param next The function the middleware should call
     *             when complete. Next is of the form
     *             function(err) .
     */
    execute(appData, next) {
        let actionData = new cgp_core_1.Ia.ActionData();
        this.appData = appData;
        let dataArray = appData.data;
        dataArray.forEach((response) => {
            let data = {
                id: response.id,
                vqts: response.vqts
            };
            if (response.id == this.options.readTag) {
                var writeToPLC = (response.vqts[0].v);
                let actionData = new cgp_core_1.Ia.ActionData();

```

```
let captureOption = this.options.writeType
let TagDataType = 4;
switch (captureOption) {
  case 'SINT':
    {
      TagDataType = 16
      break;
    }
  case 'INT':
    {
      TagDataType = 2
      break;
    }
  case 'DINT':
    {
      TagDataType = 3
      break;
    }
  case 'LINT':
    {
      TagDataType = 20
      break;
    }
  case 'REAL':
    {
      TagDataType = 4
      break;
    }
  case 'DREAL':
    {
      TagDataType = 5
      break;
    }
  case 'STRING':
    {
      TagDataType = 8
      break;
    }
}
```

```

        case 'BOOL':
            {
                TagDataType = 11
                break;
            }
        default:
            break;
    }
    let writeOptions = {
        mimeType: TagDataType,
        value: writeToPLC
    };
    actionData.add(this.options.writeTag,
cgp_core_1.Ia.ActionType.Write, this.options.writeUrl,
        this.options.contextPrefix,
this.options.trackingId).actionOptions = writeOptions;
        this.application.sendRequest(actionData);
    }
    next(null, appData);

});

}
}
exports.writeToPlc = writeToPlc;
//# sourceMappingURL=startware.js.map

```

## 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 `cp_require`. See the example that follows:

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

- 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);
});
```

- 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:

- Click the [  ] icon on the Pipeline page.

**Figure 6-7: Import Pipeline**



- Enter the Application (Pipeline) name.

**Figure 6-8: Define Import Pipeline**

Import JSON-LD

Application Name : \* pipeline1

Import JSON-LD by selecting file : \* pipeline1.json

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

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-adapter\\"}], \\"@graph\\": [ {\\"@id\\": \\"_:metadata\\", \\"@graph\\": [ {\\"@type\\": \\"metadata\\", \\"@id\\": \\"_:bjflrpeyh\\", \\"updateRateMs\\": 1000, \\"updateType\\": 1, \\"actionOptions\\": {\\"credentialsId\\": \\"1\\", \\"tags\\": [\\"Tag2\\"], \\"mapping\\": {\\"id\\": \\"tag\\", \\"v\\": \\"tagvalue\\", \\"q\\": \\"status\\", \\"t\\": \\"DataTimeStamp\\"}, \\"qualityType\\": \\"csm\\"}, \\"reactionOptions\\": {\\"pipe\\": [\\"pipe1\\"]} } ], {\\"@id\\": \\"_:actions\\", \\"@graph\\": [ {\\"@type\\": \\"action\\", \\"id\\": \\"ra-ctxjflrpeyg:dbo.GetHistoricalDataAllTags\\", \\"actionType\\": 0, \\"actionOptions\\": \\"actionOptions\\", \\"reactionOptions\\": \\"reactionOptions\\", \\"trackingId\\": \\"track1x\\", \\"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
        }
    ]
}
]
}
}

```

This page has been intentionally left blank



## Appendix

# A

## Read Data from Edge in DataFlowML

### In this chapter:

- ❑ Create and Deploy Pipeline in Edge 101

## Overview

### Prerequisite:

MQTT Broker

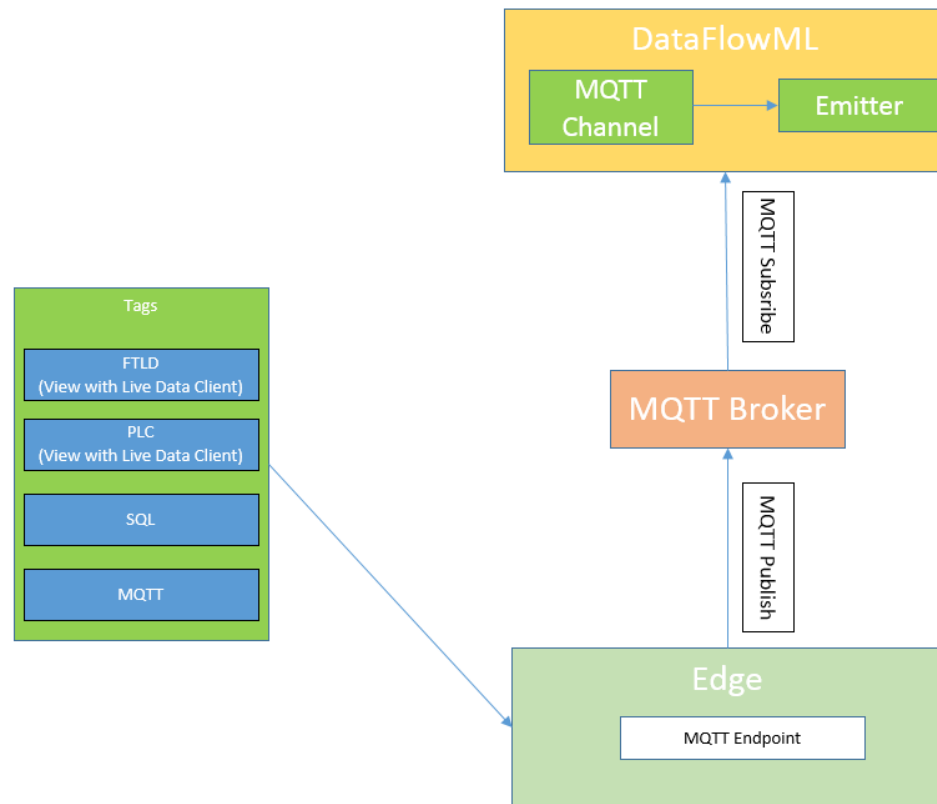
FactoryTalk Analytics DataFlowML

This chapter provides the steps to read the data from Edge in DataFlowML.

Edge reads data from multiple data sources such as FactoryTalk Live Data, SQL, CIP, MQTT and can Egress processed data to MQTT topic. DataFlowML reads the messages from the MQTT topic.

MQTT Emitter in Edge is a publisher, so there should be a subscriber listening on the same topic in order to receive the messages that Edge publishes.

**Figure 6-9: Flow Chart**



For a high-level overview of the interactions between the two products, see the following steps:

1. Create an Edge Pipeline that has a MQTT Emitter.
2. Deploy the Edge Pipeline.
3. Create a MQTT connection in DataFlowML.

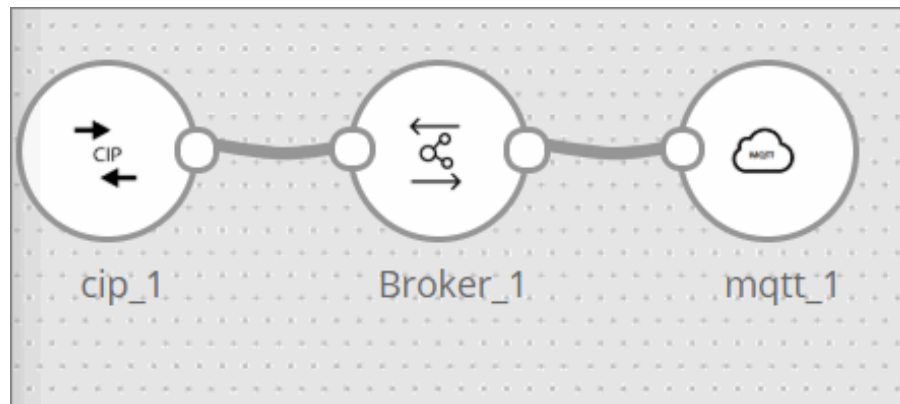
4. Create a Pipeline in DataFlowML using the MQTT channel.
5. Start the DataFlowML Pipeline.

## Create and Deploy Pipeline in Edge

Perform the following steps to create and deploy a Pipeline that has an MQTT Emitter:

1. Log in to the Edge application. Register an Ingress Channel. For example CIP.
2. Register an MQTT Endpoint using the details of MQTT broker.
3. Navigate to the Pipelines page and click the [ + ] icon to create a Pipeline.
4. Click the Adapter, Broker, and the MQTT Emitter to add them to the visual designer. Connect the components.

**Figure 6-10: Pipeline**



5. Right-click the CIP adapter and select the Pipe Path. Click [SAVE].

**Figure 6-11: Configure CIP**

The image shows a 'Configuration Settings' window for 'cip\_1'. At the top, there is an orange bar with the text 'Select Source'. Below this, the following configuration details are listed:

- Tag Name : \* realRandom
- Read Option : \* Continuous Read (dropdown menu)
- CIP Device Shortcut (Path) : \* ra-clx://cgp-cip-adapter/10.85.104.110/
- Tracking ID : 123
- Update Frequency (ms) : \* 1000
- Update Type : \* Polling (dropdown menu)

At the bottom right of the window, there are two buttons: 'CANCEL' and 'SAVE'.

6. Right-click the Broker and click [Save].
7. Right-click the MQTT Emitter and click [SAVE].

**Figure 6-12: Configure MQTT Endpoint**

Configuration Settings - mqtt\_1

Select Source

Post IP/ Host Name : \* 10.85.116.56

Post Port : \* 1883

Topic Name : \* Flow\_Test\_Topic

Clean Session : \* ☒ true ☐ false

Client ID : abc

Quality of Service : \* At most once

CANCEL SAVE

8. Save the Pipeline.
9. Deploy the Pipeline on an Edge Runtime Node. Sample Egress data:

```
[{"@type":"reaction","id":"realRandom","vqts":[{"v":70,"q":192,"t":"2019-01-12T01:50:44.904Z"}],"reactionType":0,"mimeType":"x-ra/cip/real","trackingId":"123","reactionOptions":{},"metadata":{"@type":"metadata","@id":"_bjqstddwx","updateRateMs":1000,"updateType":0,"reactionOptions":{"pipe":["cipingress_file_1_path-607"]},"links":[{"@id":"_bjqstddwx"}],"contextPrefix":"ra-clxjqstddwx","contextUri":"ra-clx://cgp-cip-adapter/10.85.104.110/1:5"}]
```

## Create and Start Pipeline in DataFlowML

Perform the following steps to create and start a Pipeline that has an MQTT Channel:

1. Log in to the DataFlowML application as a Superadmin User. Go to Connections page and click [Add Connection].
2. Select MQTT from the Component Type drop-down list and provide the following details required for creating the connection.

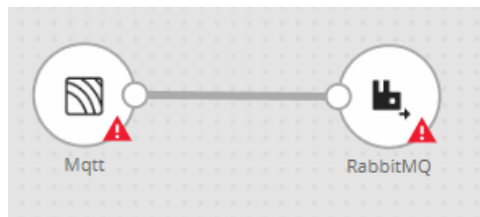
| Field          | Description   |
|----------------|---|
| Component Type | Shows all different types of available connections. |



|                        |   |
|------------------------|---|
| <b>Connection Name</b> | Name of the connection to be created.   |
| <b>Host</b>            | IP address of the machine where Socket is running.  |
| <b>Port</b>            | Port of the machine where Socket is running.  |
| <b>Test Connection</b> | <p>After entering all the details, click on the <b>Test Connection</b> button, if credentials provided are correct, services are up, and running, you will get the message <b>Connection is available</b>.</p> <p>If you enter wrong credentials or server is down and you click on Test Connection, you will get the message Connection unavailable.</p> |

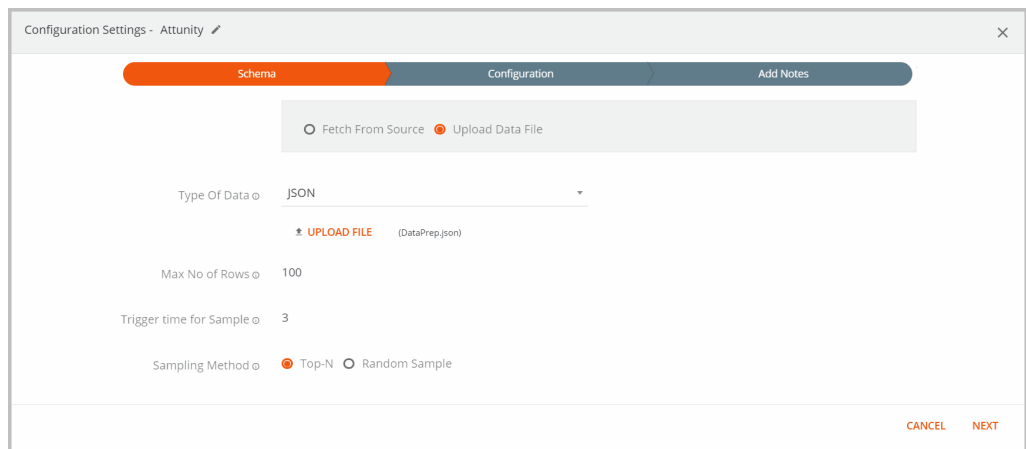
3. Log in to DataFlowML as an Admin/Developer User.
4. Go to Data Pipeline page. Drag the MQTT Channel and an Emitter (for example RabbitMQ) on to the pipeline builder (or canvas) from the right panel and connect the components.

**Figure 6-13: DataFlowML Pipeline**



5. Right click the channel to configure. Under the Schema tab, select JSON as your Type of Data.

**Figure 6-14: JSON**



6. Under the Configuration tab, define the following fields:

| Field                      | Description  |
|----------------------------|--|
| <b>Connection Name</b>     | Connections are the service identifiers. Select the connection name from the available list of connections, from where user would like to read the data. |
| <b>QueueName/TopicName</b> | Queue/topic name from which messages will be read.   |
| <b>Add Configuration</b>   | To add additional properties in key-value pairs.   |

**Figure 6-15: Configuration tab**

Configuration Settings - Mqtt

Schema Schema Configuration Add Notes

Connection Name \* MQTT\_AUTO

Queue Name/Topic Name \* sampleTopic

+ ADD CONFIGURATION

CANCEL PREVIOUS NEXT

**NOTE:** Ensure that the topic name is same that Edge publishes in order to receive the messages.

- Click the Add Notes tab, enter the notes in the space provided and click [Save].
- Right click the emitter to configure it as explained below:

**Figure 6-16: RabbitMQ Configuration Settings**

Configuration Settings - RabbitMQ

Configuration Add Notes

Connection Name + ⓘ Default

Exchange Name + ⓘ Rab\_Exchange

Exchange Type + ⓘ DIRECT

Exchange Durable + ⓘ TRUE

Routing Key + ⓘ Rab\_Key

Queue Name + ⓘ Rab\_queue

Queue Durable + ⓘ TRUE

Encrypt ⓘ ☐

Output Format + ⓘ json

Output Fields + ⓘ x c1 x c2

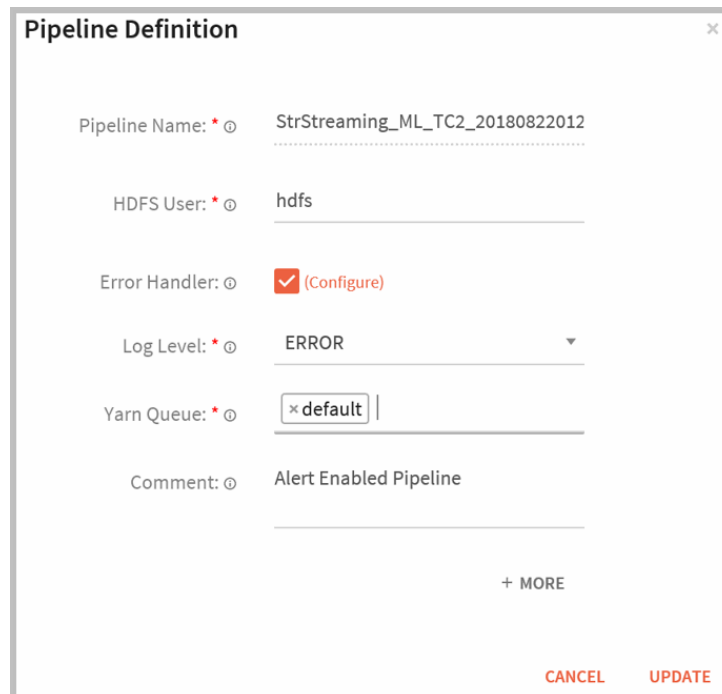
CANCEL NEXT

| Field                   | Description   |
|-------------------------|---|
| <b>Connection Name</b>  | All RabbitMQ connections will be listed here. Select a connection for connecting to the RabbitMQ server.  |
| <b>Exchange Name</b>    | RabbitMQ Exchange name  |
| <b>Exchange Type</b>    | Specifies how messages are routed through it.<br><b>Direct:</b> Delivers message to queues based on a message routing key.<br><b>Fanout:</b> Routes message to all of the queues that are bound to it.<br><b>Topic:</b> Does a wildcard match between the routing key and the routing pattern specified in the binding. |
| <b>Exchange Durable</b> | Specifies whether exchange will be deleted or remain active on server restart.<br><b>TRUE:</b> The exchange will not be deleted if user restart RabbitMQ.<br><b>FALSE:</b> The exchange will be deleted if user restart RabbitMQ.   |
| <b>Routing Key</b>      | Select RabbitMQ routing Key where data will be published.   |

| Field                     | Description   |
|---------------------------|---|
| <b>Queue Name</b>         | RabbitMQ queue name where data will be published.   |
| <b>Queue Durable</b>      | Specifies whether queue will remain active or deleted on server restart.<br><br><b>TRUE:</b> the queue will not be deleted if user restart RabbitMQ.<br><br><b>FALSE:</b> the queue will be deleted if user restart RabbitMQ. |
| <b>Output Format</b>      | Data type format of the output.   |
| <b>Output Fields</b>      | Fields of the output message.   |
| <b>Enable Message TTL</b> | If enabled, message will be discarded to TTL exchange specified.  |
| <b>Message TTL</b>        | Time to live in seconds after which message will be discarded to TTL Exchange specified.  |
| <b>TTL Exchange</b>       | The exchange to which message will be sent once time to live expires.   |
| <b>TTL Queue</b>          | The queue on which message will be sent once time to live expires.  |
| <b>TTL Routing Key</b>    | The routing key used to bind TTL queue with TTL exchange.   |

| Field                    | Description  |
|--------------------------|--|
| <b>Output Mode</b>       | <p>Output mode to be used while writing the data to Streaming sink.</p> <p>Select the output mode from the following:</p> <p><b>Append Mode:</b> This is the default mode, where only the new rows added to the Result Table since the last trigger will be delivered to the emitter. This is supported for only those queries where rows added to the Result Table are never going to change. Hence, this mode guarantees that each row will be output only once.</p> <p><b>Complete:</b> The whole Result Table will be delivered to the sink after every trigger. This is supported for aggregation queries.</p> <p><b>Update:</b> Only the rows in the Result Table that were sent since the last trigger will be delivered to the sink.</p> |
| <b>Enable Trigger</b>    | Trigger defines how frequently a streaming query should be executed.   |
| <b>Processing Time</b>   | Trigger time interval in minutes or seconds.   |
| <b>Add Configuration</b> | Enables to configure additional RabbitMQ properties.   |

9. Click [Next], enter the notes in the space provided and click [SAVE].
10. Register an MQTT Endpoint using the details of MQTT broker.
11. Once the entire configuration is done, click the floppy icon to save the pipeline.
12. Enter the details under Pipeline Definition (shown below).

**Figure 6-17: Pipeline Definition**

The screenshot shows a 'Pipeline Definition' dialog box with the following fields and values:

- Pipeline Name:** StrStreaming\_ML\_TC2\_20180822012
- HDFS User:** hdfs
- Error Handler:** ☒ (Configure)
- Log Level:** ERROR
- Yarn Queue:** × default
- Comment:** Alert Enabled Pipeline

At the bottom right of the dialog, there are two buttons: **CANCEL** and **UPDATE**. A '+ MORE' link is also visible above the buttons.

13. Click [SAVE] to save the Pipeline.
14. Click the [START] button on the pipeline card.
15. Verify the data in the data sink (RabbitMQ queue).