

---

# **Fledge Documentation**

**Dianomic Systems**

**Sep 15, 2022**







# CONTENTS

<b>1</b>	<b>Introduction to Fledge</b>	<b>1</b>
1.1	Typical Use Cases . . . . .	1
1.2	Architectural Overview . . . . .	2
1.3	No-code/Low-code Development . . . . .	2
<b>2</b>	<b>Quick Start Guide</b>	<b>3</b>
2.1	Installing Fledge . . . . .	3
2.2	Starting and stopping Fledge . . . . .	6
2.3	Troubleshooting Fledge . . . . .	6
2.4	Running the Fledge GUI . . . . .	6
2.5	Managing Data Sources . . . . .	8
2.6	Viewing Data . . . . .	11
2.7	Sending Data to Other Systems . . . . .	15
2.8	PI Web API OMF Endpoint . . . . .	16
2.9	Edge Data Store OMF Endpoint . . . . .	19
2.10	OSIsoft Cloud Services OMF Endpoint . . . . .	22
2.11	PI Connector Relay . . . . .	24
2.12	Number Format Hints . . . . .	31
2.13	Integer Format Hints . . . . .	31
2.14	Type Name Hints . . . . .	31
2.15	Type Hint . . . . .	32
2.16	Tag Name Hint . . . . .	32
2.17	Datapoint Specific Hint . . . . .	32
2.18	Asset Framework Location Hint . . . . .	32
2.19	Adding OMF Hints . . . . .	33
2.20	Backing up and Restoring Fledge . . . . .	33
2.21	Troubleshooting and Support Information . . . . .	34
2.22	Package Uninstallation . . . . .	35
<b>3</b>	<b>Processing Data</b>	<b>37</b>
3.1	Why Use Filters? . . . . .	37
3.2	What Can Be Done? . . . . .	37
3.3	Where Can it Be Done? . . . . .	38
3.4	Some Useful Filters . . . . .	48
<b>4</b>	<b>Fledge Architecture</b>	<b>49</b>
4.1	Fledge Core . . . . .	50
4.2	Storage Layer . . . . .	50
4.3	South Microservices . . . . .	50
4.4	North Microservices . . . . .	50



4.5	Filters . . . . .	51
4.6	Event Service . . . . .	52
4.7	Set Point Control Service . . . . .	52
4.8	REST API . . . . .	52
4.9	Graphical User Interface . . . . .	52
<b>5</b>	<b>Buffering &amp; Storage</b>	<b>53</b>
5.1	Configuring The Storage Plugin . . . . .	54
5.2	Installing A PostgreSQL server . . . . .	56
5.3	SQLite Plugin Configuration . . . . .	56
5.4	Storage Management . . . . .	58
<b>6</b>	<b>Additional Services</b>	<b>61</b>
6.1	Notifications Service . . . . .	61
<b>7</b>	<b>Fledge Control Features</b>	<b>79</b>
7.1	Control Functions . . . . .	79
7.2	Control Paths . . . . .	79
7.3	Control Dispatcher Service . . . . .	91
<b>8</b>	<b>Plugin Documentation</b>	<b>109</b>
8.1	Fledge South Plugins . . . . .	109
8.2	Fledge North Plugins . . . . .	173
8.3	Fledge Filter Plugins . . . . .	210
8.4	Fledge Notification Rule Plugins . . . . .	246
8.5	Fledge Notification Delivery Plugins . . . . .	253
<b>9</b>	<b>Developing Data Pipelines</b>	<b>279</b>
9.1	Best Practices . . . . .	279
<b>10</b>	<b>Securing Fledge</b>	<b>291</b>
10.1	Enabling HTTPS Encryption . . . . .	291
10.2	Requiring User Login . . . . .	293
10.3	User Management . . . . .	299
10.4	Certificate Store . . . . .	303
<b>11</b>	<b>Tuning Fledge</b>	<b>307</b>
11.1	South Service Advanced Configuration . . . . .	307
11.2	North Advanced Configuration . . . . .	309
11.3	Health Monitoring . . . . .	310
11.4	Scheduler . . . . .	311
11.5	Storage . . . . .	311
<b>12</b>	<b>Troubleshooting the PI Server integration</b>	<b>317</b>
12.1	Log files . . . . .	317
12.2	How to check the PI Web API is installed and running . . . . .	318
12.3	Commands to check the PI WEB API . . . . .	319
12.4	Error messages and causes . . . . .	322
12.5	OMF Plugin Data . . . . .	322
12.6	Possible solutions to common problems . . . . .	326
<b>13</b>	<b>Plugin Developer Guide</b>	<b>329</b>
13.1	Source Code Documentation . . . . .	329
13.2	Plugins . . . . .	329
13.3	Representing Data . . . . .	332



13.4	Writing and Using Plugins . . . . .	332
13.5	South Plugins . . . . .	341
13.6	South Plugins in C . . . . .	352
13.7	C++ Support Classes . . . . .	363
13.8	Hybrid Plugins . . . . .	372
13.9	North Plugins . . . . .	373
13.10	Storage Plugins . . . . .	382
13.11	Filter Plugins . . . . .	385
13.12	Notification Delivery Plugins . . . . .	399
13.13	Plugin Packaging . . . . .	403
13.14	Testing Your Plugin . . . . .	408
13.15	Developing with Windows Subsystem for Linux (WSL2) . . . . .	416
<b>14</b>	<b>REST API Developers Guide</b>	<b>423</b>
14.1	The Fledge REST API . . . . .	423
14.2	Administration API Reference . . . . .	424
14.3	User API Reference . . . . .	442
<b>15</b>	<b>Building Fledge</b>	<b>447</b>
15.1	Building Developers Guide . . . . .	447
15.2	Building and using Fledge on Raspbian . . . . .	487
<b>16</b>	<b>OMF Kerberos Authentication</b>	<b>489</b>
16.1	Introduction . . . . .	489
16.2	PI Server as the North endpoint . . . . .	489
16.3	North plugin . . . . .	489
16.4	Fledge server configuration . . . . .	490
<b>17</b>	<b>Fledge Plugins</b>	<b>493</b>
17.1	South Plugins . . . . .	493
17.2	North Plugins . . . . .	494
17.3	Filter Plugins . . . . .	495
17.4	Notification Rule Plugins . . . . .	496
17.5	Notification Delivery Plugins . . . . .	496
<b>18</b>	<b>Version History</b>	<b>497</b>
18.1	Fledge v2 . . . . .	497
18.2	Fledge v1 . . . . .	504
<b>19</b>	<b>Downloads</b>	<b>531</b>
19.1	Packages . . . . .	531
19.2	Download/Clone from GitHub . . . . .	531
<b>20</b>	<b>Glossary</b>	<b>533</b>
	<b>Index</b>	<b>535</b>







## INTRODUCTION TO FLEDGE

Fledge is an open Industrial IoT system designed to make collecting, filtering, processing and using operational data simpler and more open. Core to Fledge is an extensible microservice based architecture enabling any data to be read, processed and sent to any system. Coupled with this extensibility Fledge's Apache 2 license and community of developers results in an ever growing choice of components that can be used to solve your OT data needs well into the future.

Fledge provides a scalable, secure, robust infrastructure for collecting data from sensors, processing data at the edge using intelligent data pipelines and transporting data to historian and other management systems. Fledge also allows for edge based event detection and notification and control flows as a result of events, stimulus from upstream systems or user action. Fledge can operate over the unreliable, intermittent and low bandwidth connections often found in industrial or rugged environments.

### 1.1 Typical Use Cases

The depth and breadth of Industrial IoT use cases is considerable. Fledge is designed to address them. Below are some examples of typical Fledge deployments.

**Unified data collection** The industrial edge is one of the more challenging in computing. Today there are over 100 different protocols, no standards in machine data definitions, different types of data (time-series, vibration, array, image, radiometric, transactional, etc.), sensors producing bytes/hr to gigs/hr all in environments with network, power and environmental challenges. This diversity creates pain in managing, scaling, securing and orchestrating industrial data. Ultimately resulting in silos of data with competing context. Fledge is designed to eliminate those silos by providing a very flexible data collections and distribution mechanism all using the same APIs, features and functions.

**Specialized Analytical Environments** With the advent of cloud systems and sophisticated analytic tools it may no longer be possible to have a single system that is both your system of record and the place on which the analytics takes place. Fledge allows you to distribute your data to multiple systems, either in part or as a whole. This allows you to get just the data you need to the systems that need it without compromising your system of record.

**Resilience** Fledge provides mechanisms to store and forward your data. Data is no longer lost if a connection to some key system is unavailable.

**Edge processing** Using the Fledge intelligent data pipelines concept, Fledge allows for your data to be processed close to where it is gathered. This can save both network bandwidth and reduce costs when high bandwidth sensors such as vibration monitors or image capture is used. In addition it reduces the latency when timely action is required compared with shipping and processing data in the cloud or at some centralized IT location.

**No code/Low code solutions** Fledge provides tools that allow the OT engineer to create solutions by use of existing processing elements that can be combined and augmented with little or no coding required. This allows the OT organization to be able to quickly and independently obtain the data they need for their specific requirements.



**Process Optimization & Operational Efficiency** The Fledge intelligent pipelines, with their prebuilt processing elements and through use of machine learning techniques can be used to improve operational efficiency by giving operators immediate feedback on the state of the process of product being produced without remote analytics and the associated delays involved.

## 1.2 Architectural Overview

Fledge is implemented as a collection of microservices which include:

- Core services, including security, monitoring, and storage
- Data transformation and alerting services
- South services: Collect data from sensors and other Fledge systems
- North services: Transmit and integrate data to historians and other systems
- Edge data processing applications
- Event detection and notification
- Set point control

Services can easily be developed and incorporated into the Fledge framework. Fledge services may also be customized by creating new plugins, written in C/C++ or Python, for data collection, processing, export, rule evaluation and event notification. The describe how to do this.

More detail on the Fledge architecture can be found in the section .

## 1.3 No-code/Low-code Development

Fledge can be extended by writing code to add new plugins. Additionally, it is easily tailored by combining pre-written data processing filters applied in linear pipelines to data as it comes into or goes out of the Fledge system. A number of filters exist that can be customized with small snippets of code written in the Python scripting language. These snippets of code allow the end user to produce custom processing without the need to develop more complex plugins or other code. The environment also allows them to experiment with these code snippets to obtain the results desired.

Data may be processed on the way into Fledge or on the way out. Processing on the way in allows the data to be manipulated to the way the organization wants it. Processing on the way out allows the data to be manipulate to suit the up stream system that will use the data without impacting the data that might go to another up stream system.

See the section .



## QUICK START GUIDE

### 2.1 Installing Fledge

Fledge is extremely lightweight and can run on inexpensive edge devices, sensors and actuator boards. For the purposes of this manual, we assume that all services are running on a Raspberry Pi running the Raspbian operating system. Be sure your system has plenty of storage available for data readings.

If your system does not have Raspbian pre-installed, you can find instructions on downloading and installing it at <https://www.raspberrypi.org/downloads/raspbian/>. After installing Raspbian, ensure you have the latest updates by executing the following commands on your Fledge server:

```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get update
```

You can obtain Fledge in three ways:

- Dianomic Systems hosts a package repository that allows the Fledge packages to be loaded using the system package manager. This is the recommended method for long term use of Fledge as it gives access to all the Fledge plugins and provides a route for easy upgrade of the Fledge packages. This also has the advantages that once the repository is configured you are able to install new plugins directly from the Fledge user interface without the need to resort to the Linux command line.
- Dianomic Systems offers pre-built, certified binaries of Fledge for Debian using either Intel or ARM architectures. This is perhaps the simplest method for users not used to Linux. You can download the complete set of packages from <http://dianomic.com/download-fledge/>.
- As source code from <https://github.com/fledge-iot/>. Instructions for downloading and building Fledge source code can be found in the Fledge Developer's Guide

In general, Fledge installation will require the following packages:

- Fledge core
- Fledge user interface
- One or more Fledge South services
- One or more Fledge North service (OSI PI and OCS north services are included in Fledge core)



### 2.1.1 Using the package repository to install Fledge

If you choose to use the Dianomic Systems package repository to install the packages you will need to follow the steps outlined below for the particular platform you are using.

#### Ubuntu or Debian

On a Ubuntu or Debian system, including the Raspberry Pi, the package manager that is supported is *apt*. You will need to add the Dianomic Systems archive server into the configuration of apt on your system. The first thing that must be done is to add the key that is used to verify the package repository. To do this run the command

```
wget -q -O - http://archives.fledge-iot.org/KEY.gpg | sudo apt-key add -
```

Once complete you can add the repository itself into the apt configuration file `/etc/apt/sources.list`. The simplest way to do this is the use the *add-apt-repository* command. The exact command will vary between systems;

- Raspberry Pi does not have an *apt-add-repository* command, the user must edit the apt sources file manually

```
sudo vi /etc/apt/sources.list
```

and add the line

```
deb http://archives.fledge-iot.org/latest/buster/armv7l/ /
```

to the end of the file.

---

**Note:** Replace *buster* with *stretch* or *bullseye* based on the OS image used.

---

- Users with an Intel or AMD system with Ubuntu 18.04 should run

```
sudo add-apt-repository "deb http://archives.fledge-iot.org/latest/ubuntu1804/x86_
↳64/ / "
```

- Users with an Intel or AMD system with Ubuntu 20.04 should run

```
sudo add-apt-repository "deb http://archives.fledge-iot.org/latest/ubuntu2004/x86_
↳64/ / "
```

---

**Note:** We do not support the *aarch64* architecture with Ubuntu 20.04 yet.

---

- Users with an Arm system with Ubuntu 18.04, such as the Odroid board, should run

```
sudo add-apt-repository "deb http://archives.fledge-iot.org/latest/ubuntu1804/
↳aarch64/ / "
```

- Users of the Mendel operating system on a Google Coral create the file `/etc/apt/sources.list.d/fledge.list` and insert the following content

```
deb http://archives.fledge-iot.org/latest/mendel/aarch64/ /
```

Once the repository has been added you must inform the package manager to go and fetch a list of the packages it supports. To do this run the command



```
sudo apt -y update
```

You are now ready to install the Fledge packages. You do this by running the command

```
sudo apt -y install *package*
```

You may also install multiple packages in a single command. To install the base fledge package, the fledge user interface and the sinusoid south plugin run the command

```
sudo DEBIAN_FRONTEND=noninteractive apt -y install fledge fledge-gui fledge-south-
↪sinusoid
```

## 2.1.2 Installing Fledge downloaded packages

Assuming you have downloaded the packages from the download link given above. Use SSH to login to the system that will host Fledge services. For each Fledge package that you choose to install, type the following command:

```
sudo apt -y install PackageName
```

The key packages to install are the Fledge core and the Fledge User Interface:

```
sudo DEBIAN_FRONTEND=noninteractive apt -y install ./fledge-1.8.0-armv7l.deb
sudo apt -y install ./fledge-gui-1.8.0.deb
```

You will need to install one of more South plugins to acquire data. You can either do this now or when you are adding the data source. For example, to install the plugin for the Sense HAT sensor board, type:

```
sudo apt -y install ./fledge-south-sensehat-1.8.0-armv7l.deb
```

You may also need to install one or more North plugins to transmit data. Support for OSIsoft PI and OCS are included with the Fledge core package, so you don't need to install anything more if you are sending data to only these systems.

## 2.1.3 Checking package installation

To check what packages have been installed, ssh into your host system and use the dpkg command:

```
dpkg -l | grep 'fledge'
```

## 2.1.4 Run with PostgreSQL

To start Fledge with PostgreSQL, first you need to install the PostgreSQL package explicitly. See the below links for setup

Also you need to change the value of Storage plugin. See or with below curl command

```
$ curl -sX PUT localhost:8081/fledge/category/Storage/plugin -d '{"value": "postgres"}'
↪'
{
  "description": "The main storage plugin to load",
  "type": "string",
  "order": "1",
```

(continues on next page)



(continued from previous page)

```
"displayName": "Storage Plugin",
"default": "sqlite",
"value": "postgres"
}
```

Now, it's time to restart Fledge. Thereafter you will see Fledge is running with PostgreSQL.

## 2.2 Starting and stopping Fledge

Fledge administration is performed using the “fledge” command line utility. You must first ssh into the host system. The Fledge utility is installed by default in `/usr/local/fledge/bin`.

The following command options are available:

- **Start:** Start the Fledge system
- **Stop:** Stop the Fledge system
- **Status:** Lists currently running Fledge services and tasks
- **Reset:** Delete all data and configuration and return Fledge to factory settings
- **Kill:** Kill Fledge services that have not correctly responded to Stop
- **Help:** Describe Fledge options

For example, to start the Fledge system, open a session to the Fledge device and type:

```
/usr/local/fledge/bin/fledge start
```

## 2.3 Troubleshooting Fledge

Fledge logs status and error messages to syslog. To troubleshoot a Fledge installation using this information, open a session to the Fledge server and type:

```
grep -a 'fledge' /var/log/syslog | tail -n 20
```

## 2.4 Running the Fledge GUI

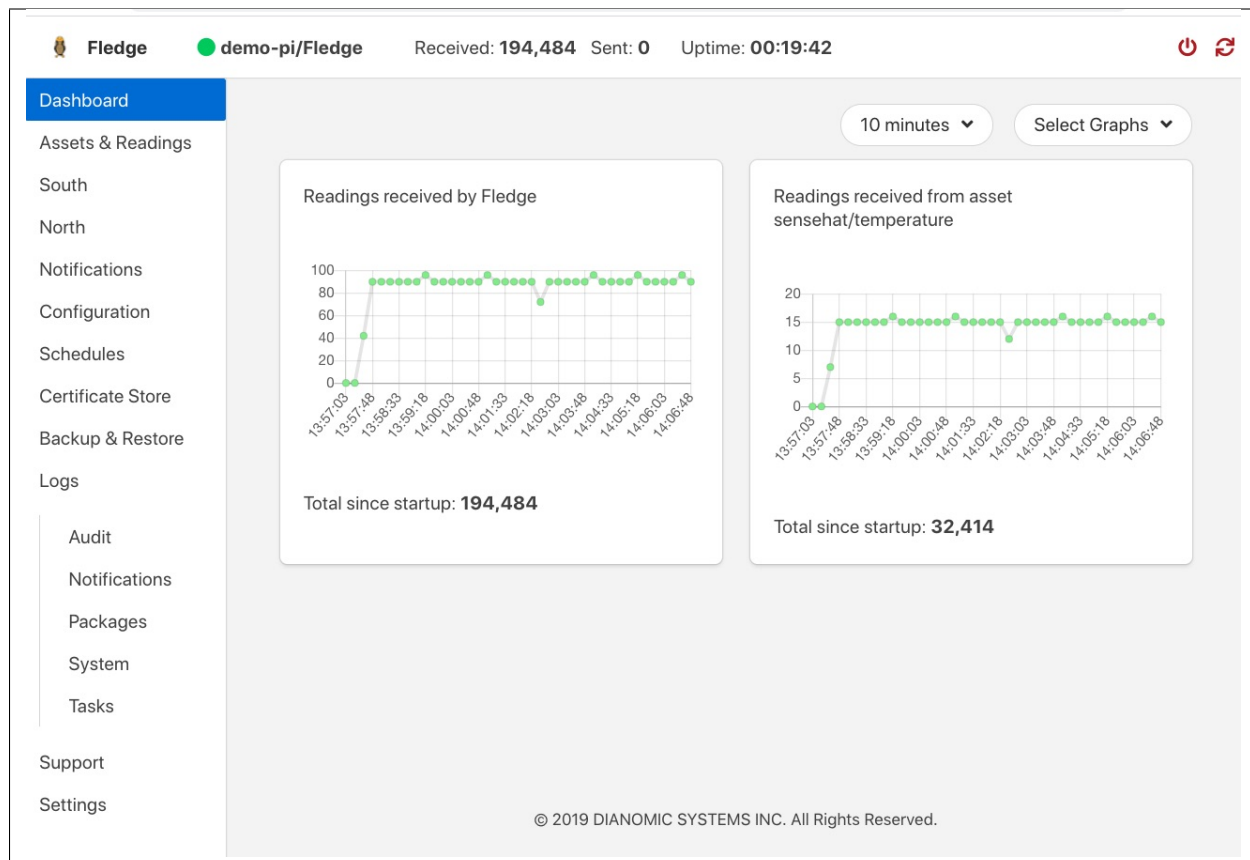
Fledge offers an easy-to-use, browser-based GUI. To access the GUI, open your browser and enter the IP address of the Fledge server into the address bar. This will display the Fledge dashboard.

You can easily use the Fledge UI to monitor multiple Fledge servers. To view and manage a different server, click “Settings” in the left menu bar. In the “Connection Setup” pane, enter the IP address and port number for the new server you wish to manage. Click the “Set the URL & Restart” button to switch the UI to the new server.

If you are managing a very lightweight server or one that is connected via a slow network link, you may want to reduce the UI update frequency to minimize load on the server and network. You can adjust this rate in the “GUI Settings” pane of the Settings screen. While the graph rate and ping rate can be adjusted individually, in general you should set them to the same value.



## 2.4.1 Fledge Dashboard



This screen provides an overview of Fledge operations. You can customize the information and time frames displayed on this screen using the drop-down menus in the upper right corner. The information you select will be displayed in a series of graphs.

You can choose to view a graph of any of the sensor reading being collected by the Fledge system. In addition, you can view graphs of the following system-wide information:

- **Readings:** The total number of data readings collected by Fledge since system boot
- **Buffered:** The number of data readings currently stored by the system
- **Discarded:** Number of data readings discarded before being buffered (due to data errors, for example)
- **Unsent:** Number of data readings that were not sent successfully
- **Purged:** The total number of data readings that have been purged from the system
- **Unsnpurged:** The number of data readings that were purged without being sent to a North service.



## 2.5 Managing Data Sources

Name	Status	Plugin	Version	Assets	Readings
<u>SenseHAT</u>	enabled	sensehat	1.7.0	sensehat/pressure	33,418
				sensehat/temperature	33,418
				sensehat/humidity	33,418
				sensehat/magnetometer	33,418
				sensehat/gyroscope	33,418
				sensehat/accelerometer	33,418

Data sources are managed from the South Services screen. To access this screen, click on “South” from the menu bar on the left side of any screen.

The South Services screen displays the status of all data sources in the Fledge system. Each data source will display its status, the data assets it is providing, and the number of readings that have been collected.

### 2.5.1 Adding Data Sources

To add a data source, you will first need to install the plugin for that sensor type. If you have not already done this, open a terminal session to your Fledge server. Download the package for the plugin and enter:

```
sudo apt -y install PackageName
```

Once the plugin is installed return to the Fledge GUI and click on “Add+” in the upper right of the South Services screen. Fledge will display a series of 3 screens to add the data source:

1. The first screen will ask you to select the plugin for the data source from the list of installed plugins. If you do not see the plugin you need, refer to the Installing Fledge section of this manual. In addition, this screen allows you to specify a display name for the data source.
2. The second screen allows you to configure the plugin and the data assets it will provide.

**Note:** Every data asset in Fledge must have a unique name. If you have multiple sensors using the same plugin,



modify the asset names on this screen so they are unique.

---

Some plugins allow you to specify an asset name prefix that will apply to all the asset names for that sensor. Refer to the individual plugin documentation for descriptions of the fields on this screen.

3. If you modify any of the configuration fields, click on the “save” button to save them.
4. The final screen allows you to specify whether the service will be enabled immediately for data collection or await enabling in the future.





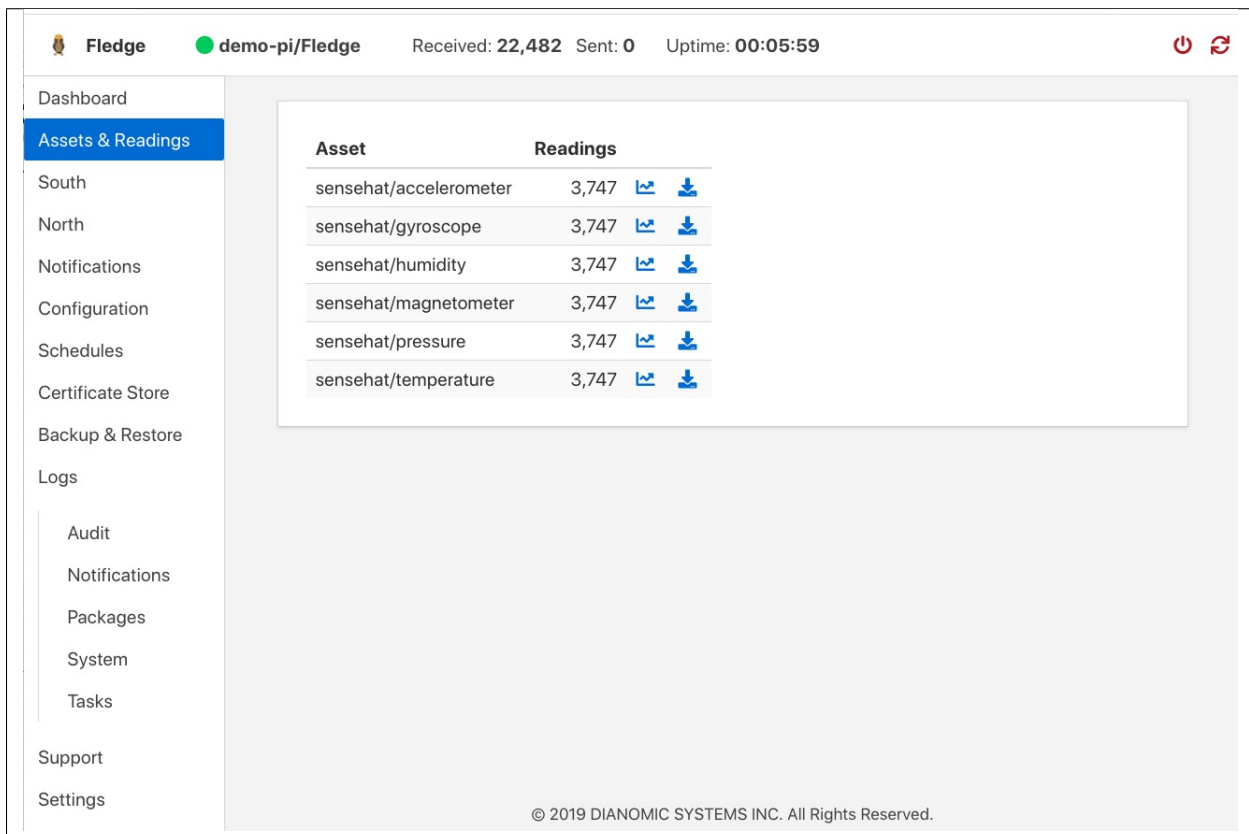


To modify the configuration of a data source, click on its name in the South Services screen. This will display a list of all parameters available for that data source. If you make any changes, click on the “save” button in the top panel to save the new configuration. Click on the “x” button in the upper right corner to return to the South Services screen.

## 2.5.3 Enabling and Disabling Data Sources

To enable or disable a data source, click on its name in the South Services screen. Under the list of data source parameters, there is a check box to enable or disable the service. If you make any changes, click on the “save” button in the bottom panel near the check box to save the new configuration.

## 2.6 Viewing Data



The screenshot shows the Fledge web interface. The top bar includes the Fledge logo, a green status indicator for 'demo-pi/Fledge', and system statistics: 'Received: 22,482', 'Sent: 0', and 'Uptime: 00:05:59'. A left-hand navigation menu lists various system components, with 'Assets & Readings' currently selected. The main content area displays a table titled 'Assets & Readings' with the following data:

Asset	Readings		
sensehat/accelerometer	3,747		
sensehat/gyroscope	3,747		
sensehat/humidity	3,747		
sensehat/magnetometer	3,747		
sensehat/pressure	3,747		
sensehat/temperature	3,747		

At the bottom of the interface, a copyright notice reads: '© 2019 DIANOMIC SYSTEMS INC. All Rights Reserved.'

You can inspect all the data buffered by the Fledge system on the Assets page. To access this page, click on “Assets & Readings” from the left-side menu bar.

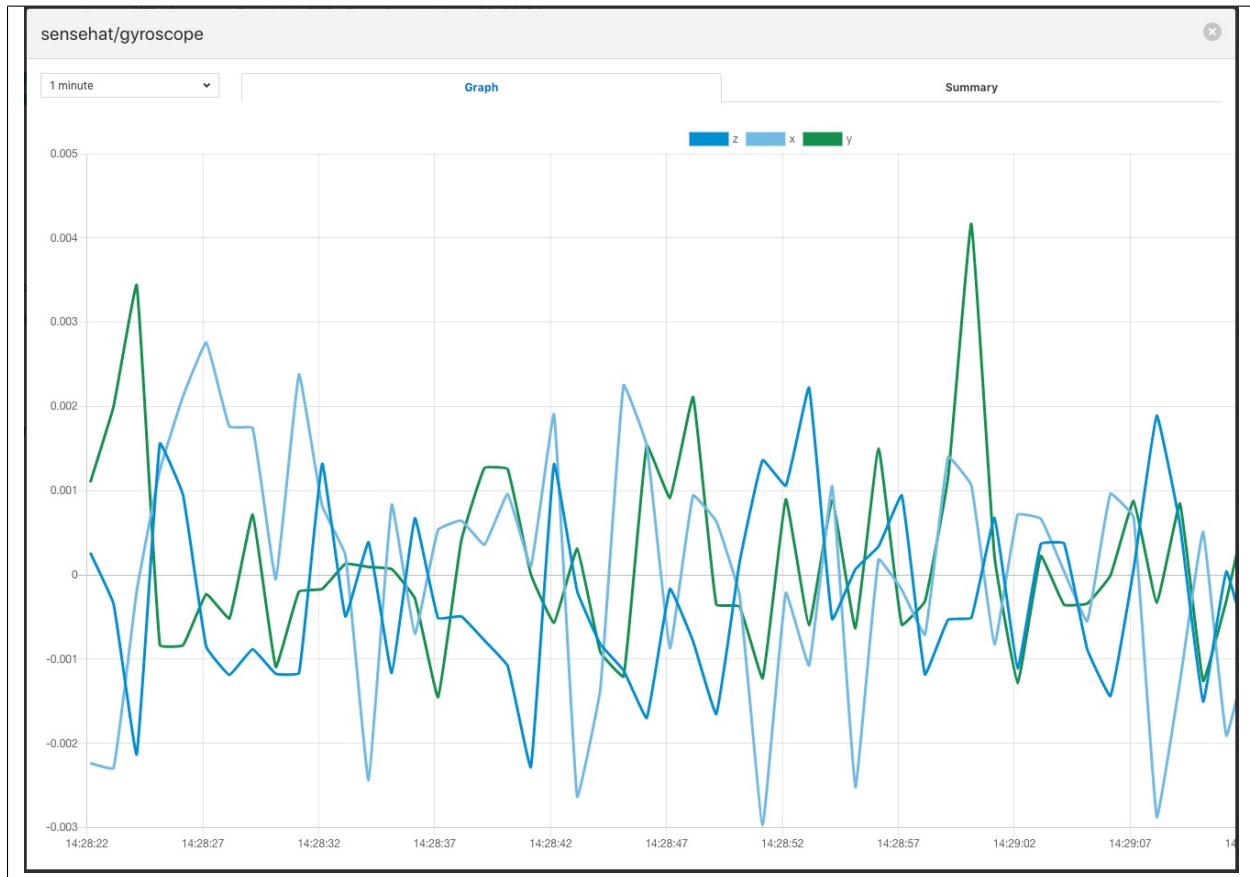
This screen will display a list of every data asset in the system. Alongside each asset are two icons; one to display a graph of the asset and another to download the data stored for that asset as a CSV file.



## 2.6.1 Display Graph

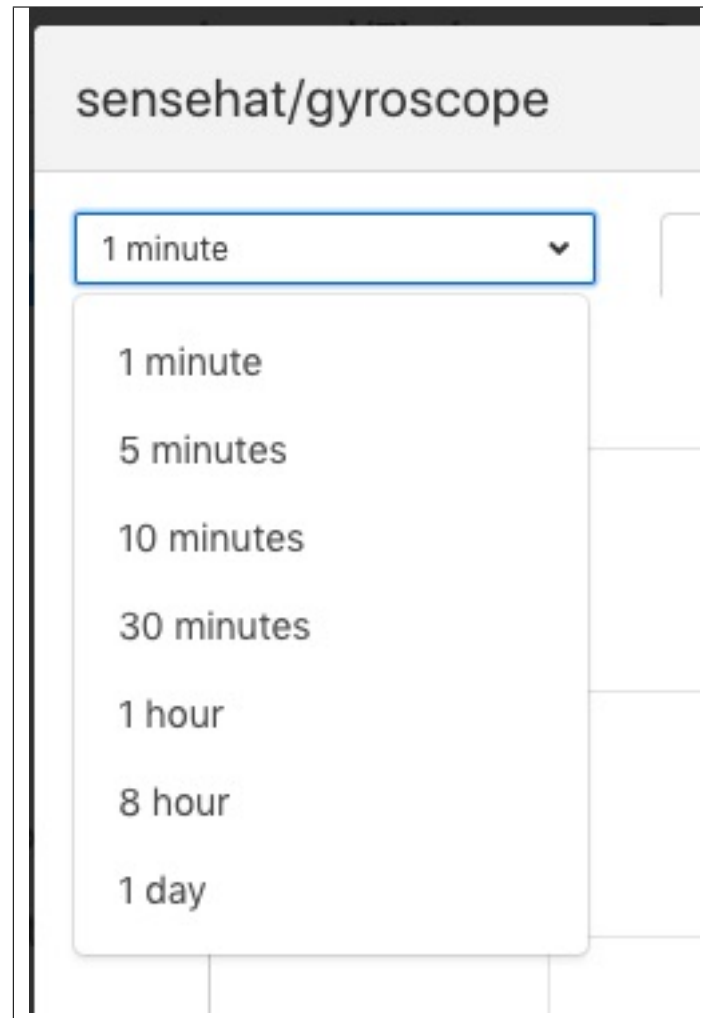


By clicking on the graph button next to each asset name, you can view a graph of individual data readings. A graph will be displayed with a plot for each data point within the asset.



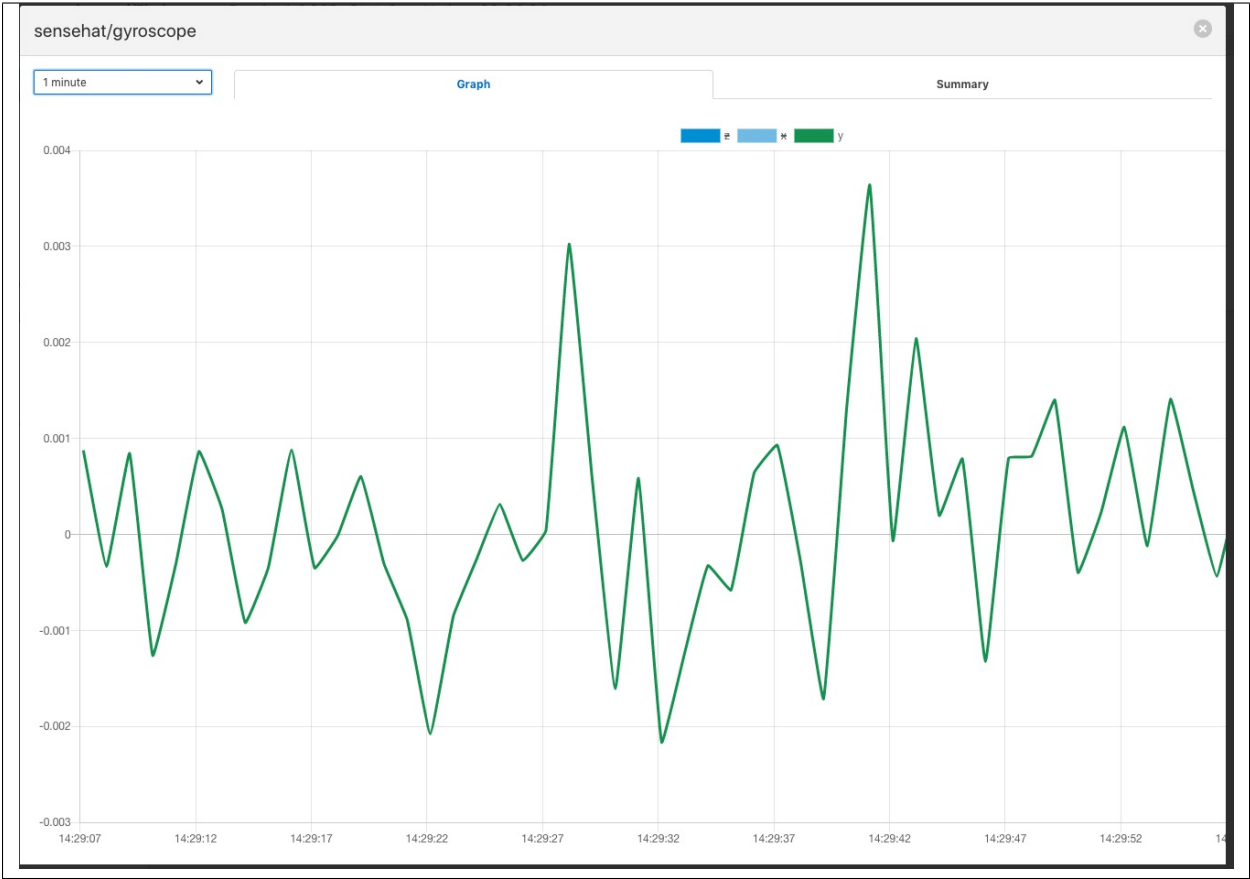
It is possible to change the time period to which the graph refers by use of the plugin list in the top left of the graph.





Where an asset contains multiple data points each of these is displayed in a different colour. Graphs for particular data points can be toggled on and off by clicking on the key at the top of the graph. Those data points not should will be indicated by striking through the name of the data point.





A summary tab is also available, this will show the minimum, maximum and average values for each of the data points. Click on *Summary* to show the summary tab.



## 2.6.2 Download Data



By clicking on the download icon adjacent to each asset you can download the stored data for the asset. The format of the file is download is a CSV file that is designed to be loaded into a spreadsheet such as Excel, Numbers or OpenOffice Calc.




The file contains a header row with the names of the data points within the asset, the first column is always the timestamp when the reading was taken, the header for this being *timestamp*. The data is sorted in chronological order with the newest data first.



## sensehat\_gyroscope-readings

timestamp	z	x	y
2020-05-04 14:30:49.145006	0.000792725	0.0010765493	0.0022465843
2020-05-04 14:30:48.145022	0.0010982286	-0.0004502609	0.000719551
2020-05-04 14:30:47.145006	0.0007928684	0.0032151192	-0.0011130939
2020-05-04 14:30:46.145008	-0.0013448559	0.0047423765	0.0001088944
2020-05-04 14:30:45.145000	-0.0004286431	0.0007723272	-0.0020291833
2020-05-04 14:30:44.144999	-0.0001233947	0.0013834909	0.0007194807
2020-05-04 14:30:43.145001	-0.000734292	-0.0001437888	0.0004143068

## 2.7 Sending Data to Other Systems

 **Fledge**
● demo-pi/Fledge
 Received: 22,938 Sent: 0 Uptime: 00:07:14
 



Dashboard
Assets & Readings
South
**North**
Notifications
Configuration
Schedules
Certificate Store
Backup & Restore
Logs

Audit
Notifications
Packages
System
Tasks

Support
Settings

North Instances
Create North Instance +

Process	Status	Plugin	Version	Sent
<a href="#">Plant Librarian</a>	enabled			0

© 2019 DIANOMIC SYSTEMS INC. All Rights Reserved.

Data destinations are managed from the North Services screen. To access this screen, click on “North” from the menu bar on the left side of any screen.



The North Services screen displays the status of all data sending processes in the Fledge system. Each data destination will display its status and the number of readings that have been collected.

### 2.7.1 Adding Data Destinations

To add a data destination, click on “Create North Instance+” in the upper right of the North Services screen. Fledge will display a series of 3 screens to add the data destination:

1. The first screen will ask you to select the plugin for the data destination from the list of installed plugins. If you do not see the plugin you need, refer to the Installing Fledge section of this manual. In addition, this screen allows you to specify a display name for the data destination. In addition, you can specify how frequently data will be forwarded to the destination in days, hours, minutes and seconds. Enter the number of days in the interval in the left box and the number of hours, minutes and seconds in format HH:MM:SS in the right box.
2. The second screen allows you to configure the plugin and the data assets it will send. See the section below for specifics of configuring a PI, EDS or OCS destination.
3. The final screen loads the plugin. You can specify whether it will be enabled immediately for data sending or to await enabling in the future.

### 2.7.2 Configuring Data Destinations

To modify the configuration of a data destination, click on its name in the North Services screen. This will display a list of all parameters available for that data source. If you make any changes, click on the “save” button in the top panel to save the new configuration. Click on the “x” button in the upper right corner to return to the North Services screen.

### 2.7.3 Enabling and Disabling Data Destinations

To enable or disable a data source, click on its name in the North Services screen. Under the list of data source parameters, there is a check box to enable or disable the service. If you make any changes, click on the “save” button in the bottom panel near the check box to save the new configuration.

### 2.7.4 Using the OMF plugin

OSISoft data historians are one of the most common destinations for Fledge data. Fledge supports the full range of OSISoft historians; the PI System, Edge Data Store (EDS) and OSISoft Cloud Services (OCS). To send data to a PI server you may use either the older PI Connector Relay or the newer PI Web API OMF endpoint. It is recommended that new users use the PI Web API OMF endpoint rather than the Connector Relay which is no longer supported by OSISoft.

## 2.8 PI Web API OMF Endpoint

To use the PI Web API OMF endpoint first ensure the OMF option was included in your PI Server when it was installed.

Now go to the Fledge user interface, create a new North instance and select the “OMF” plugin on the first screen. The second screen will request the following information:



?

Endpoint

PI Web API

Send full structure

☒

Naming Scheme

Concise

Server hostname

localhost

Server port, 0=use the default

0

Producer Token

omf\_north\_0001

Data Source

readings

Static Data

Location: Palo Alto, Company: Dianomic

Sleep Time Retry

1

Maximum Retry

3

HTTP Timeout

10

Integer Format

int64

Number Format

float64

Compression

☒

Default Asset Framework Location

/fledge/data\_piwebapi/default

Asset Framework hierarchy rules

1

{}

PI Web API Authentication Method

anonymous

PI Web API User Id

user\_id

PI Web API Password

\*\*\*\*\*

PI Web API Kerberos keytab file

piwebapi\_kerberos\_https.keytab

OCS Namespace

name\_space

OCS Tenant ID

ocs\_tenant\_id

OCS Client ID

ocs\_client\_id

OCS Client Secret

\*\*\*\*\*



Select PI Web API from the Endpoint options.

- **Basic Information**

- **Endpoint:** This is the type of OMF endpoint. In this case, choose PI Web API.
- **Send full structure:** Used to control if Asset Framework structure messages are sent to the PI Server. If this is turned off then the data will not be placed in the Asset Framework.
- **Naming scheme:** Defines the naming scheme to be used when creating the PI points in the PI Data Archive. See [Naming Scheme](#).
- **Server hostname:** The hostname or address of the PI Web API server. This is normally the same address as the PI Server.
- **Server port:** The port the PI Web API OMF endpoint is listening on. Leave as 0 if you are using the default port.
- **Data Source:** Defines which data is sent to the PI Server. Choices are: readings or statistics (that is, Fledge's internal statistics).
- **Static Data:** Data to include in every reading sent to PI. For example, you can use this to specify the location of the devices being monitored by the Fledge server.

- **Asset Framework**

- **Default Asset Framework Location:** The location in the Asset Framework hierarchy into which the data will be inserted. All data will be inserted at this point in the Asset Framework hierarchy unless a later rule overrides this. Note this field does not include the name of the target Asset Framework Database; the target database is defined on the PI Web API server by the PI Web API Admin Utility.
- **Asset Framework Hierarchies Rules:** A set of rules that allow specific readings to be placed elsewhere in the Asset Framework. These rules can be based on the name of the asset itself or some metadata associated with the asset. See [Asset Framework Hierarchy Rules](#).

- **PI Web API authentication**

- **PI Web API Authentication Method:** The authentication method to be used: anonymous, basic or kerberos. Anonymous equates to no authentication, basic authentication requires a user name and password, and Kerberos allows integration with your single signon environment.
- **PI Web API User Id:** For Basic authentication, the user name to authenticate with the PI Web API.
- **PI Web API Password:** For Basic authentication, the password of the user we are using to authenticate.
- **PI Web API Kerberos keytab file:** The Kerberos keytab file used to authenticate.

- **Connection management (These should only be changed with guidance from support)**

- **Sleep Time Retry:** Number of seconds to wait before retrying the HTTP connection (Fledge doubles this time after each failed attempt).
- **Maximum Retry:** Maximum number of times to retry connecting to the PI Server.
- **HTTP Timeout:** Number of seconds to wait before Fledge will time out an HTTP connection attempt.

- **Other (Rarely changed)**

- **Integer Format:** Used to match Fledge data types to the data type configured in PI. This defaults to int64 but may be set to any OMF data type compatible with integer data, e.g. int32.
- **Number Format:** Used to match Fledge data types to the data type configured in PI. The default is float64 but may be set to any OMF datatype that supports floating point values.



- **Compression:** Compress the readings data before sending them to the PI Web API OMF endpoint. This setting is not related to data compression in the PI Data Archive.

## 2.9 Edge Data Store OMF Endpoint

To use the OSIsoft Edge Data Store first install Edge Data Store on the same machine as your Fledge instance. It is a limitation of Edge Data Store that it must reside on the same host as any system that connects to it with OMF.

Now go to the Fledge user interface, create a new North instance and select the “OMF” plugin on the first screen. The second screen will request the following information:



?

Endpoint

Edge Data Store

Send full structure

☒

Naming Scheme

Concise

Server hostname

localhost

Server port, 0=use the default

0

Producer Token

omf\_north\_0001

Data Source

readings

Static Data

Location: Palo Alto, Company: Dianomic

Sleep Time Retry

1

Maximum Retry

3

HTTP Timeout

10

Integer Format

int64

Number Format

float64

Compression

☒

Default Asset Framework Location

/fledge/data\_piwebapi/default

Asset Framework hierarchy rules

1

{}

PI Web API Authentication Method

anonymous

PI Web API User Id

user\_id

PI Web API Password

.....

PI Web API Kerberos keytab file

piwebapi\_kerberos\_https.keytab

OCS Namespace

name\_space

OCS Tenant ID

ocs\_tenant\_id

OCS Client ID

ocs\_client\_id

OCS Client Secret

.....



Select Edge Data Store from the Endpoint options.

- **Basic Information**

- **Endpoint:** This is the type of OMF endpoint. In this case, choose Edge Data Store.
- **Naming scheme:** Defines the naming scheme to be used when creating the PI points within the PI Server. See [Naming Scheme](#).
- **Server hostname:** Normally the hostname or address of the OMF endpoint. For Edge Data Store, this must be *localhost*.
- **Server port:** The port the Edge Data Store is listening on. Leave as 0 if you are using the default port.
- **Data Source:** Defines which data is sent to the Edge Data Store. Choices are: readings or statistics (that is, Fledge's internal statistics).
- **Static Data:** Data to include in every reading sent to PI. For example, you can use this to specify the location of the devices being monitored by the Fledge server.

- **Connection management (These should only be changed with guidance from support)**

- **Sleep Time Retry:** Number of seconds to wait before retrying the HTTP connection (Fledge doubles this time after each failed attempt).
- **Maximum Retry:** Maximum number of times to retry connecting to the PI server.
- **HTTP Timeout:** Number of seconds to wait before Fledge will time out an HTTP connection attempt.

- **Other (Rarely changed)**

- **Integer Format:** Used to match Fledge data types to the data type configured in PI. This defaults to int64 but may be set to any OMF data type compatible with integer data, e.g. int32.
- **Number Format:** Used to match Fledge data types to the data type configured in PI. The default is float64 but may be set to any OMF datatype that supports floating point values.
- **Compression:** Compress the readings data before sending them to the Edge Data Store.



## 2.10 OSIssoft Cloud Services OMF Endpoint

Go to the Fledge user interface, create a new North instance and select the “OMF” plugin on the first screen. The second screen will request the following information:



Endpoint

OSIsoft Cloud Services

Send full structure

☒

Naming Scheme

Concise

Server hostname

localhost

Server port, 0=use the default

0

Producer Token

omf\_north\_0001

Data Source

readings

Static Data

Location: Palo Alto, Company: Dianomic

Sleep Time Retry

1

Maximum Retry

3

HTTP Timeout

10

Integer Format

int64

Number Format

float64

Compression

☒

Default Asset Framework Location

/fledge/data\_piwebapi/default

Asset Framework hierarchy rules

1

{}

PI Web API Authentication Method

anonymous

PI Web API User Id

user\_id

PI Web API Password

.....

PI Web API Kerberos keytab file

piwebapi\_kerberos\_https.keytab

OCS Namespace

name\_space

OCS Tenant ID

ocs\_tenant\_id

OCS Client ID

ocs\_client\_id

OCS Client Secret

.....



Select OSIsoft Cloud Services from the Endpoint options.

- **Basic Information**

- **Endpoint:** This is the type of OMF endpoint. In this case, choose OSIsoft Cloud Services.
- **Naming scheme:** Defines the naming scheme to be used when creating the PI points within the PI Server. See [Naming Scheme](#).
- **Data Source:** Defines which data is sent to OSIsoft Cloud Services. Choices are: readings or statistics (that is, Fledge’s internal statistics).
- **Static Data:** Data to include in every reading sent to OSIsoft Cloud Services. For example, you can use this to specify the location of the devices being monitored by the Fledge server.

- **Authentication**

- **OCS Namespace:** Your namespace within OSIsoft Cloud Services.
- **OCS Tenant ID:** Your OSIsoft Cloud Services Tenant ID for your account.
- **OCS Client ID:** Your OSIsoft Cloud Services Client ID for your account.
- **OCS Client Secret:** Your OSIsoft Cloud Services Client Secret.

- **Connection management (These should only be changed with guidance from support)**

- **Sleep Time Retry:** Number of seconds to wait before retrying the HTTP connection (Fledge doubles this time after each failed attempt).
- **Maximum Retry:** Maximum number of times to retry connecting to the PI server.
- **HTTP Timeout:** Number of seconds to wait before Fledge will time out an HTTP connection attempt.

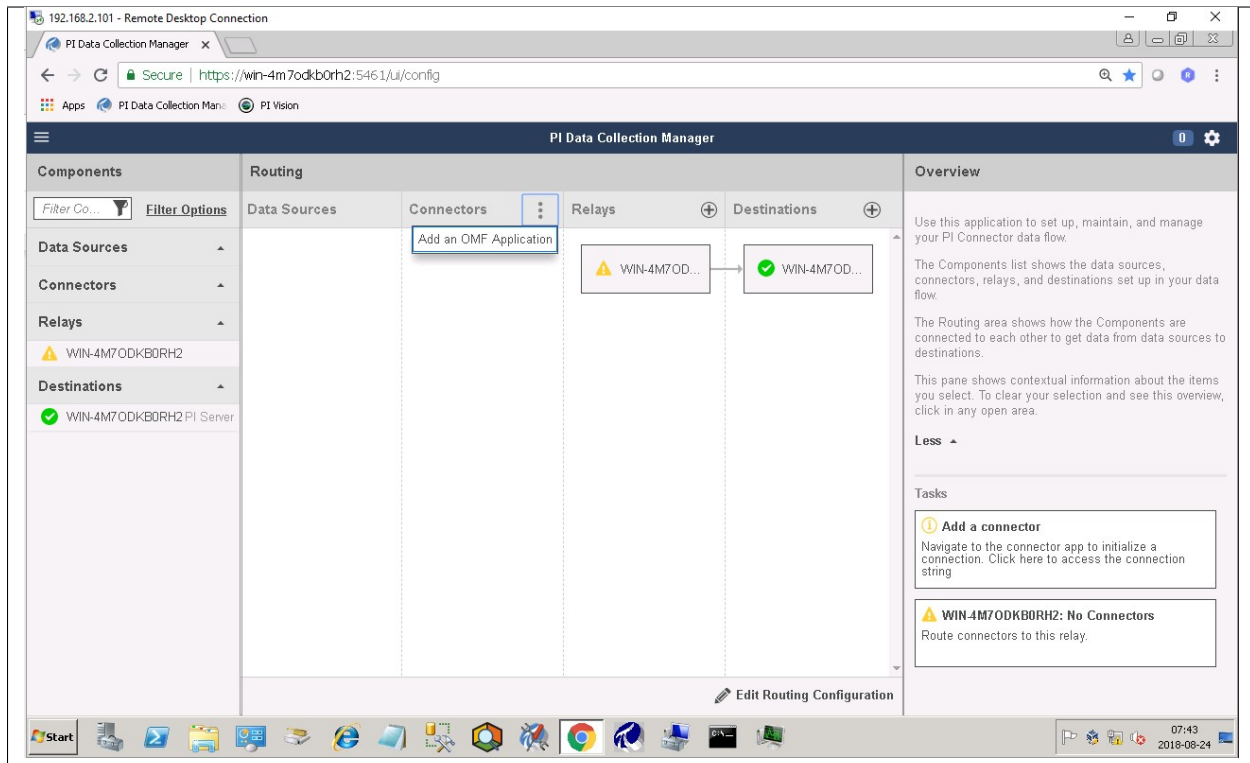
- **Other (Rarely changed)**

- **Integer Format:** Used to match Fledge data types to the data type configured in PI. This defaults to int64 but may be set to any OMF data type compatible with integer data, e.g. int32.
- **Number Format:** Used to match Fledge data types to the data type configured in PI. The default is float64 but may be set to any OMF datatype that supports floating point values.
- **Compression:** Compress the readings data before sending them to OSIsoft Cloud Services.

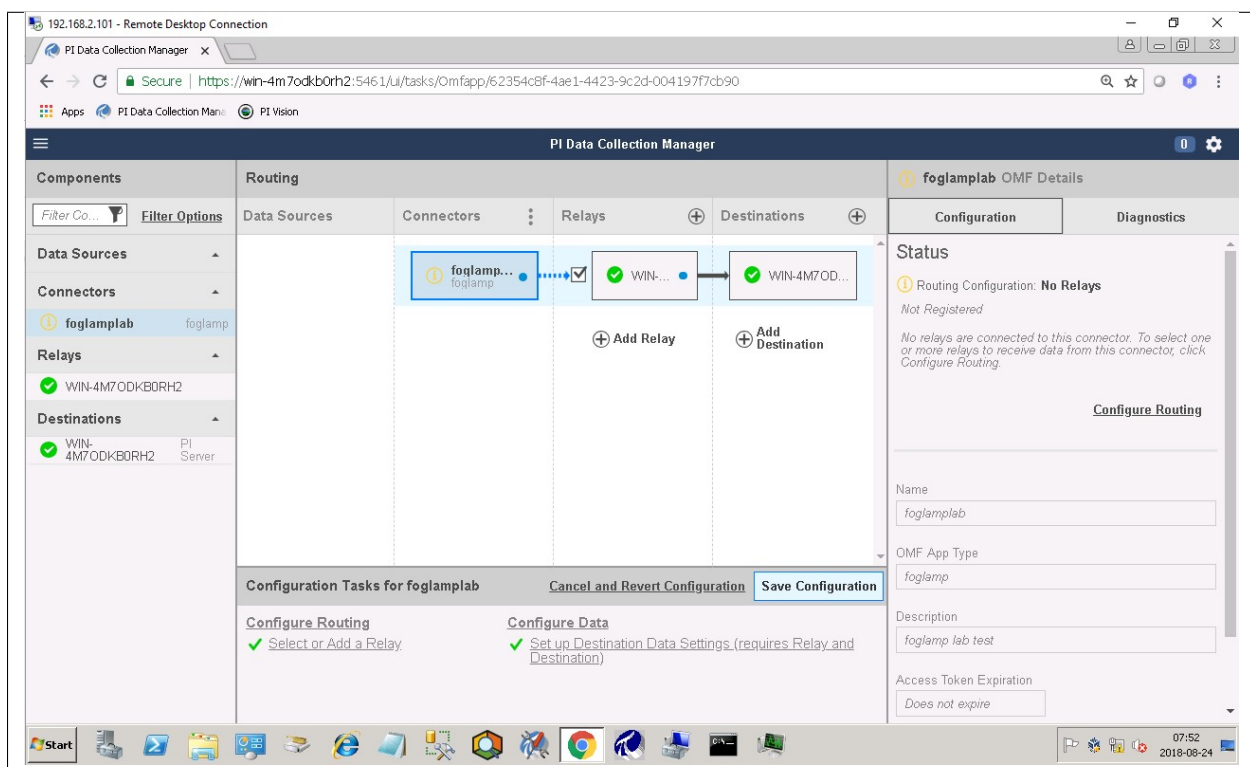
## 2.11 PI Connector Relay

**The PI Connector Relay has been discontinued by OSIsoft.** All new deployments should use the PI Web API endpoint. Existing installations will still be supported. The PI Connector Relay was the original mechanism by which OMF data could be ingesting into a PI Server. To use the PI Connector Relay, open and sign into the PI Relay Data Connection Manager.



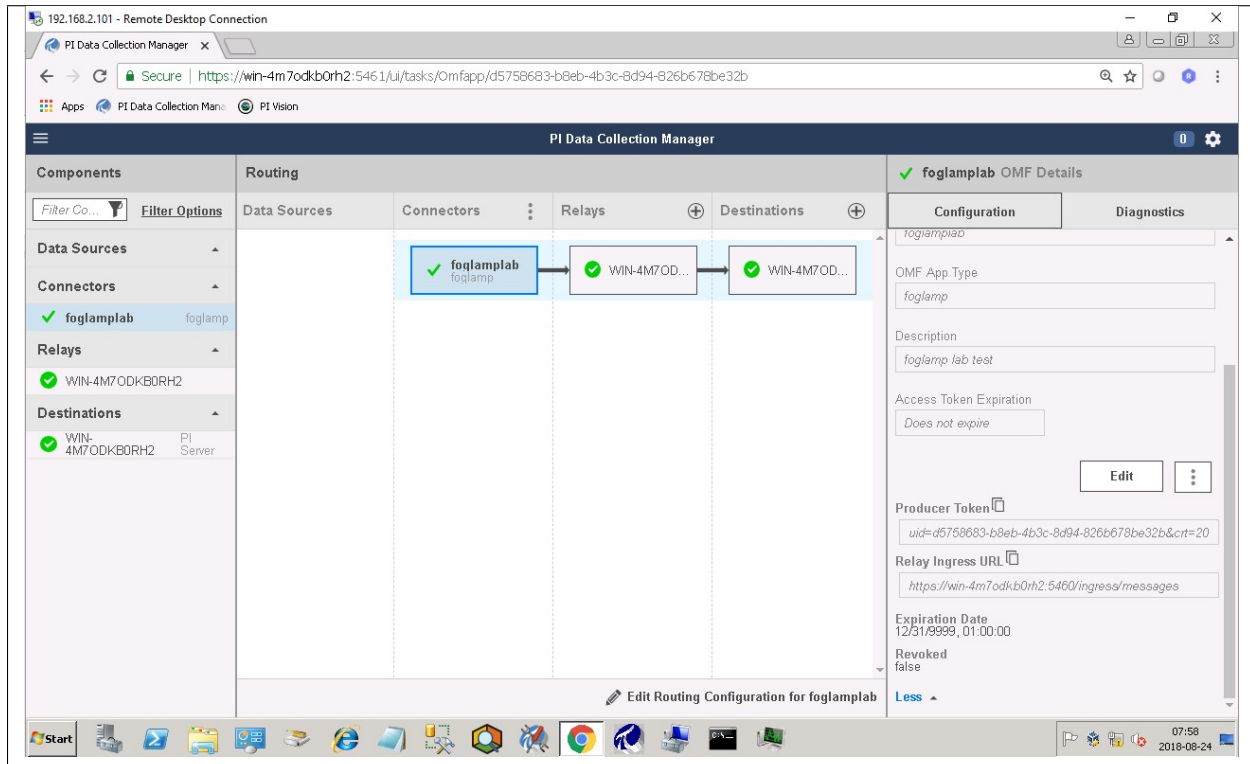


To add a new connector for the Fledge system, click on the drop down menu to the right of “Connectors” and select “Add an OMF application”. Add and save the requested configuration information.





Connect the new application to the PI Connector Relay by selecting the new Fledge application, clicking the check box for the PI Connector Relay and then clicking “Save Configuration”.



Finally, select the new Fledge application. Click “More” at the bottom of the Configuration panel. Make note of the Producer Token and Relay Ingress URL.

Now go to the Fledge user interface, create a new North instance and select the “OMF” plugin on the first screen. The second screen will request the following information:



?

Endpoint

Connector Relay

Send full structure

☒

Naming Scheme

Concise

Server hostname

localhost

Server port, 0=use the default

0

Producer Token

omf\_north\_0001

Data Source

readings

Static Data

Location: Palo Alto, Company: Dianomic

Sleep Time Retry

1

Maximum Retry

3

HTTP Timeout

10

Integer Format

int64

Number Format

float64

Compression

☒

Default Asset Framework Location

/fledge/data\_piwebapi/default

Asset Framework hierarchy rules

1

{}

PI Web API Authentication Method

anonymous

PI Web API User Id

user\_id

PI Web API Password

.....

PI Web API Kerberos keytab file

piwebapi\_kerberos\_https.keytab

OCS Namespace

name\_space

OCS Tenant ID

ocs\_tenant\_id

OCS Client ID

ocs\_client\_id

OCS Client Secret

.....



- **Basic Information**

- **Endpoint:** This is the type of OMF endpoint. In this case, choose Connector Relay.
- **Server hostname:** The hostname or address of the PI Connector Relay.
- **Server port:** The port the PI Connector Relay is listening on. Leave as 0 if you are using the default port.
- **Producer Token:** The Producer Token provided by the PI Relay Data Connection Manager.
- **Data Source:** Defines which data is sent to the PI Connector Relay. Choices are: readings or statistics (that is, Fledge's internal statistics).
- **Static Data:** Data to include in every reading sent to PI. For example, you can use this to specify the location of the devices being monitored by the Fledge server.

- **Connection management (These should only be changed with guidance from support)**

- **Sleep Time Retry:** Number of seconds to wait before retrying the HTTP connection (Fledge doubles this time after each failed attempt).
- **Maximum Retry:** Maximum number of times to retry connecting to the PI server.
- **HTTP Timeout:** Number of seconds to wait before Fledge will time out an HTTP connection attempt.

- **Other (Rarely changed)**

- **Integer Format:** Used to match Fledge data types to the data type configured in PI. This defaults to int64 but may be set to any OMF data type compatible with integer data, e.g. int32.
- **Number Format:** Used to match Fledge data types to the data type configured in PI. The default is float64 but may be set to any OMF datatype that supports floating point values.
- **Compression:** Compress the readings data before sending it to the PI System.

### 2.11.1 Naming Scheme

The naming of objects in the Asset Framework and of the attributes of those objects has a number of constraints that need to be understood when storing data into a PI Server using OMF. An important factor in this is the stability of your data structures. If you have objects in your environment that are likely to change, you may wish to take a different naming approach. Examples of changes are a difference in the number of attributes between readings, and a change in the data types of attributes.

This occurs because of a limitation of the OMF interface to the PI Server. Data is sent to OMF in a number of stages. One of these is the definition of the Types used to create AF Element Templates. OMF uses a Type to define an AF Element Template but once defined it cannot be changed. If an updated Type definition is sent to OMF, it will be used to create a new AF Element Template rather than changing the existing one. This means a new AF Element Template is created each time a Type changes.

The OMF plugin names objects in the Asset Framework based upon the asset name in the reading within Fledge. Asset names are typically added to the readings in the south plugins, however they may be altered by filters between the south ingest and the north egress points in the data pipeline. Asset names can be overridden using the *OMF Hints* mechanism described below.

The attribute names used within the objects in the PI System are based on the names of the datapoints within each Reading within Fledge. Again *OMF Hints* can be used to override this mechanism.

The naming used within the objects in the Asset Framework is controlled by the *Naming Scheme* option:

**Concise** No suffix or prefix is added to the asset name and property name when creating objects in the Asset Framework and PI Points in the PI Data Archive. However, if the structure of an asset changes a new AF Element Template will be created which will have the suffix `-type*x*` appended to it.



**Use Type Suffix** The AF Element names will be created from the asset names by appending the suffix -type\*x\* to the asset name. If the structure of an asset changes a new AF Element name will be created with an updated suffix.

**Use Attribute Hash** AF Attribute names will be created using a numerical hash as a prefix.

**Backward Compatibility** The naming reverts to the rules that were used by version 1.9.1 and earlier of Fledge: both type suffixes and attribute hashes will be applied to the name.

## 2.11.2 Asset Framework Hierarchy Rules

The Asset Framework rules allow the location of specific assets within the Asset Framework to be controlled. There are two basic types of hint:

- Asset name placement: the name of the asset determines where in the Asset Framework the asset is placed,
- Meta data placement: metadata within the reading determines where the asset is placed in the Asset Framework.

The rules are encoded within a JSON document. This document contains two properties in the root of the document: one for name-based rules and the other for metadata based rules.

```
{
  "names" :
  {
    "asset1" : "/Building1/EastWing/GroundFloor/Room4",
    "asset2" : "Room14"
  },
  "metadata" :
  {
    "exist" :
    {
      "temperature" : "temperatures",
      "power"       : "/Electrical/Power"
    },
    "nonexist" :
    {
      "unit" : "Uncalibrated"
    },
    "equal" :
    {
      "room" :
      {
        "4" : "ElecticalLab",
        "6" : "FluidLab"
      }
    },
    "notequal" :
    {
      "building" :
      {
        "plant" : "/Office/Environment"
      }
    }
  }
}
```

The name type rules are simply a set of asset name and Asset Framework location pairs. The asset names must be complete names; there is no pattern matching within the names.



The metadata rules are more complex. Four different tests can be applied:

- **exists**: This test looks for the existence of the named datapoint within the asset.
- **nonexist**: This test looks for the lack of a named datapoint within the asset.
- **equal**: This test looks for a named datapoint having a given value.
- **notequal**: This test looks for a name datapoint having a value different from that specified.

The *exist* and *nonexist* tests take a set of name/value pairs that are tested. The name is the datapoint name to examine and the value is the Asset Framework location to use. For example

```
"exist" :
{
  "temperature" : "temperatures",
  "power"       : "/Electrical/Power"
}
```

If an asset has a datapoint called *temperature* it will be stored in the AF hierarchy *temperatures*, if the asset had a datapoint called *power* the asset will be placed in the AF hierarchy */Electrical/Power*.

The *equal* and *notequal* tests take an object as a child, the name of the object is datapoint to examine, the child nodes are sets of values and locations. For example

```
"equal" :
{
  "room" :
  {
    "4" : "ElectricalLab",
    "6" : "FluidLab"
  }
}
```

In this case if the asset has a datapoint called *room* with a value of *4* then the asset will be placed in the AF location *ElectricalLab*, if it has a value of *6* then it is placed in the AF location *FluidLab*.

If an asset matches multiple rules in the ruleset it will appear in multiple locations in the hierarchy, the data is shared between each of the locations.

If an OMF Hint exists within a particular reading this will take precedence over generic rules.

The AF location may be a simple string or it may also include substitutions from other datapoints within the reading. For example if the reading has a datapoint called *room* that contains the room in which the readings were taken, an AF location of */BuildingA/\${room}* would put the reading in the Asset Framework using the value of the room datapoint. The reading

```
"reading" : {
  "temperature" : 23.4,
  "room"        : "B114"
}
```

would be put in the AF at */BuildingA/B114* whereas a reading of the form

```
"reading" : {
  "temperature" : 24.6,
  "room"        : "2016"
}
```

would be put at the location */BuildingA/2016*.



It is also possible to define defaults if the referenced datapoint is missing. In our example above if we used the location `/BuildingA/${room:unknown}` a reading without a `room` datapoint would be placed in `/BuildingA/unknown`. If no default is given and the data point is missing then the level in the hierarchy is ignored. E.g. if we use our original location `/BuildingA/${room}` and we have the reading

```
"reading" : {
  "temperature" : 22.8,
}
```

this reading would be stored in `/BuildingA`.

### 2.11.3 OMF Hints

The OMF plugin also supports the concept of hints in the actual data that determine how the data should be treated by the plugin. Hints are encoded in a specially named datapoint within the asset, *OMFHint*. The hints themselves are encoded as JSON within a string.

## 2.12 Number Format Hints

A number format hint tells the plugin what number format to use when inserting data into the PI Server. The following will cause all numeric data within the asset to be written using the format *float32*.

```
"OMFHint" : { "number" : "float32" }
```

The value of the *number* hint may be any numeric format that is supported by the PI Server.

### 2.13 Integer Format Hints

An integer format hint tells the plugin what integer format to use when inserting data into the PI Server. The following will cause all integer data within the asset to be written using the format *integer32*.

```
"OMFHint" : { "number" : "integer32" }
```

The value of the *number* hint may be any numeric format that is supported by the PI Server.

## 2.14 Type Name Hints

A type name hint specifies that a particular name should be used when defining the name of the type that will be created to store the object in the Asset Framework. This will override the *Naming Scheme* currently configured.

```
"OMFHint" : { "typeName" : "substation" }
```



## 2.15 Type Hint

A type hint is similar to a type name hint, but instead of defining the name of a type to create it defines the name of an existing type to use. The structure of the asset *must* match the structure of the existing type with the PI Server, it is the responsibility of the person that adds this hint to ensure this is the case.

```
"OMFHint" : { "type" : "pump" }
```

## 2.16 Tag Name Hint

Specifies that a specific tag name should be used when storing data in the PI Server.

```
"OMFHint" : { "tagName" : "AC1246" }
```

## 2.17 Datapoint Specific Hint

Hints may also be targeted to specific data points within an asset by using the datapoint hint. A *datapoint* hint takes a JSON object as its value; the object defines the name of the datapoint and the hint to apply.

```
"OMFHint" : { "datapoint" : { "name" : "voltage:", "number" : "float32" } }
```

The above hint applies to the datapoint *voltage* in the asset and applies a *number format* hint to that datapoint.

## 2.18 Asset Framework Location Hint

An Asset Framework location hint can be added to a reading to control the placement of the asset within the Asset Framework. An Asset Framework hint would be as follows:

```
"OMFHint" : { "AFLocation" : "/UK/London/TowerHill/Floor4" }
```

Note the following when defining an *AFLocation* hint:

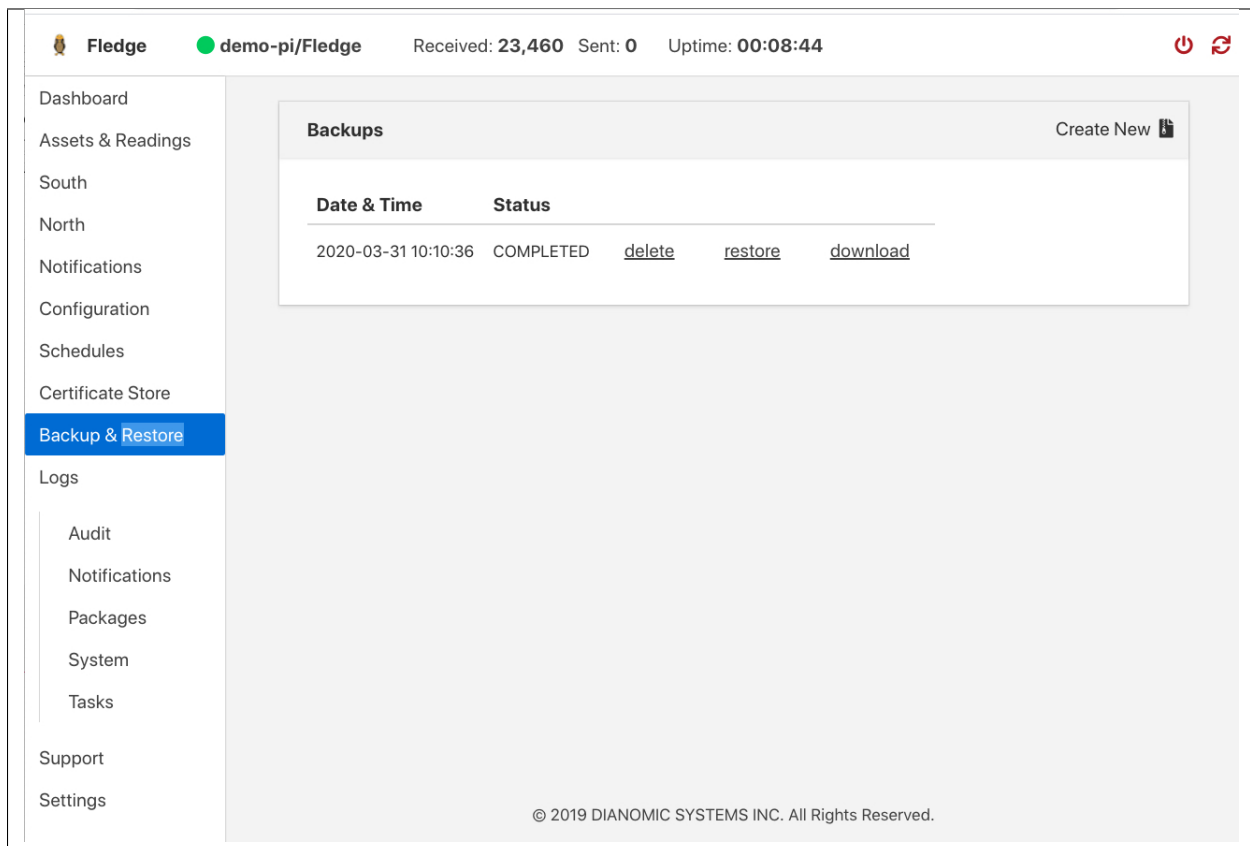
- An asset in a Fledge Reading is used to create a [Container in the OSIsoft Asset Framework](#). A *Container* is an AF Element with one or more AF Attributes that are mapped to PI Points using the OSIsoft PI Point Data Reference. The name of the AF Element comes from the Fledge Reading asset name. The names of the AF Attributes come from the Fledge Reading datapoint names.
- If you edit the AF Location hint, the Container will be moved to the new location in the AF hierarchy.
- If you disable the OMF Hint filter, the Container will not move.
- If you wish to move a Container, you can do this with the PI System Explorer. Right-click on the AF Element that represents the Container. Choose Copy. Select the AF Element that will serve as the new parent of the Container. Right-click and choose Paste. You can then return to the original Container and delete it. *Note that PI System Explorer does not have the traditional Cut function for AF Elements.*
- If you move a Container, OMF North will not recreate it. If you then edit the AF Location hint, the Container will appear in the new location.



## 2.19 Adding OMF Hints

An OMF Hint is implemented as a string data point on a reading with the data point name of *OMFHint*. It can be added at any point in the processing of the data, however a specific plugin is available for adding the hints, the .

## 2.20 Backing up and Restoring Fledge

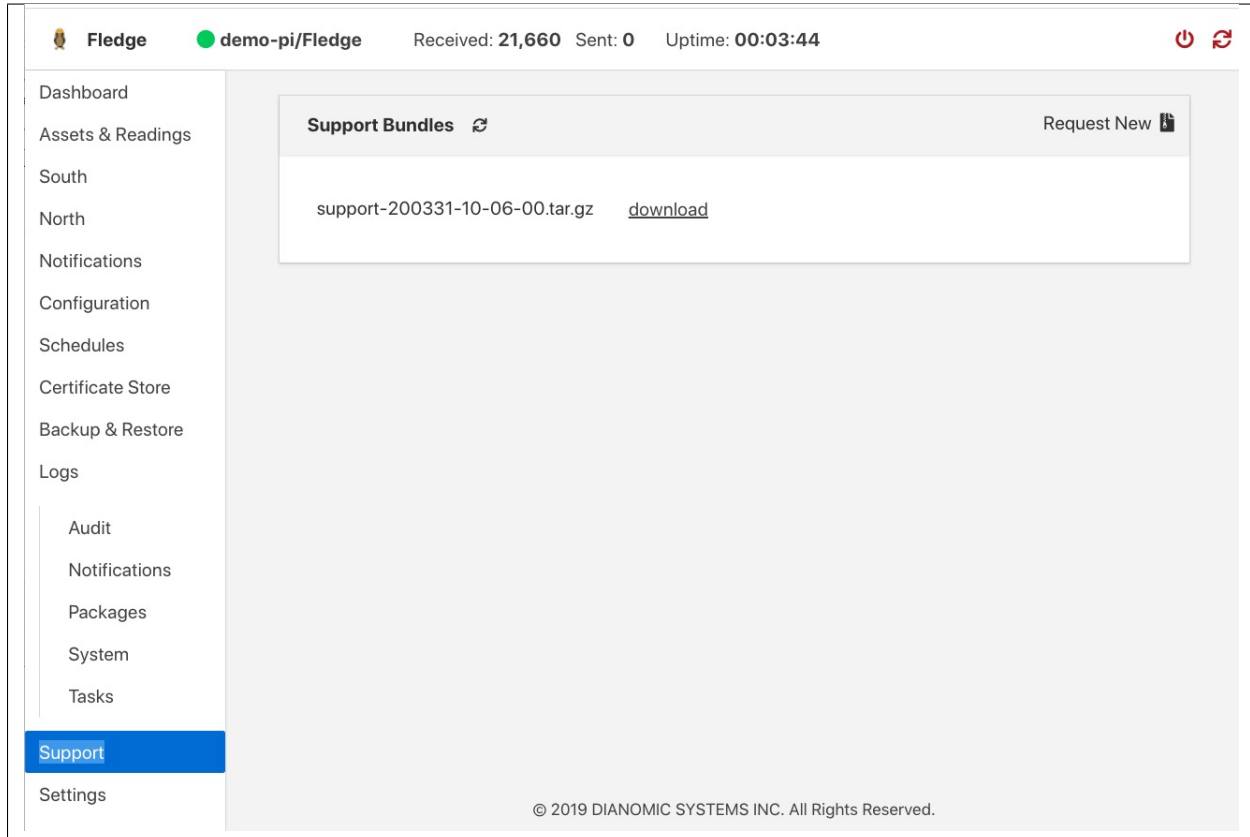


The screenshot displays the Fledge web interface. At the top, the status bar shows 'demo-pi/Fledge', 'Received: 23,460', 'Sent: 0', and 'Uptime: 00:08:44'. The left sidebar lists various system components, with 'Backup & Restore' highlighted. The main panel, titled 'Backups', contains a 'Create New' button and a table of backup records. The table has two columns: 'Date & Time' and 'Status'. A single record is listed with the date '2020-03-31 10:10:36' and status 'COMPLETED'. To the right of the status are three links: 'delete', 'restore', and 'download'. The footer of the interface indicates '© 2019 DIANOMIC SYSTEMS INC. All Rights Reserved.'

You can make a complete backup of all Fledge data and configuration. To do this, click on “Backup & Restore” in the left menu bar. This screen will show a list of all backups on the system and the time they were created. To make a new backup, click the “Backup” button in the upper right corner of the screen. You will briefly see a “Running” indicator in the lower left of the screen. After a period of time, the new backup will appear in the list. You may need to click the refresh button in the upper left of the screen to refresh the list. You can restore, delete or download any backup simply by clicking the appropriate button next to the backup in the list.



## 2.21 Troubleshooting and Support Information



Fledge keep detailed logs of system events for both auditing and troubleshooting use. To access them, click “Logs” in the left menu bar. There are five logs in the system:

- **Audit:** Tracks all configuration changes and data uploads performed on the Fledge system.
- **Notifications:** If you are using the Fledge notification service this log will give details of notifications that have been triggered
- **Packages:** This log will give you information about the installation and upgrade of Fledge packages for services and plugins.
- **System:** All events and scheduled tasks and their status.
- **Tasks:** The most recent scheduled tasks that have run and their status

If you have a service contract for your Fledge system, your support technician may ask you to send system data to facilitate troubleshooting an issue. To do this, click on “Support” in the left menu and then “Request New” in the upper right of the screen. This will create an archive of information. Click download to retrieve this archive to your system so you can email it to the technician.



## 2.22 Package Uninstallation

### 2.22.1 Debian Platform

Use the `apt` or the `apt-get` command to uninstall Fledge:

```
sudo apt -y purge fledge
```

---

**Note:** You may notice the warning in the last row of the package removal output:

`dpkg: warning: while removing fledge, directory '/usr/local/fledge' not empty so not removed`

---

This is due to the fact that the data directory (`/usr/local/fledge/data` by default) has not been removed, in case we might want to analyze or reuse the data further. So, if you want to remove fledge completely from your system, then do `rm -rf /usr/local/fledge` directory.







## PROCESSING DATA

We have already seen that Fledge can collect data from a variety of sources, buffer it locally and send it on to one or more destination systems. It is also possible to process the data within Fledge to edit, augment or remove data as it traverses the Fledge system. In the same way Fledge makes extensive use of plugin components to add new sources of data and new destinations for that data, Fledge also uses plugins to add processing filters to the Fledge system.

### 3.1 Why Use Filters?

The concept behind filters is to create a set of small, useful pieces of functionality that can be inserted into the data flow from the south data ingress side to the north data egress side. By making these elements small and dedicated to a single task it increases the re-usability of the filters and greatly improves the chances when a new requirement is encountered that it can be satisfied by creating a filter pipeline from existing components or by augmenting existing components with the addition of any incremental processing required. The ultimate aim being to be able to create new applications within Fledge by merely configuring filters from the existing pool of available filters into a suitable pipeline without the need to write any new code.

### 3.2 What Can Be Done?

Data processing is done via plugins that are known as *filters* in Fledge, therefore it is not possible to give a definitive list of all the different processing that can occur, the design intent is that it is expandable by the user. The general types of things that can be done are;

- **Modify a value in a reading.** This could be as simple as applying a scale factor to convert from one measurement scale to another or more complex mathematical operation.
- **Modify asset or datapoint names.** Perform a simple textual substitution in order to change the name of an asset or a data point within that asset.
- **Add a new calculated value.** A new value can be calculated from a set of values, either based over a time period or based on a combination of different values, e.g. calculate power from voltage and current.
- **Add metadata to an asset.** This allows data such as units of measurement or information about the data source to be added to the data.
- **Compress data.** Only send data forward when the data itself shows significant change from previous values. This can be a useful technique to save bandwidth in low bandwidth or high cost network connections.
- **Conditionally forward data.** Only send data when a condition is satisfied or send low rate data unless some *interesting* condition is met.
- **Data conditioning.** Remove data from the data stream if the values are suspect or outside of reasonable conditions.



## 3.3 Where Can it Be Done?

Filters can be applied in two locations in the Fledge system;

- In the south service as data arrives in Fledge and before it is added to the storage subsystem for buffering.
- In the north tasks as the data is sent out to the upstream systems that receive data from the Fledge system.

More than one filter can be added to a single south or north within a Fledge instance. Filters are placed in an ordered pipeline of filters that are applied to the data in the order of the pipeline. The output of the first filter becomes the input to the second. Filters can thus be combined to perform complex sets of operations on a particular data stream into Fledge or out of Fledge.

The same filter plugin can appear in multiple places within a filter pipeline, a different instance is created for each and each one has its own configuration.

### 3.3.1 Adding a South Filter

In the following example we will add a filter to a south service. The filter we will use is the *expression* filter and we will convert the incoming value to a logarithmic scale. The south plugin used in this simple example is the *sinusoid* plugin that creates a simulated sine wave.

The process starts by selecting the *South* services in the Fledge GUI from the left-hand menu bar. Then click on the south service of interest. This will display a dialog that allows the south service to be edited.

The screenshot shows a configuration window titled "Sine South Service". Inside the window, there is a form with the following elements:

- Asset name:** A text input field containing the value "sinusoid".
- Enabled:** A checkbox that is currently checked.
- Show Advanced Config:** A link located to the right of the "Enabled" checkbox.
- Applications:** A section header with a plus icon (+) to its right, followed by a list area.
- Buttons:** "Cancel" and "Save" buttons are located at the bottom right of the main form area.
- Service Info:** A section header at the bottom of the window, with "Export Readings" and "Delete Service" buttons located below it.

Towards the bottom of this dialog is a section labeled *Applications* with a + icon to the right, select the + icon to add a filter to the south service. A filter wizard is now shown that allows you to select the filter you wish to add and give that filter a name.



The screenshot shows a configuration window titled "Sine South Service" with a close button in the top right corner. A progress bar at the top indicates two steps: "1" (Plugin Name) and "2" (Review Configuration). The "Plugin Name" step is active. Below the progress bar, there is a section for selecting a plugin. On the left, under the label "Plugin", there is a dropdown menu with four options: "asset", "change", "delta", and "expression". The "expression" option is highlighted with a blue background. To the right of the dropdown, the text "Apply an expression to the data stream" is displayed. Below the dropdown, there is a link that says "Install from available plugins". Under the "Name" label, there is a text input field containing the placeholder text "name". At the bottom of the window, there are two buttons: "Back" and "Next".

Select the *expression* filter and enter a name in the dialog. Now click on the *Next* button. A new page in the wizard appears that allows the configuration of the filter.



Sine South Service

1 Plugin Name 2 Review Configuration

Datapoint Name LogSine

Expression to apply log(sinusoid)

Enabled ☒

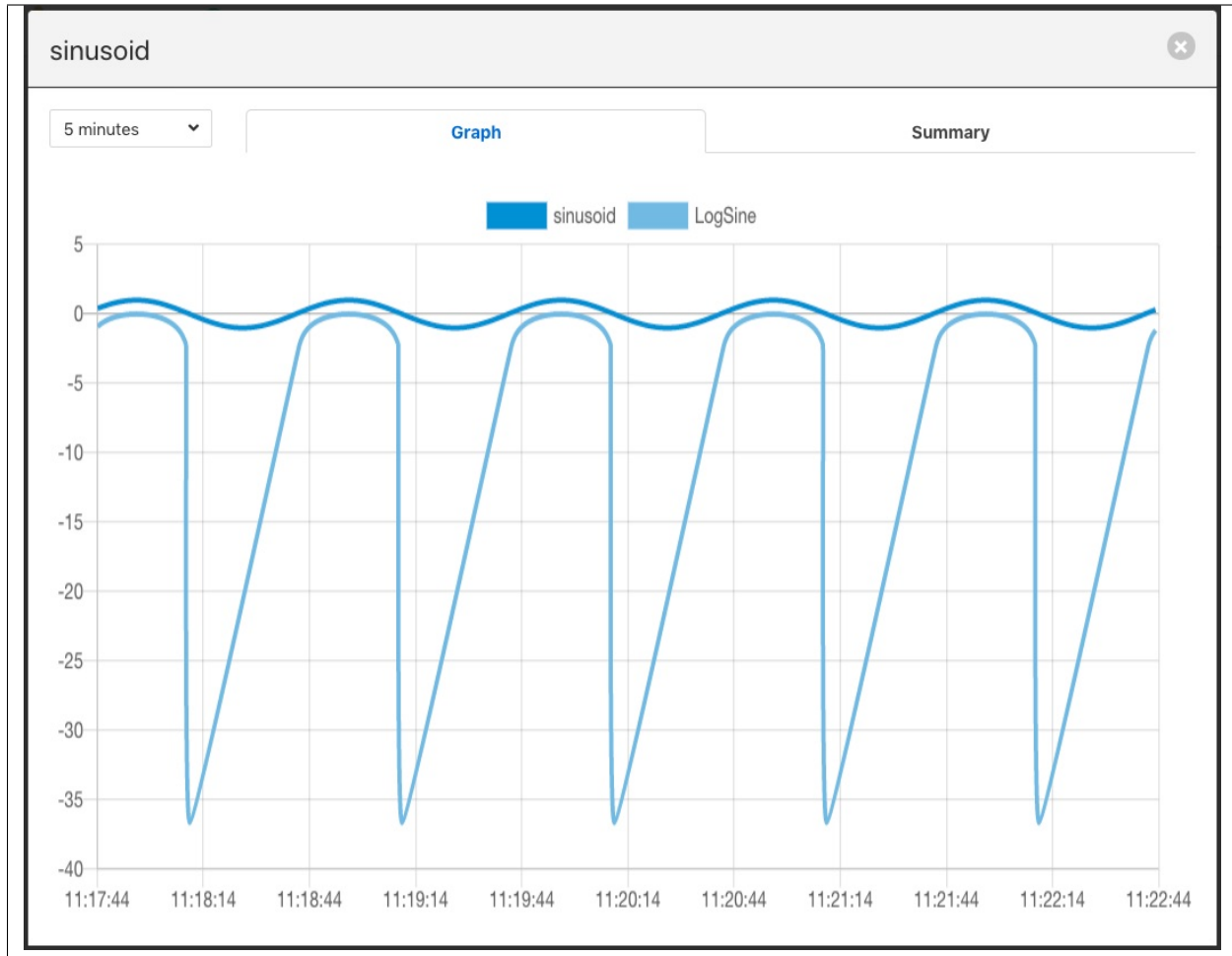
Previous Done

In the case of our expression filter we should add the expression we wish to execute *log(sinusoid)* and the name of the datapoint we wish to put the result in, *LogSine*. We can also choose to enable or disable the execution of this filter. We will enable it and click on *Done* to complete adding the filter.

Click on *Save* in the south edit dialog and our filter is now installed and running.

If we select the *Assets & Readings* option from the menu bar we can examine the sinusoid asset and view a graph of that asset. We will now see a second datapoint has been added, *LogSine* which is the result of executing our expression in the filter.





A second filter can be added in the same way, for example a *metadata* filter to create a pipeline. Now when we go back and view the south service we see two applications in the dialog.



Sine South Service

Asset name

sinusoid

Enabled

☒

[Show Advanced Config](#)

Applications +

≡ MyExpression

▼

≡ Location

▼

Cancel

Save

Service Info

http://localhost:37799

Export Readings

Delete Service

## Reordering Filters

The order in which the filters are applied can be changed in the south service dialog by clicking and dragging one filter above another in the *Applications* section of dialog.



Sine South Service

Asset name

sinusoid

Enabled

☒

[Show Advanced Config](#)

Applications +

≡ Location

▼

≡ MyExpression

▼

Cancel

Save

Service Info

http://localhost:37799

Export Readings

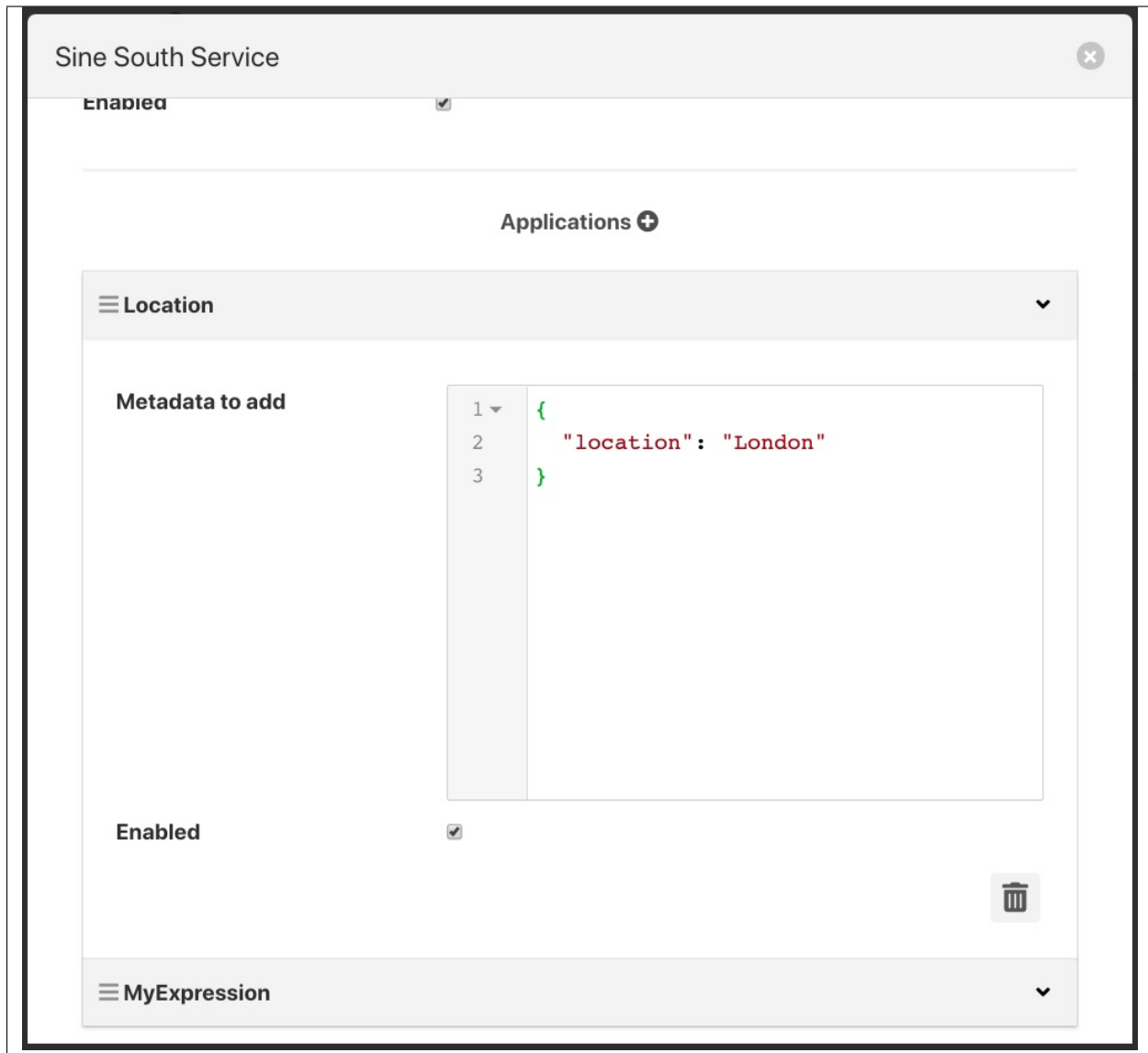
Delete Service

Filters are executed in a top to bottom order always. It may not matter in some cases what order a filter is executed in, in others it can have significant effect on the result.

### Editing Filter Configuration

A filters configuration can be altered from the south service dialog by selecting the down arrow to the right of the filter name. This will open the edit area for that filter and show the configuration that can be altered.





You can also remove a filter from the pipeline of filters by select the trash can icon at the bottom right of the edit area for the filter.

### 3.3.2 Adding Filters To The North

Filters can also be added to the north in the same way as the south. The same set of filters can be applied, however some may be less useful in the north than in the south as they apply to all assets that are sent north.

In this example we will use the metadata filter to label all the data that goes north as coming via a particular Fledge instance. As with the *South* service we start by selecting our north task from the *North* menu item in the left-hand menu bar.



PI Server

PI Web API Password

....

PI Web API Kerberos keytab file

piwebapi\_kerberos\_https.keytab

OCS Namespace

name\_space

OCS Tenant ID

ocs\_tenant\_id

OCS Client ID

ocs\_client\_id

OCS Client Secret

....

Enabled

☒

[Show Advanced Config](#)

Exclusive

☒

Interval

0

00:00:30

Applications +

Cancel

Save

Delete Instance

At the bottom of the dialog there is a *Applications* area, you may have to scroll the dialog to find it, click on the + icon. A selection dialog appears that allows you to select the filter to use. Select the *metadata* filter.



The screenshot shows a web interface titled "PI Server" with a close button in the top right corner. A progress bar at the top indicates two steps: "1 Plugin Name" (highlighted with a green circle) and "2 Review Configuration". The main content area contains a "Plugin" dropdown menu with options: "fft", "FlirValidity", "metadata" (selected and highlighted), and "python27". To the right of the dropdown, the text "Metadata filter plugin" is displayed. Below the dropdown is a blue link that says "Install from available plugins". Underneath this is a "Name" label followed by a text input field containing the word "Floor". At the bottom of the form are two buttons: "Back" and "Next".

After clicking *Next* you will be shown the configuration page for the particular filter you have chosen. We will edit the JSON that defines the metadata tags to add and set a name of *floor* and a value of *1*.



PI Server

1 Plugin Name 2 Review Configuration

Metadata to add

```
1 {  
2   "floor": "1"  
3 }
```

Enabled ☒

Previous Done

After enabling and clicking on *Done* we save the north changes. All assets sent to this PI Server connection will now be tagged with the tag “floor” and value “1”.

Although this is a simple example of labeling data other things can be done here, such as limiting the rate we send data to the PI Server until an *interesting* condition becomes true, perhaps to save costs on an expensive link or prevent a network becoming loaded until normal operating conditions. Another option might be to block particular assets from being sent on this link, this could be useful if you have two destinations and you wish to send a subset of assets to each.

This example used a PI Server as the destination, however the same mechanism and filters may be used for any north destination.



### 3.4 Some Useful Filters

A number of simple filters are worthy of mention here, a complete list of the currently available filters in Fledge can be found in the section .

#### 3.4.1 Scale

The filter *fledge-filter-scale* applies a scale factor and offset to the numeric values within an asset. This is useful for operations such as changing the unit of measurement of a value. An example might be to convert a temperature reading from Centigrade to Fahrenheit.

#### 3.4.2 Metadata

The filter *fledge-filter-metadata* will add metadata to an asset. This could be used to add information such as unit of measurement, machine data (make, model, serial no) or the location of the asset to the data.

#### 3.4.3 Delta

The filter *fledge-filter-delta* allows duplicate data to be removed, only forwarding data that changes by more than a configurable percentage. This can be useful if a value does not change often and there is a desire not to forward all the *similar* values in order to save network bandwidth or reduce storage requirements.

#### 3.4.4 Rate

The filter *fledge-filter-rate* is similar to the delta filter above, however it forwards data at a fixed rate that is lower the rate of the oncoming data but can send full rate data should an *interesting* condition be detected. The filter is configured with a rate to send data, the values sent at that rate are an average of the values seen since the last value was sent.

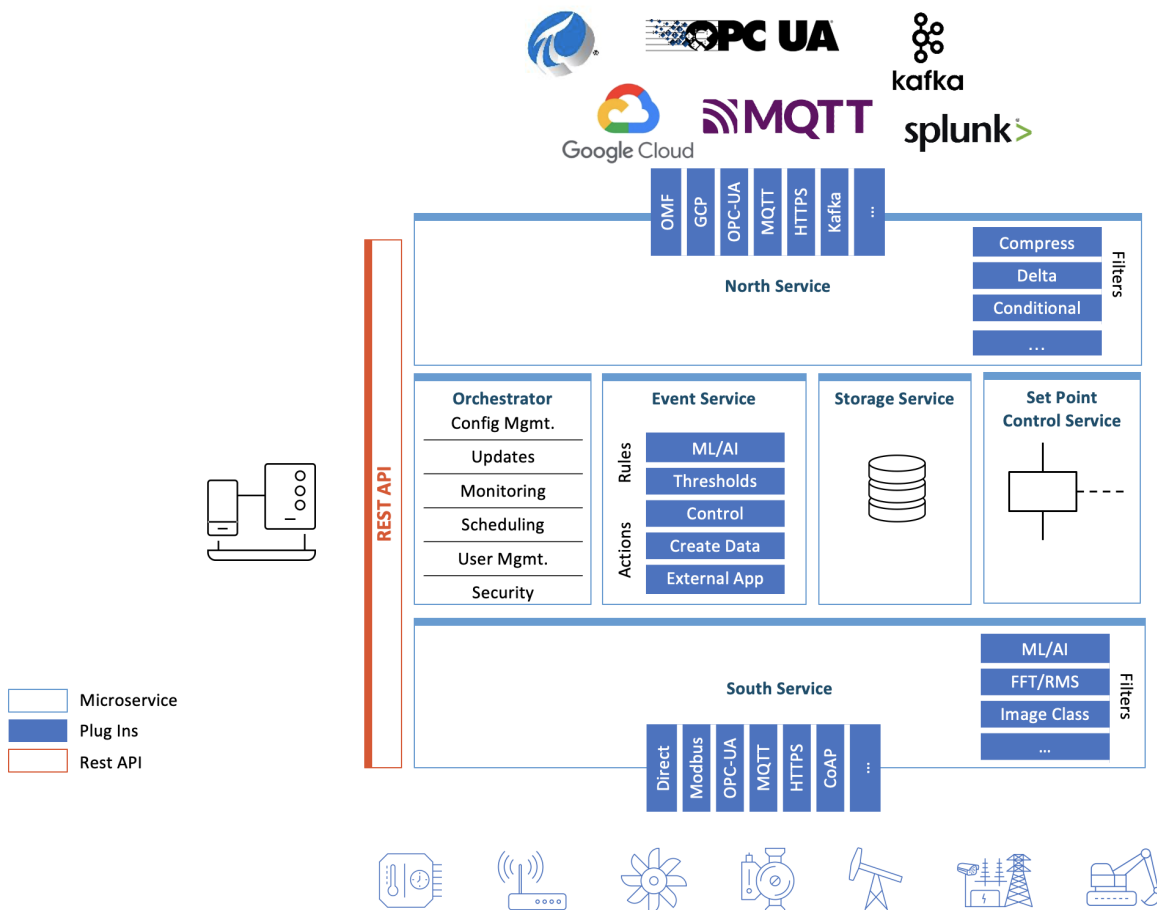
A rate of one reading per minute for example would average all the values for 1 minute and then send that average as the reading at the end of that minute. A condition can be added, when that condition is triggered all data is forwarded at full rate of the incoming data until a further condition is triggered that causes the reduced rate to be resumed.



## FLEDGE ARCHITECTURE

The following diagram shows the architecture of Fledge:

- Components in blue are **plugins**. Plugins are light-weight modules that enable Fledge to be extended. There are a variety of types of plugins: south-facing, north-facing, storage engine, filters, event rules and event delivery mechanisms. Plugins can be written in python (for fast development) or C++ (for high performance).
- Components with a blue line at the top of the box are **microservices**. They can co-exist in the same operating environment or they can be distributed across multiple environments.





### 4.1 Fledge Core

The Core microservice coordinates all of the Fledge operations. Only one Core service can be active at any time.

Core functionality includes:

**Scheduler:** Flexible scheduler to bring up processes.

**Configuration Management:** maintain configuration of all Fledge components. Enable software updates across all Fledge components.

**Monitoring:** monitor all Fledge components, and if a problem is discovered (such as an unresponsive microservice), attempt to self-heal.

**REST API:** expose external management and data APIs for functionality across all components.

**Backup:** Fledge system backup and restore functionality.

**Audit Logging:** maintain logs of system changes for auditing purposes.

**Certificate Storage:** maintain security certificates for different components, including south services, north services, and API security.

**User Management:** maintain authentication and permission info on Fledge administrators.

**Asset Browsing:** enable querying of stored asset data.

### 4.2 Storage Layer

The Storage microservice provides two principal functions: a) maintenance of Fledge configuration and run-time state, and b) storage/buffering of asset data. The type of storage engine is pluggable, so in installations with a small footprint, a plugin for SQLite may be chosen, or in installations with a high number of concurrent requests and larger footprint Postgresql may be suitable. In micro installations, for example on Edge devices, or when high bandwidth is required, an in-memory temporary storage may be the best option.

### 4.3 South Microservices

South microservices offer bi-directional communication of data and metadata between Edge devices, such as sensors, actuators or PLCs and Fledge. Smaller systems may have this service installed onboard Edge devices. South components are typically deployed as always-running services, which continuously wait for new data.

### 4.4 North Microservices

North microservices offer bi-directional communication of data and metadata between the Fledge platform and larger systems located locally or in the cloud. Larger systems may be private and public Cloud data services, proprietary solutions or Fledge instances with larger footprints. North components are typically deployed as one-shot tasks, which periodically spin up and send data which has been batched, then spin down. However, they can also be deployed as continually-running services.



## 4.5 Filters

Filters are plugins which modify streams of data that flow through Fledge. They can be deployed at ingress (in a South service), or at egress (in a North service). Typically, ingress filters are used to transform or enrich data, and egress filters are used to reduce flow to northbound pipes and infrastructure, i.e. by compressing or reducing data that flows out. Multiple filters can be applied in “pipelines”, and once configured, pipelines can be applied to multiple south or north services.

A sample of existing Filters:

**Expression:** apply an arbitrary mathematical equation across one or more assets.

**Python35:** run user-specified python code across one or more assets.

**Metadata:** apply tags to data, to note the device/location it came from, or to attribute data to a manufactured part.

**RMS/Peak:** summarize vibration data by generating a Root Mean Squared (RMS) across n samples.

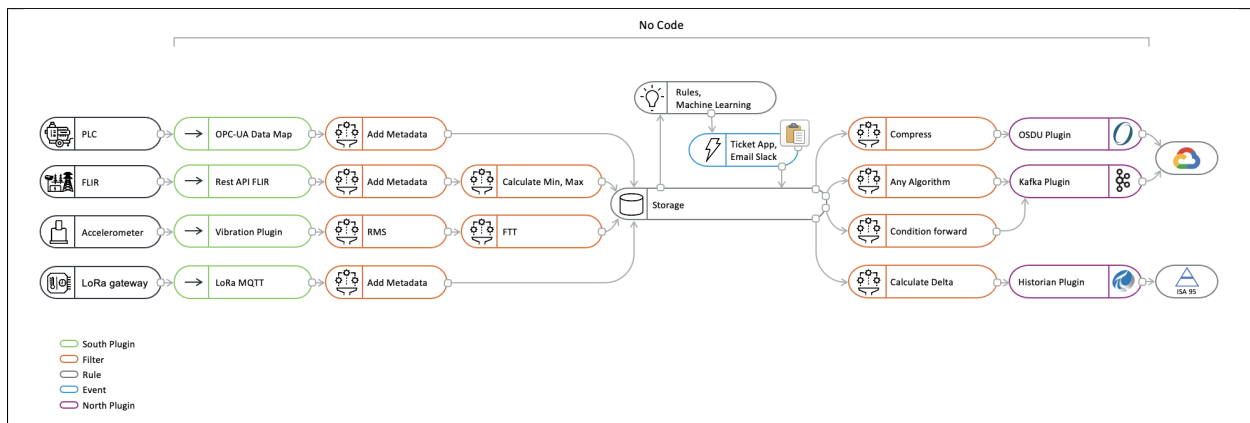
**FFT:** generate a Fast Fourier Transform (FFT) of vibration data to discover component waveforms.

**Delta:** Only send data that has changed by a specified amount.

**Rate:** buffer data but don't send it, then if an error condition occurs, send the previous data.

**Contrast:** Enhance the contrast of image type data

Filters may be concatenated together to form a data pipeline from the data source to the storage layer, in the south microservice. Or from the storage layer to the destination in the north.



This allows for data processing to be built up via the graphical interface of Fledge with little or no coding required. Filters that are applied in a south service will affect all out going streams whilst those applied in the north only affect the data that is sent on that particular connection to an external system.



### 4.6 Event Service

The event engine maintains zero or more rule/action pairs. Each rule subscribes to desired asset data, and evaluates it. If the rule triggers, its associated action is executed.

**Data Subscriptions:** Rules can evaluate every data point for a specified asset, or they can evaluate the minimum, maximum or average of a specified window of data points.

**Rules:** the most basic rule evaluates if values are over/under a specified threshold. The Expression plugin will evaluate an arbitrary math equation across one or more assets. The Python35 plugin will execute user-specified python code to one or more assets.

**Actions:** A variety of delivery mechanisms exist to execute a python application, create arbitrary data, alter the configuration of Fledge, send a control message, raise a ticket in a problem ticking system or email/slack/hangout/communicate a message.

### 4.7 Set Point Control Service

Fledge is not designed to replace real time control systems, it does however allow for non-time-critical control using the control microservice. Control messages may originate from a number of sources; the north microservice, the event service, the REST API or from scheduled events. It is the job of the control service to route these control messages to the correct destination. It also provides a simple form of scripting to allow control messages to generate chains of writes and operations on the south service and also modify the configuration of the Fledge itself.

### 4.8 REST API

The Fledge API provides methods to administer Fledge, and to interact with the data inside it.

### 4.9 Graphical User Interface

A GUI enables administration of Fledge. All GUI capability is through the REST API, so Fledge can also be administered through scripts or other management tools. The GUI contains pages to:

**Health:** See if services are responsive. See data that's flowed in and out of Fledge

**Assets & Readings:** analytics of data in Fledge

**South:** manage south services

**North:** manage north services

**Notifications:** manage event engine rules and delivery mechanisms

**Configuration Management:** manage configuration of all components

**Schedules:** flexible scheduler management for processes and tasks

**Certificate Store:** manage certificates

**Backup & Restore:** backup/restore Fledge

**Logs:** see system, notification, audit, packages and tasks logging information

**Support:** support bundle contents with system diagnostic reports

**Settings:** set/reset connection and GUI related settings



## BUFFERING & STORAGE

One of the micro-services that makes up the core of a Fledge implementation is the storage micro-service. This is responsible for

- storing the configuration of Fledge
- buffering the data read from the south
- maintaining the Fledge audit log
- persisting the state of the system

The storage service is configurable, like other services within Fledge and uses plugins to extend the functionality of the storage system. These storage plugins provide the underlying mechanism by which data is stored within Fledge. Fledge can make use of either one or two of these plugins at any one time. If a single plugin is used then this plugin provides the storage for all data. If two plugins are used, one will be for the buffering of readings and the other for the storage of the configuration.

As standard Fledge comes with 3 storage plugins

- **SQLite**: A plugin that can store both configuration data and the readings data using SQLite files as the backing store. The plugin uses multiple SQLite database to store different assets, allowing for high bandwidth data at the expense of limiting the number of assets that a single instance can ingest.,
- **SQLiteLB**: A plugin that can store both configuration data and the readings data using SQLite files as the backing store. This version of the SQLite plugin uses a single readings database and is better suited for environments that do not have very high bandwidth data. It does not limit the number of distinct assets that can be ingested.
- **PostgreSQL**: A plugin that can store both configuration and readings data which uses the PostgreSQL SQL server as a storage medium.
- **SQLiteMemory**: A plugin that can only be used to store reading data. It uses SQLite's in memory storage engine to store the reading data. This provides a high performance reading store however capacity is limited by available memory and if Fledge is stopped or there is a power failure the buffered data will be lost.

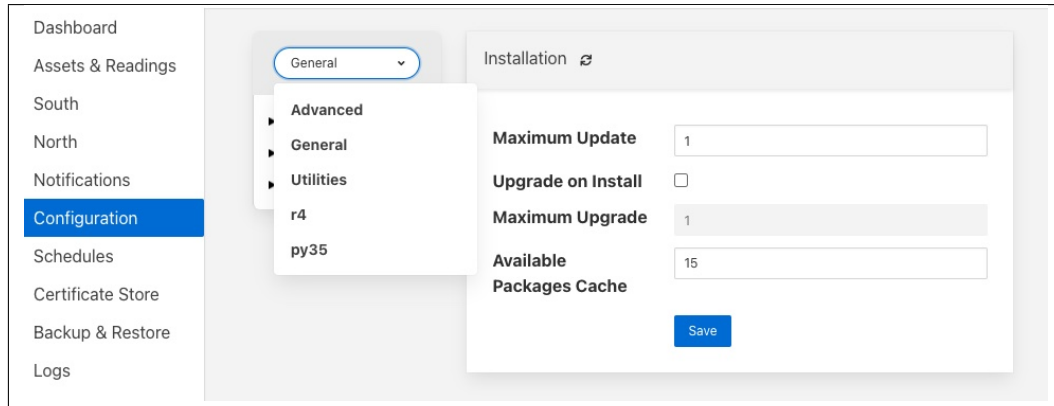
The default configuration uses the SQLite disk based storage engine for both configuration and reading data



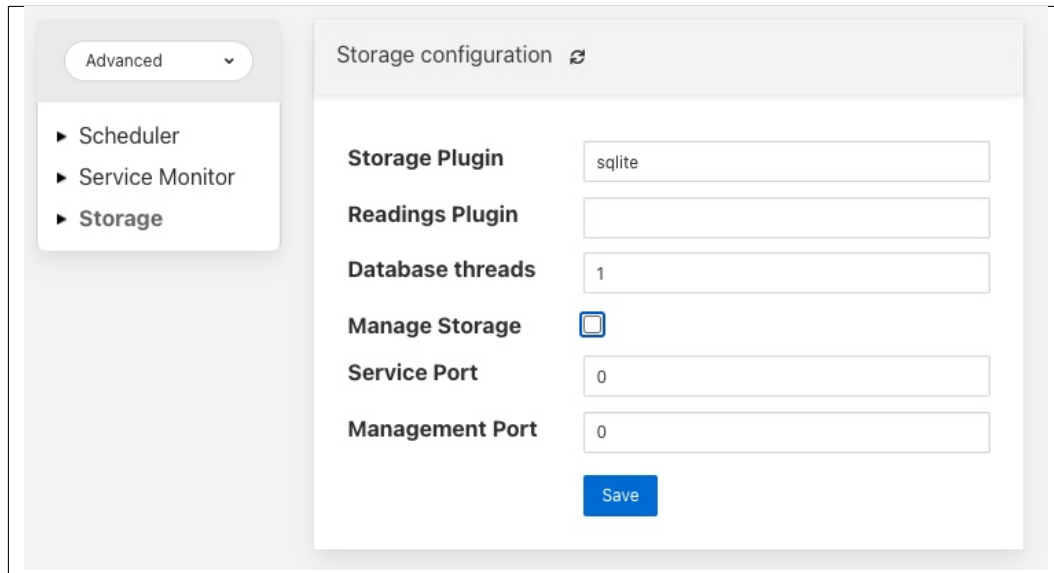
## 5.1 Configuring The Storage Plugin

Once installed the storage plugin can be reconfigured in much the same way as any Fledge configuration, either using the API or the graphical user interface to set the storage engine and its options.

- Using the user interface to configuration the storage, select the *Configuration* item in the left hand menu bar.



- In the category pull down menu select *Advanced*.



- To change the storage plugin to use for both configuration and readings enter the name of the new plugin in the *Storage Plugin* entry field. If *Readings Plugin* is left empty then the storage plugin will also be used to store reading data. The default set of plugins installed with Fledge that can be used as *Storage Plugin* values are:
  - *sqlite* - the SQLite file based storage engine.
  - *postgres* - the PostgreSQL server. Note the Postgres server is not installed by default when Fledge is installed and must be installed before it can be used.
- The *Readings Plugin* may be set to any of the above and may also be set to use the SQLite In Memory plugin by entering the value *sqlitememory* into the configuration field.



- The *Database threads* field allows for the number of threads used for database housekeeping to be controlled. In normal circumstances 1 is sufficient. If performance issues are seen this can be increased however it is rarely required to be greater than 1 and can have counter productive effects on heavily loaded systems.
- The *Manage Storage* option is only used when the database storage uses an external database server, such as PostgreSQL. Toggling this option on causes Fledge to start as stop the database server when Fledge is started and stopped. If it is left off then Fledge will assume the database server is running when it starts.
- The *Management Port* and *Service Port* options allow fixed ports to be assigned to the storage service. These settings are for debugging purposes only and the values should be set to 0 in normal operation.

Note: Additional storage engines may be installed to extend the set that is delivered with the standard Fledge installation. These will be documented in the packages that provide the storage plugin.

Storage plugin configurations are not dynamic and Fledge *must* be restarted after changing these values. Changing the plugin used to store readings will *not* cause the data in the previous storage system to be migrated to the new storage system and this data may be lost if it has not been sent onward from Fledge.

### 5.1.1 SQLite Plugin Configuration

The SQLite plugin has a more complex set of configuration options that can be used to configure how and when it creates more database to accommodate more distinct assets. This plugin is designed to allow greater ingest rates for readings by separating the readings for each asset into a database table for that asset. It does however result in limiting the number of distinct assets that can be handled due to the requirement to handle large number of database files.

- **Purge Exclusions:** This option allows the user to specify that the purge process should not be applied to particular assets. The user can give a comma separated list of asset names that should be excluded from the purge process. Note, it is recommended that this option is only used for extremely low bandwidth, lookup data that would otherwise be completely purged from the system when the purge process runs.
- **Pool Size:** The number of connections to create in the database connection pool.
- **No. Readings per database:** This option control how many assets can be stored in a single database. Each asset will be stored in a distinct table within the database. Once all tables within a database are allocated the plugin will use more databases to store further assets.
- **No. databases allocate in advance:** This option defines how many databases are create initially by the SQLite plugin.



- **Database allocation threshold:** The number of unused databases that must exist within the system. Once the number of available databases falls below this value the system will begin the process of creating extra databases.
- **Database allocation size:** The number of databases to create when the above threshold is crossed. Database creation is a slow process and hence the tuning of these parameters can impact performance when an instance receives a large number of new asset names for which it has previously not allocated readings tables.

## 5.2 Installing A PostgreSQL server

The precise commands needed to install a PostgreSQL server vary for system to system, in general a packaged version of PostgreSQL is best used and these are often available within the standard package repositories for your system.

### 5.2.1 Ubuntu Install

On Ubuntu or other apt based distributions the command to install postgres:

```
sudo apt install -y postgresql postgresql-client
```

Now, make sure that PostgreSQL is installed and running correctly:

```
sudo systemctl status postgresql
```

Before you proceed, you must create a PostgreSQL user that matches your Linux user. Supposing that user is `<fledge_user>`, type:

```
sudo -u postgres createuser -d <fledge_user>
```

The `-d` argument is important because the user will need to create the Fledge database.

A more generic command is:

```
sudo -u postgres createuser -d $(whoami)
```

## 5.3 SQLite Plugin Configuration

The SQLite storage engine has further options that may be used to configure its behavior. To access these configuration parameters click on the *sqlite* option under the *Storage* category in the configuration page.



The screenshot shows the 'Storage Plugin' configuration page in the Fledge web interface. On the left, a sidebar menu has 'Advanced' selected, with sub-items 'Scheduler', 'Service Monitor', 'Storage', and 'sqlite'. The main content area is titled 'Storage Plugin' and contains the following configuration options:

- Pool Size:** 5
- No. Readings per database:** 15
- No. databases to allocate in advance:** 3
- Database allocation threshold:** 1
- Database allocation size:** 2

A blue 'Save' button is located at the bottom right of the configuration panel.

Many of these configuration options control the performance of SQLite and it is important to have some background on how readings are stored within SQLite. The storage plugin stores readings for each distinct asset in a table for that asset. These tables are stored within a database. In order to improve concurrency multiple databases are used within the storage plugin. A set of parameters are used to define how these tables and databases are used.

- **Pool Size:** The number of connections to maintain to the database server.
- **No. Readings per database:** This controls the number of different assets that will be stored in each database file within SQLite.
- **No. databases to allocate in advance:** The number of SQLite databases that will be created at startup.
- **Database allocation threshold:** The point at which new databases are created. If the number of empty databases falls below this value then another set of databases will be created.
- **Database allocation size:** The number of database to allocate each time a new set of databases is required.

The setting of these parameters also imposes an upper limit on the number of assets that can be stored within a Fledge instance as SQLite has a maximum limit of 61 databases that can be in use at any time. Therefore the maximum number of readings is 60 times the number of readings per database. One database is reserved for the configuration data.



## 5.4 Storage Management

Fledge manages the amount of storage used by means of purge processes that run periodically to remove older data and thus limit the growth of storage use. The purging operations are implemented as Fledge tasks that can be scheduled to run periodically. There are two distinct tasks that are run

- **purge:** This task is responsible for limiting the readings that are maintained within the Fledge buffer.
- **system purge:** This task limit the amount of system data in the form of logs, audit trail and task history that is maintained.

### 5.4.1 Purge Task

The purge task is run via a scheduled called *purge*, the default for this schedule is to run the purge task every hour. This can be modified via the user interface in the *Schedules* menu entry or via the REST API by updating the schedule.

The purge task has two metrics it takes into consideration, the age of the readings within the system and the number of readings in the system. These can be configured to control how much data is retained within the system. Note however that this does not mean that there will never be data older than specified or more rows than specified as purge runs periodically and between executions of the purge task the readings buffered will continue to grow.

The configuration of the purge task can be found in the *Configuration* menu item under the *Utilities* section.

The screenshot shows the Fledge configuration interface for the 'Purge' task. On the left, a sidebar menu under 'Utilities' has 'Purge' selected. The main panel is titled 'Purge the readings, log, statistics history table' with a refresh icon. It contains five configuration fields: 'Age Of Data To Be Retained (In Hours)' with a value of 1, 'Max rows of data to retain' with a value of 1000000, 'Retain Unsent Data' with a dropdown menu set to 'purge unsent', 'Retain Stats History Data (In Days)' with a value of 30, and 'Retain Audit Trail Data (In Days)' with a value of 60. A blue 'Save' button is at the bottom right of the configuration area.

- **Age Of Data To Be Retained:** This configuration option sets the limit on how old data has to be before it is considered for purging from the system. It defines a value in hours, and only data older than this is considered for purging from the system.
- **Max rows of data to retain:** This defines how many readings should be retained in the buffer. This can override the age of data to retain and defines the maximum allowed number of readings that should be in the buffer after the purge process has completed.
- **Retain Unsent Data:** This defines how to treat data that has been read by Fledge but not yet sent onward to one or more of the north destinations for data. It supports a number of options



### Retain Unsent Data

### Retain Stats History Data (In Days)

✓ purge unsent

retain unsent to any destination

retain unsent to all destinations

- **purge unsent:** Data will be purged regardless if it has been sent onward from Fledge or not.
- **retain unsent to any destination:** Data will not be purged, i.e. it will be retained, if it has not been sent to any of the north destinations. If it has been sent to at least one of the north destinations then it will be purged.
- **retain unsent to all destinations:** Data will be retained until it has been sent to all north destinations that are enabled at the time the purge process runs. Disabled north destinations are not included in order to prevent them from stopping all data from being purged.

Note: This configuration category will not appear until after the purge process has run for the first time. By default this will be 1 hour after the Fledge instance is started for the first time.

## 5.4.2 System Purge Task

The system purge task is run via a scheduled called *system\_purge*, the default for this schedule is to run the system purge task every 23 hours and 50 minutes. This can be modified via the user interface in the *Schedules* menu entry or via the REST API by updating the schedule.

The configuration category for the system purge can be found in the *Configuration* menu item under the *Utilities* section.

Utilities ▼

► Purge

▼ Purge System

### Configuration of the Purge System ↗

**Statistics Retention**

7

**Audit Retention**

30

**Task Retention**

30

Save

- **Statistics Retention:** This defines the number of days for which full statistics are held within Fledge. Statistics older than this number of days are removed and only a summary of the statistics is held.
- **Audit Retention:** This defines the number of day for which the audit log entries will be retained. Once the entries reach this age they will be removed from the system.
- **Task Retention:** This defines the number of days for which history if task execution within Fledge is maintained.

Note: This configuration category will not appear until after the system purge process has run for the first time.

5.4. Storage Management

59







## ADDITIONAL SERVICES

The following additional services are currently available to extend the functionality of Fledge. These are optional services not installed as part of the base Fledge installation.

### 6.1 Notifications Service

Fledge supports an optional service, known as the notification service that adds an event engine to the Fledge installation. The notification services observed data as it flows into the Fledge storage service buffer and processes that data against a set of rules that are configurable by the user to determine if an event has occurred. Events may be either when a condition that was previously not met being is, or a condition that was previously met becoming no longer true. The notification service can then send a notification when an event occurs or, in the case of a condition that is met, it can send notifications as long as that condition is met.

The notification services operates on data that is in the storage layer, and is independent of the individual south services. This means that the notification rules can use data from several south services to evaluate if a condition has occurred. Also the data that is observed by the notification is after any filtering rules have been applied in the south services but before any filtering that occurs in the north tasks. The mechanism used to allow the notification service to observe data is that the notifications register with the storage service to be given the values for particular assets as they arrive at the storage service. A notification may register for several assets and is free to buffer that data internally within the notification service. This registration does not impact how the data that is requested is treated in the rest of the system; it will still for example follow the normal processing rules to be sent onward to the north systems.

#### 6.1.1 Notifications

The notification services manages *Notifications*, these are a set of parameters that it uses to determine if an event has occurred and a notification delivery should be made on the basis of that event.

A notification within the notification service consists of;

- A notification rule plugin that contains the logic to evaluate if a rule has been triggered, thus creating an event.
- A set of assets that are required to execute a notification rule.
- Information that defines how the data for each asset should be delivered to the notification rule.
- Configuration for the rule plugin that customizes that logic to this notification instance.
- A delivery plugin that provides the mechanism to delivery an event to destination for the notification.
- Configuration that may be required for the delivery plugin to operate.



### Notification Rules

Notification rules are the logic that is used by the notification to determine if an event has occurred or not. An event is basically based on the values of a number of attributes, either at a single point in time or over a period of time. The notification services is delivered with one built in rule, this is a very simple rule called the *threshold rule* it simply looks at a single asset to determine if the value of a datapoint within the asset goes above or below a set value.

A notification rule has associated with it a set of configuration options, these define how the plugin behaves but also what data the plugin requires to execute the evaluation logic within the plugin. These configuration parameters can be divided into two sets; those items that define the data the rule requires from the notification service itself and those that relate directly to the logic of the rule.

A rule may work across one or more assets, the assets it requires are configured in the rule configuration and passed the the notification service to enable the service to subscribe to those assets and be sent that data by the storage service. A rule plugin may ask for every value of the asset as it changes or it may ask for a window of data. A window is defined as the values of an asset within a given time frame. An example might be the last 10 minutes of values. In the case of the window the rule may be passed the average value, minimum, maximum or all values in that window. The requirements about how data is delivered to a rule may be hard coded within the logic of a rule or may be part of the configuration a user of the rule should provide.

The second type of configuration parameter a rule might include are those that control the logic itself, in the example of the *threshold rule* this would be the threshold value itself and the control if the event is considered to have triggered if the value is above or below the threshold.

The section contains a full list of currently available rule plugins for Fledge. As with other plugin types they are designed to be easily written by end users and developers, a guide is available for anyone wishing to write a notification rule plugin of their own.

### Notification Types

Notifications can be delivered under a number of different conditions based on the state returned from a notification rule and how it related to the previous state returned by the notification rule, this is known as the notification type. A notification may be one of three types, these types are used to define when and how often notification are delivered.

#### One shot

A one shot notification is sent once when the notification triggers but will not be resent again if the notification triggers on successive evaluations. Once the evaluation does not trigger, the notification is cleared and will be sent again the next time the notification rule triggers.

One shot notifications may be further tailored with a maximum repeat frequency, e.g. no more than once in any 15 minute period.

#### Toggle

A toggle notification is sent when the notification rule triggers and will not be resent again until the rule fails to trigger, in exactly the same way as a one shot trigger. However in this case when the notification rule first stops triggering a cleared notification is sent.

Again this may be modified by the addition of a maximum repeat frequency.



## Retriggered

A retriggered notification will continue to be sent when a notification rule triggers. The rate at which the notification is sent can be controlled by a maximum repeat frequency, e.g. send a notification every 5 minutes until the condition fails to trigger.

## Notification Delivery

The notification service does not natively support any form of notification delivery, it relies upon a notification delivery plugin in order to delivery a notification of an event to a user or external system that should be alerted to the event that has occurred. Typical notification deliveries might be to alert a user via some form of paging or messaging system, push an event to an external application by sending some machine level message, execute an external program or code segment to make an action occur, switching on an indication light or in extreme cases maybe shutting down a machine for which a critical fault has been detected. The section contains a full list of currently available notification delivery plugins, however like other plugins these are easily extended and a guide is available for writing notification plugins to extend the available set of plugins.

### 6.1.2 Installing the Notification Service

The notification service is not part of the base Fledge installation and is not a plugin, it is a separate microservice dedicated to the detection of events and the sending of notifications.

#### Building Notification Service

As with *Fledge* itself there is always the option to build the notification service from the source code repository. This is only recommended if you also built your *Fledge* from source code, if you did not then you should first do this before building the notification, otherwise you should install a binary package of the notification service.

The steps involved in building the notification service, assuming you have already built Fledge itself and the environment variable *FLEDGE\_ROOT* points to where you built your *Fledge*, are;

```
$ git clone https://github.com/fledge-iot/fledge-service-notification.git
...
$ cd fledge-service-notification
$ ./requirements.sh
...
$ mkdir build
$ cd build
$ cmake ..
...
$ make
...
```

This will result in the creation of a notification service binary, you now need to copy that binary into the *Fledge* installation. There are two options here, one if you used *make install* to create your installation and the other if you are running directly from the build environment.

If you used *make install* to create your *Fledge* installation then simply run *make install* to install your notification service. This should be run from the *build* directory under the *fledge-service-notification* directory.

```
$ make install
```



---

**Note:** You may need to run *make install* under a *sudo* command if your user does not have permissions to write to the installation directory. If you use a `DESTDIR=...` option to the *make install* of *Fledge* then you should use the same `DESTDIR=...` option here also.

---

If you are running your *Fledge* directly from the build environment, then execute the command

```
$ cp ./C/services/notification/fledge.services.notification $FLEDGE_ROOT/services
```

### Installing Notification Service Package

If you are using the packaged binaries for your system then you can use the package manager to install the *fledge-service-notification* package. The exact command depends on your package manager and how you obtained your packages.

If you downloaded your packages then you should navigate to the directory that contains your package files and run the package manager. If you have deb package files run the command

```
$ sudo apt -y install ./fledge-service-notification-1.7.0-armhf.deb
```

---

**Note:** The version number, 1.7.0 may be different on your system, this will depend which version you have downloaded. Also the armhf may be different for your machine architecture. Verify the precise name of your package before running the above command.

---

If you are using a RedHat or CentOS distribution and have rpm package files then run the command

```
$ sudo yum -y localinstall ./fledge-service-notification-1.7.0-x86_64.deb
```

---

**Note:** The version number, 1.7.0 may be different on your system, this will depend which version you have downloaded. Verify the precise name of your package before running the above command.

---

If you have configured your system to search a package repository that contains the Fledge packages then you can simply run the command

```
$ sudo apt-get -y install fledge-service-notification
```

On a Debian/Ubuntu system, or

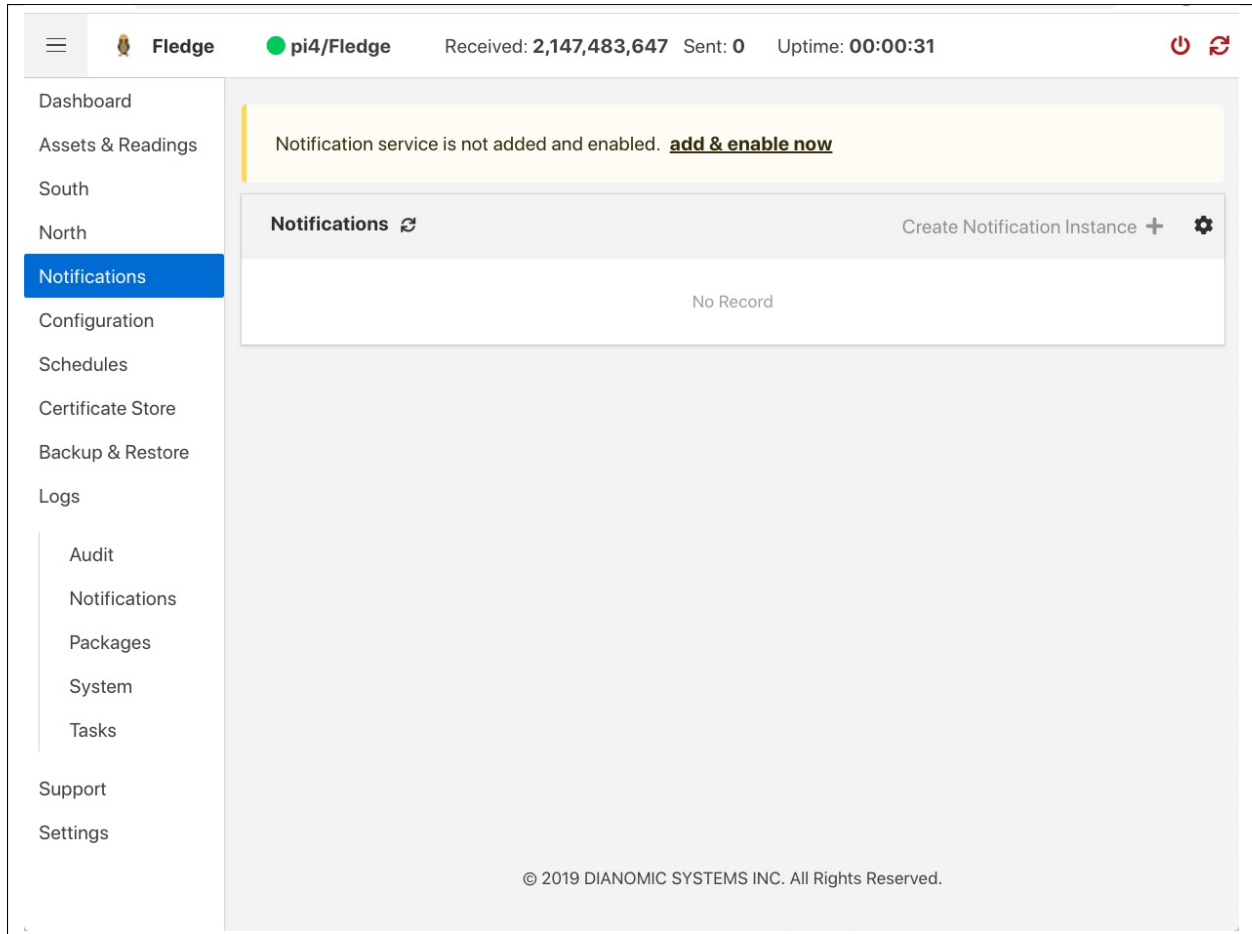
```
$ sudo yum -y install fledge-service-notification
```

On a RedHat/CentOS system. This will install the latest version of the notification service on your machine.



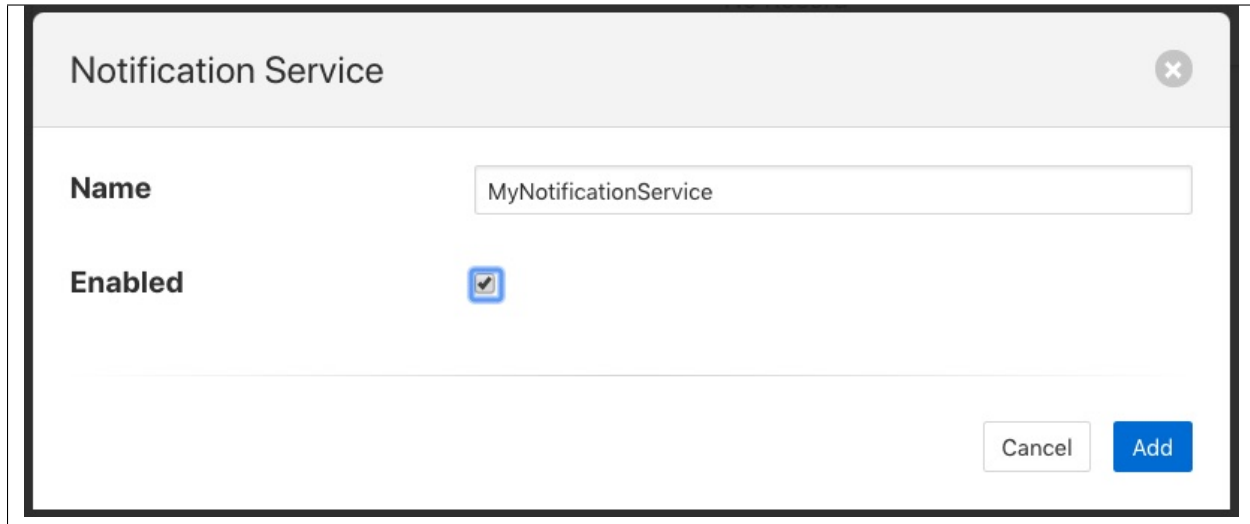
### 6.1.3 Starting The Notification Service

Once installed you must configure Fledge to start the notification service. This is simply done from the GUI by selecting the *Notifications* option from the left-hand menu. In the page that is then shown you will see a panel at the top that allows you to *add & enable now* the notification service. This only appears if one has not already be added.



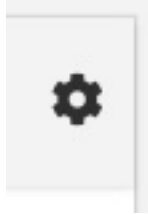
Select this link to *add & enable now* the notification service, a new dialog will appear that allows you to name and enable your service.



A screenshot of a 'Notification Service' configuration dialog box. The dialog has a title bar with the text 'Notification Service' and a close button (an 'x' in a circle) on the right. Below the title bar, there are two fields: 'Name' with a text input containing 'MyNotificationService', and 'Enabled' with a checked checkbox. At the bottom right of the dialog, there are two buttons: 'Cancel' and 'Add'.

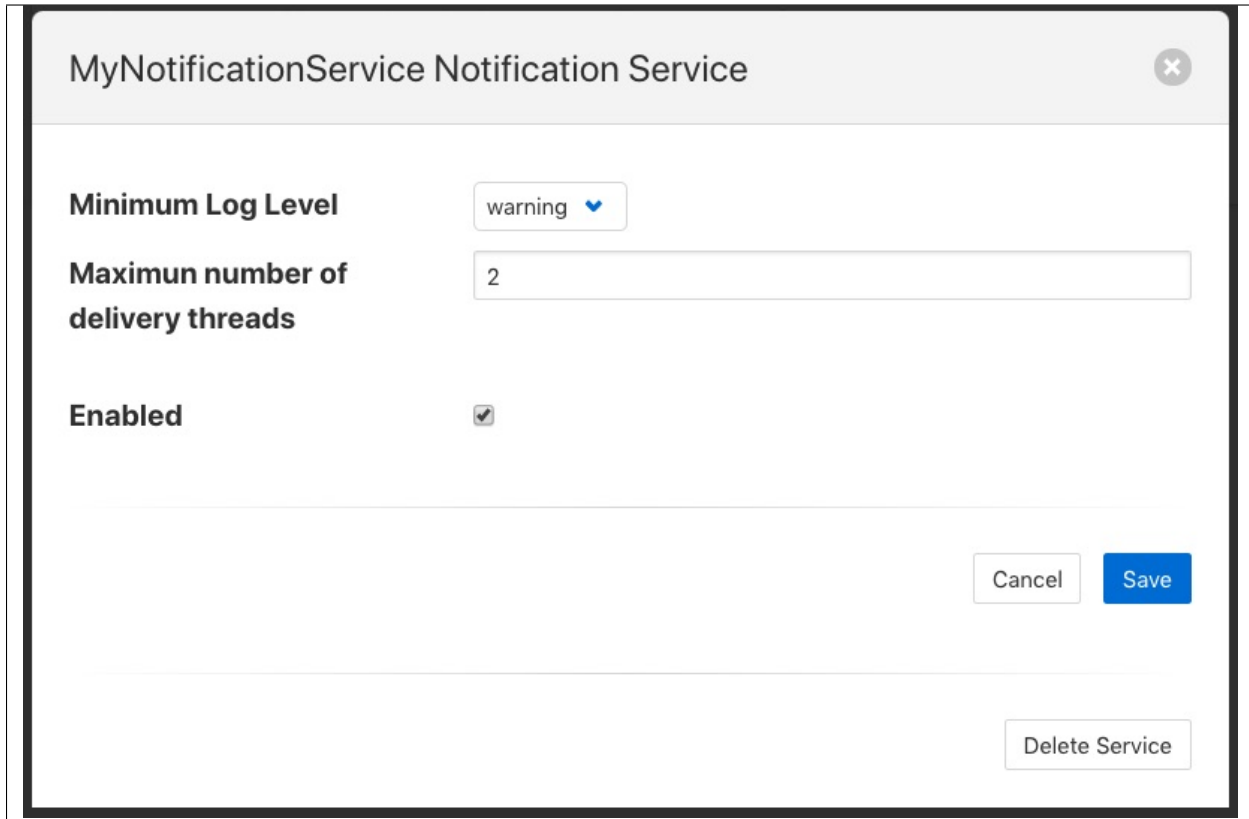
### 6.1.4 Configuring The Notification Service

Once the notification service has been added and enabled a new icon will appear in the *Notifications* page that allows you to configure the notification service. The icon appears in the top right and is in the shape of a gear wheel.



Clicking on this icon will display the notification service configuration dialog.



The image shows a configuration window titled "MyNotificationService Notification Service" with a close button (X) in the top right corner. The window contains three settings: "Minimum Log Level" with a dropdown menu showing "warning" and a blue arrow; "Maximun number of delivery threads" with a text input field containing the number "2"; and "Enabled" with a checked checkbox. At the bottom right, there are three buttons: "Cancel", "Save" (highlighted in blue), and "Delete Service".

MyNotificationService Notification Service

Minimum Log Level warning ▾

Maximun number of delivery threads 2

Enabled ☒

Cancel Save

Delete Service

You can use this dialog to control the level of logging that is done from the service by setting the *Minimum Log Level* to the least severity log level you wish to see. All log entries at the select level and of greater severity will be logged.

It is also possible to set the number of threads that will be used for delivering notifications. This defines how many notifications can be delivered in parallel. This only needs to be increased if the delivery process of any of the in use delivery plugins are long running.

The final setting allows you to disable the notification service.

Once you have updated the configuration of the service click on *Save*.

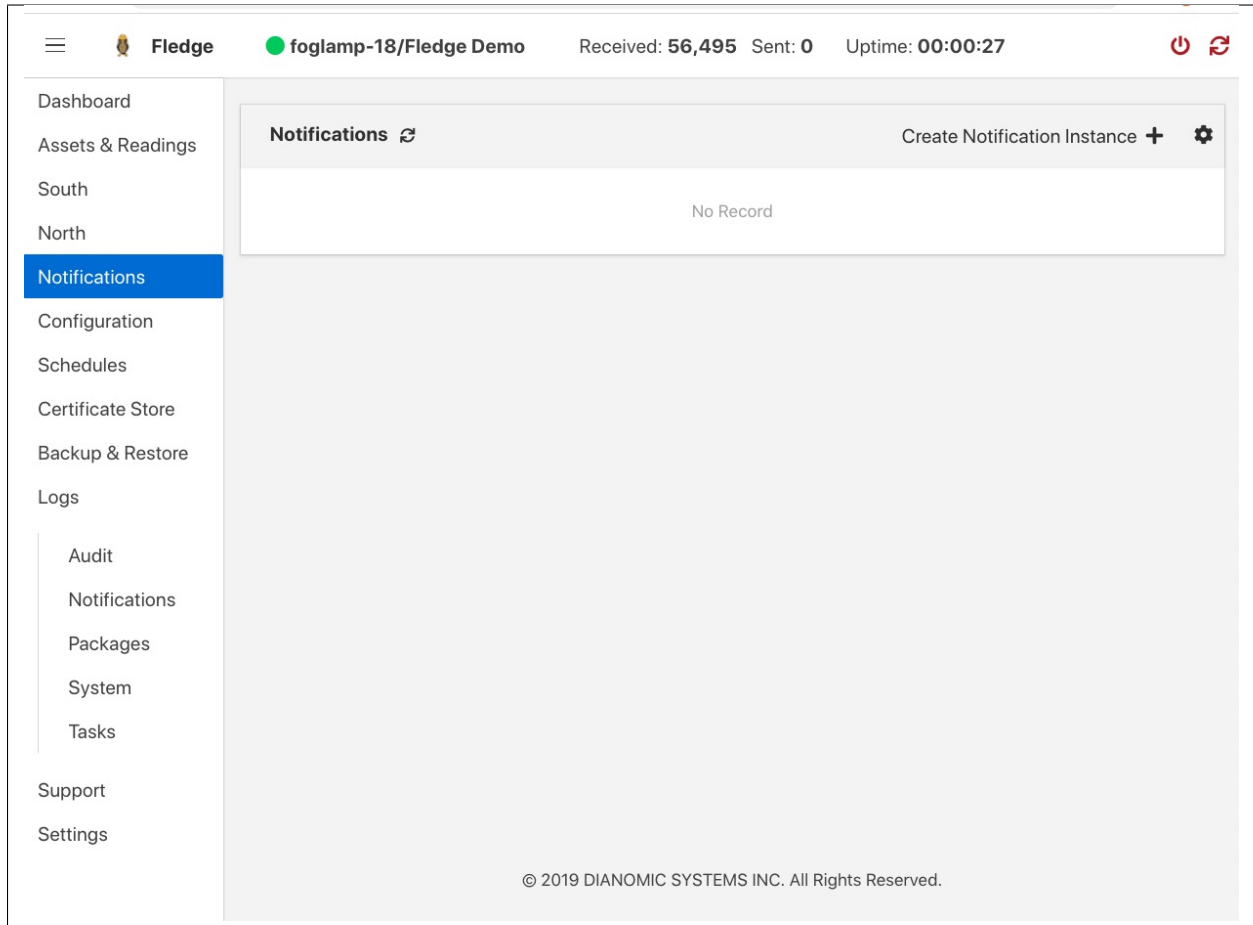
It is also possible to delete the notification service using the *Delete Service* button at the bottom of this dialog.

## 6.1.5 Using The Notification Service

### Add A Notification

In order to add s notification, select the Notifications page in the left-hand menu, an empty set of notifications will appear.





Click on the + icon to add a new notification.



Dashboard

Assets & Readings

South

North

Notifications

Configuration

Schedules

Certificate Store

Backup & Restore

Logs

Audit

Notifications

Packages

System

Tasks

Support

Settings

Received: 92,998 Sent: 0 Uptime: 18:22:45

1 Notification Instance 2 Rule 3 Delivery Channel 4 Done

Name

Description

Back Next

© 2019 DIANOMIC SYSTEMS INC. All Rights Reserved.

You will be presented with a dialog to enter a name and description for your notification.

1 Notification Instance 2 Rule 3 Delivery Channel 4 Done

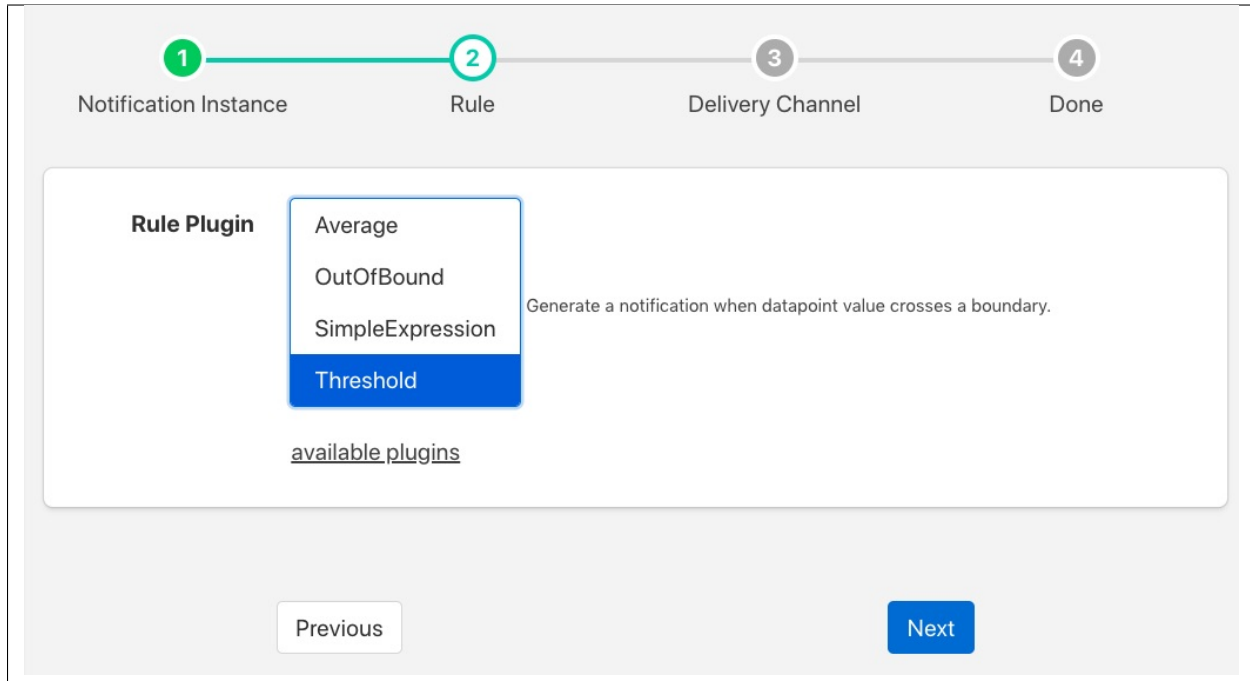
Name

Description

Back Next

Enter text for the name you require, a suggested description will be automatically added, however you can modify this to any string you desire. When complete click on the *Next* button to move forwards in the definition process. You can always click on *Previous* to go back a screen and modify what has been entered.





You are presented with the set of installed rules on the system. If the rule you wish to use is not installed and you wish to install it then use the link *available plugins* to be presented with the list of plugins that are available to be installed.

**Note:** The *available plugins* link will only work if you have added the Fledge package repository to the package manager of your system.

When you select a rule plugin a short description of what the rules does will be displayed to the right of the list. In this example we will use the threshold rule that is built into the notification service. Click on *Next* once you have selected the rule you wish to use.



1 Notification Instance 2 Rule 3 Delivery Channel 4 Done

**Asset name** FastSine

**Datapoint name** sinusoid

**Condition** >

**Trigger value** 0.5

**Evaluation data** Single Item

**Window evaluation** Average

**Time window** 30

Previous Next

You will be presented with the configuration parameters applicable to the rule you have chosen. Enter the name of the asset and the datapoint within that asset that you wish the rule to operate on. In the case of the *threshold rule* you can also define if you want the rule to trigger if the value is greater than, greater than or equal, less than or less than or equal to a *Trigger value*.

You can also choose to look at *Single Item* or *Window* data. If you choose the later you can then choose to define if the minimum, maximum or average within the window that must cross the threshold value.



1 Notification Instance 2 Rule 3 Delivery Channel 4 Done

**Asset name** FastSine

**Datapoint name** sinusoid

**Condition** >

**Trigger value** 0.5

**Evaluation data** Maximum Minimum **Average**

**Window evaluation** ☒

**Time window** 30

Previous Next

Once you have set the parameters for the rule click on the *Next* button to select the delivery plugin to use to delivery the notification data.

1 Notification Instance 2 Rule 3 Delivery Channel 4 Done

**Delivery Plugin**

- alexa
- asset
- Blynk
- email

[available plugins](#)

Previous Next

A list of available delivery plugins will be presented, along with a similar link that allows you to install new delivery



plugins if desired. As you select a plugin a short text description will be displayed to the right of the plugin list. In this example we will select the *Slack* messaging platform for the delivery of the notification.

Once you have selected the plugin you wish to use click on the *Next* button.

The screenshot shows a four-step progress bar at the top: 1 Notification Instance, 2 Rule, 3 Delivery Channel (active), and 4 Done. Below the progress bar is a configuration form for a Slack Webhook. It includes a 'Slack Webhook URL' field with a long URL, a 'Message Text' field with the text 'The value of the sinusoid is greater than 0.5', and an 'Enabled' checkbox which is checked. At the bottom of the form are 'Previous' and 'Next' buttons.

You will then be presented with the configuration parameters the delivery plugin requires to deliver the notification. In the case of the *Slack* plugin this consists of the webhook that you should obtain from the *Slack* application and a message text that will be sent when the event triggers.

**Note:** You may disable the delivery of a notification separately to enabling or disabling the notification. This allows you to test the logic of a notification without delivering the notification. Entries will still be made in the notification log when delivery is disabled.

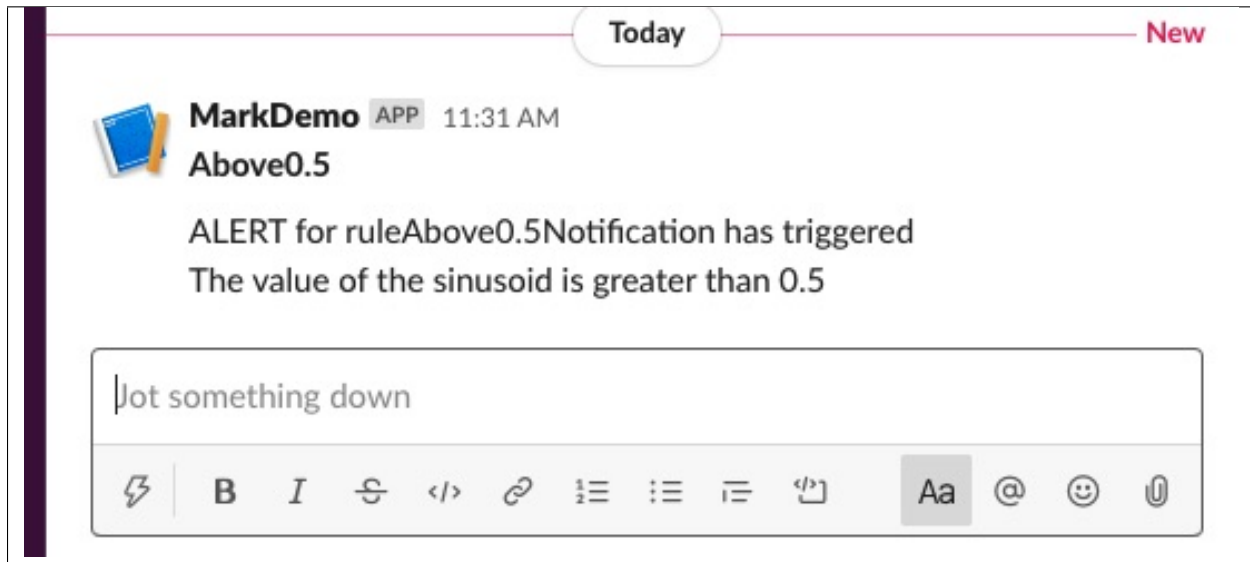
Once you have completed the configuration of the delivery plugin click on *Next* to move to the final stage in setting up your notification.

The screenshot shows the same four-step progress bar, but now step 4 'Done' is active. The configuration form below it has a 'Trigger' dropdown menu set to 'one shot', with a list of options (one shot, retriggered, toggled) visible. There is an 'Enabled' checkbox which is checked. A 'Done' button is located at the bottom right of the form.

The final stage of setting up your configuration is to set the notification type and the retrigger time for the notification. Enable the notification and click on *Done* to complete setting up your notification.



After a period of time, when a *sinusoid* value greater than 0.5 is received, a message will appear in your *Slack* window.



This will repeat at a maximum rate defined by the *Retrigger Time* whenever a value of greater than 0,5 is received.

### Notification Log

You can see activity related to the notification service by selecting the *Notifications* option under *Logs* in the left-hand menu.



**Fledge**
● foglamp-18/Fledge Demo
 Received: **98,181** Sent: **0** Uptime: **00:08:34**

Dashboard
Assets & Readings
South
North
Notifications
Configuration
Schedules
Certificate Store
Backup & Restore
Logs

Audit

Notifications

Packages
System
Tasks

Support
Settings

**Notification Logs**

Count: 13

ALL

INFORMATION

Name

Timestamp	Name	Severity	Source
2020-04-21 10:34:57	Above0.5	INFORMATION	NTFSN
2020-04-21 10:33:57	Above0.5	INFORMATION	NTFSN
2020-04-21 10:32:57	Above0.5	INFORMATION	NTFSN
2020-04-21 10:31:57	Above0.5	INFORMATION	NTFSN
2020-04-21 10:28:49	MyNotificationService	INFORMATION	NTFST
2020-04-21 09:41:48	MyNotificationService	INFORMATION	NTFST
2020-04-21 09:20:15	MyNotificationService	INFORMATION	NTFST
2020-04-21 09:15:32	Above0.5	INFORMATION	NTFAD
2020-04-20 14:47:38	MyNotificationService	INFORMATION	NTFST
2020-04-20 14:28:17	MyNotificationService	INFORMATION	NTFST
2020-04-20 13:56:17	MyNotificationService	INFORMATION	NTFST
2020-04-20 13:54:27	MyNotificationService	INFORMATION	NTFST
2020-04-20 11:22:38	MyNotificationService	INFORMATION	NTFST

You may filter this output using the drop down menus along the top of the page. The list to the left defines the type of event that you filter, clicking on this list will show you the meaning of the different audit types.



**Notification Logs**

Count: 13

ALL

INFORMATION

Name

ALL
NTFDL - Notification Deleted
NTFAD - Notification Added
NTFSN - Notification Sent
NTFCL - Notification Cleared
NTFST - Notification Server Startup
NTFSD - Notification Server Shutdown

	Severity	Source
	INFORMATION	NTFSN
	INFORMATION	NTFSN
	INFORMATION	NTFSN
	INFORMATION	NTFSN
	INFORMATION	NTFST
	INFORMATION	NTFST
2020-04-21 09:20:15	MyNotificationService	INFORMATION NTFST
2020-04-21 09:15:32	Above0.5	INFORMATION NTFAD
2020-04-20 14:47:38	MyNotificationService	INFORMATION NTFST
2020-04-20 14:28:17	MyNotificationService	INFORMATION NTFST
2020-04-20 13:56:17	MyNotificationService	INFORMATION NTFST
2020-04-20 13:54:27	MyNotificationService	INFORMATION NTFST
2020-04-20 11:22:38	MyNotificationService	INFORMATION NTFST

## Editing Notifications

It is possible to update existing notifications or remove them using the *Notifications* option from the left-hand menu. Clicking on *Notifications* will bring up a list of the currently defined notifications within the system.



The screenshot shows the Fledge web interface. The top header includes the Fledge logo, the instance name 'foglamp-18/Fledge Demo', and system statistics: 'Received: 98,064', 'Sent: 0', and 'Uptime: 00:06:34'. A sidebar on the left contains a menu with items like Dashboard, Assets & Readings, South, North, Notifications (highlighted), Configuration, Schedules, Certificate Store, Backup & Restore, Logs, Audit, Notifications, Packages, System, Tasks, Support, and Settings. The main content area is titled 'Notifications' and features a table with one notification instance. The table has columns for Name, Channel, Rule, Type, and Status. The notification 'Above0.5' is listed with channel 'slack', rule 'Threshold', type 'one shot', and status 'enabled'. A 'Create Notification Instance' button is visible in the top right of the main area. The footer of the interface states '© 2019 DIANOMIC SYSTEMS INC. All Rights Reserved.'

Name	Channel	Rule	Type	Status
<a href="#">Above0.5</a>	slack	Threshold	one shot	enabled

Click on the name of the notification of interest to display the details of that notification and allow it to be edited.



Above0.5

Asset name

sinusoid

Datapoint name

sinusoid

Condition

>

Trigger value

0.5

Evaluation data

Single Item

Window evaluation

Maximum

Time window

30

Delivery Channel - slack

Slack Webhook URL

https://hooks.slack.com/services/T2GBZ52AF/BLH4E9VPX/SpTueiK9t73KSaNSe3

Message Text

The value of the sinusoid is greater than 0.5

Enabled

☒

Cancel

Save

A single page dialog appears that allows you to change any of the parameters of your notification.

**Note:** You can not change the rule plugin or delivery plugin you are using. If you wish to change either of these then you must delete this notification and create a new one with the desired plugins.

Once you have updated your notification click *Save* to action the changes.

If you wish to delete your notification this may be done by clicking the *Delete* button at the base of the dialog.



## FLEDGE CONTROL FEATURES

Fledge supports facilities that allows control of devices via the south service and plugins. This control is known as *set point control* as it is not intended for real time critical control of devices but rather to modify the behavior of a device based on one of many different information flows. The latency involved in these control operations is highly dependent on the control path itself and also the scheduling limitations of the underlying operating system. Hence the caveat that the control functions are not real time or guaranteed to be actioned within a specified time window.

### 7.1 Control Functions

There are two types of control function supported

- Modify the value in a device via the south service and plugin.
- Request the device to perform an action.

#### 7.1.1 Set Point

Setting the value within the device is known as a set point action in Fledge. This can be as simple as setting a speed variable within a controller for a fan or it may be more complete. Typically a south plugin would provide a set of values that can be manipulated, giving each a symbolic name that would be available for a set point command. The exact nature of these is defined by the south plugin.

#### 7.1.2 Operation

Operations, as the name implies provides a means for the south service to request a device to perform an operation, such as reset or re-calibrate. The names of these operations and any arguments that can be given are defined within the south plugin and are specific to that south plugin.

### 7.2 Control Paths

Set point control may be invoked via a number of paths with Fledge

- As the result of a notification within Fledge itself.
- As a result of a request via the Fledge public REST API.
- As a result of a control message flowing from a north side system into a north plugin and being routed onward to the south service.



Currently only the notification method is fully implemented within Fledge.

The use of a notification in the Fledge instance itself provides the fastest response for an edge notification. All the processing for this is done on the edge by Fledge itself.

### 7.2.1 Edge Based Control

Edge based control is the name we use for a class of control applications that take place solely within the Fledge instance at the edge. The data that is required for the control decision to be made is gathered in the Fledge instance, the logic to trigger the control action runs in the Fledge instance and the control action is taken within the Fledge instance. Typically this will involve one or more south plugins to gather the data required to make the control decision, possibly some filters to process that data, the notification engine to make the decision and one or more south services to deliver the control messages.

As an example of how edge based control might work lets consider the following case.

We have a machine tool that is being monitored by Fledge using the OPC/UA south plugin to read data from the machine tools controlling PLC. As part of that data we receive an asset which contains the temperature of the motor which is running the tool. We can assume this asset is called *MotorTemperature* and it contains a single data point called *temperature*.

We also have a fan unit that is able to cool that motor which is controlled via a Modbus interface. The modbus contains one a coil that toggles the fan on and off and a register that controls the speed of the fan. We configure the *fledge-south-modbus* as a service called *MotorFan* with a control map that will map the coil and register to a pair of set points.

```
{
  "values" : [
    {
      "name" : "run",
      "coil" : 1
    },
    {
      "name" : "speed",
      "register" : 1
    }
  ]
}
```

**Control**

Use Control Map ▾

**Control Map**

1 ▾  
2 ▾  
3 ▾  
4  
5  
6  
7 ▾  
8  
9  
10  
11  
12

{  
 "values": [  
 {  
 "name": "run",  
 "coil": 1  
 },  
 {  
 "name": "speed",  
 "register": 1  
 }  
 ]  
}



If the measured temperature of the motor going above 35 degrees centigrade we want to turn the fan on at 1200 RPM. We create a new notification to do this. The notification uses the *threshold* rule and triggers if the asset *MotorTemperature*, data point *temperature* is greater than 35.

The screenshot displays the Fledge notification configuration interface. At the top, a progress bar shows four steps: 1. Notification Instance, 2. Rule (current step), 3. Delivery Channel, and 4. Done. The 'Rule' step is active, showing a configuration form for a threshold rule. The form includes the following fields:

- Asset name:** MotorTemperature
- Datapoint name:** temperature
- Condition:** >
- Trigger value:** 0.0
- Evaluation data:** Single Item
- Window evaluation:** Average
- Time window:** 30

At the bottom of the form, there are two buttons: 'Previous' and 'Next'.

We select the *setpoint* delivery plugin from the list and configure it.



The screenshot shows the Fledge Rule configuration interface. At the top, a progress bar indicates four steps: 1. Notification Instance, 2. Rule (current step), 3. Delivery Channel, and 4. Done. The main configuration area is divided into three sections: Service, Trigger Value, and Cleared Value. The Service section has a text input field containing 'MotorFan'. The Trigger Value section contains a JSON object: 

```
{  "values": {    "run": "1",    "speed": 1200  }}
```

. The Cleared Value section contains a JSON object: 

```
{  "values": {    "run": "0"  }}
```

. At the bottom left, there is an 'Enabled' checkbox which is checked. At the bottom right, there are 'Previous' and 'Next' buttons.

- In *Service* we set the name of the service we are going to use to control the fan, in this case *MotorFan*
- In *Trigger Value* we set the control message we are going to send to the service. This will turn the fan on and set the speed to 1200RPM
- In *Cleared Value* we set the control message we are going to send to turn off the fan when the value falls below 35 degrees.

The plugin is enabled and we go on to set the notification type to toggled, since we want to turn off the fan if the motor cools down, and set a retrigger time to prevent the fan switching on and off too quickly. The notification type and the retrigger time are important parameters for tuning the behavior of the control system and are discussed in more detail below.

If we required the fan to speed up at a higher temperature then this could be achieved with a second notification. In this case it would have a higher threshold value and would set the speed to a higher value in the trigger condition and set it back to 1200 in the cleared condition. Since the notification type is *toggled* the notification service will ensure that these are called in the correct order.



## Data Substitution

There is another option that can be considered in our example above that would allow the fan speed to be dependent on the temperature, the use of data substitution in the *setpoint* notification delivery.

Data substitution allows the values of a data point in the asset that caused the notification rule to trigger to be substituted into the values passed in the set point operation. The data that is available in the substitution is the same data that is given to the notification rule that caused the alert to be triggered. This may be a single asset with all of its data points for simple rules or may be multiple assets for more complex rules. If the notification rule is given averaged data then it is these averages that will be available rather than the individual values.

Parameters are substituted using a simple macro mechanism, the name of an asset and data point with in the asset is inserted into the value surrounded by the \$ character. For example to substitute the value of the *temperature* data point of the *MotorTemperature* asset into the *speed* set point parameter we would define the following in the *Trigger Value*

```
{
  "values" : {
    "speed" : "$MotorTemperature.temperature$"
  }
}
```

Note that we separate the asset name from the data point name using a period character.

This would have the effect of setting the fan speed to the temperature of the motor. Whilst allowing us to vary the speed based on temperature it would probably not be what we want as the fan speed is too low. We need a way to map a temperature to a higher speed.

A simple option is to use the macro mechanism to append a couple of 0s to the temperature, a temperature of 21 degrees would result in a fan speed of 2100 RPM.

```
{
  "values" : {
    "speed" : "$MotorTemperature.temperature$00"
  }
}
```

This works, but is a little primitive and limiting. Another option is to add data to the asset that triggers the notification. In this case we could add an expression filter to create a new data point with a desired fan speed. If we were to add an expression filter and give it the expression *desiredSpeed = temperature > 20 ? temperature \* 50 + 1200 : 0* then we would create a new data point in the asset called *desiredSpeed*. The value of *desiredSpeed* would be 0 if the temperature was 20 degrees or below, however for temperatures above it would be 1200 plus 50 times the temperature.

This new desired speed can then be used to set the temperature in the *setpoint* notification plugin.

```
{
  "values" : {
    "speed" : "$MotorTemperature.desiredSpeed$"
  }
}
```

The user then has the choice of adding the desired speed item to the data stored in the north, or adding an asset filter in the north to remove this data point from the data that is sent onward to the north.

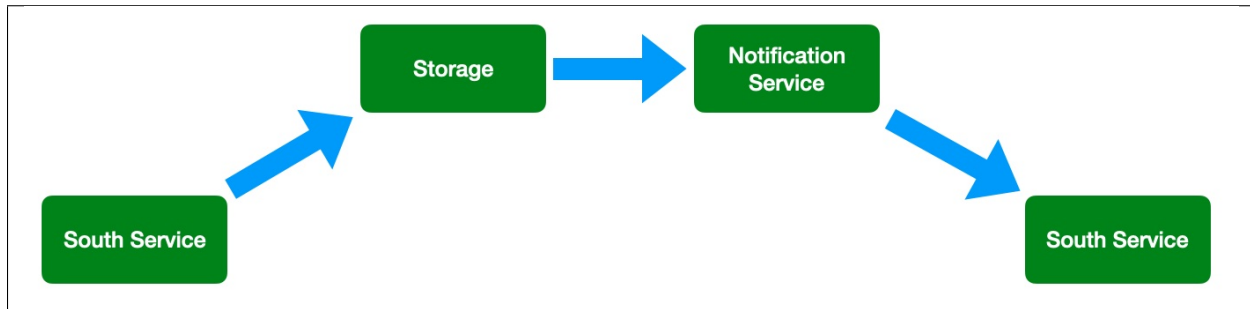


## Tuning edge control systems

The set point control features of Fledge are not intended to replace real time control applications such as would be seen in PLCs that are typically implemented in ladder logic, however Fledge does allow for high performance control to be implemented within the edge device. The precise latency in control decisions is dependent on a large number of factors and there are various tuning parameters that can be used to reduce the latency in the control path.

In order to understand the latency inherent in the control path we should first start by examining that path to discover where latency can occur. To do this we will choose a simple case of a single south plugin that is gathering data required by a control decision within Fledge. The control decision will be taken in a notification rule and delivered via the *fledge-notify-setpoint* plugin to another south service.

A total of four services within Fledge will be involved in the control path



- the south service that is gathering the data required for the decision
- the storage service that will dispatch the data to the notification service
- the notification service that will run the decision rule and trigger the delivery of the control message
- the south service that will send the control input to the device that is being controlled

Each of these services can add to that latency in the control path, however the way in which these are configured can significantly reduce that latency.

The south service that is gathering the data will typically be either polling a device or obtaining data asynchronously from the device. This will be sent to the ingest thread of the south service where it will be buffered before sending the data to the storage service.

The advanced settings for the south service can be used to trigger how often that data is sent to the storage service. Since it is the storage service that is responsible for routing the data onward to the notification service this impacts the latency of the delivery of the control messages.

<a href="#">Hide Advanced Config</a>	
Maximum Reading Latency (mS)	<input type="text" value="5000"/>
Maximum buffered Readings	<input type="text" value="100"/>
Reading Rate	<input type="text" value="1"/>
Throttle	<input type="checkbox"/>

The above shows the default configuration of a south service. In this case data will not be sent to the storage service until there are either 100 readings buffered in the south service, or the oldest reading in the south service buffer has been in the buffer for 5000 milliseconds. In this example we are reading 1 new readings every second, therefore will send data to the storage service every 5 seconds, when the oldest reading in the buffer has been there for 5000mS.



When it sends data it will send all the data it has buffered, in this case 5 readings as one block. If the oldest reading is the one that triggers the notification we have therefore introduced a 5 second latency into the control path.

The control path latency can be reduced by reducing the *Maximum Reading Latency* of this south plugin. This will of course put greater load on the system as a whole and should be done with caution as it increases the message traffic between the south service and the storage service.

The storage service has little impact on the latency, it is designed such that it will forward data it receives for buffering to the notification service in parallel to buffering it. The storage service will only forward data the notification service has subscribed to receive and will forward that data in the blocks it arrives at the storage service in. If a block of 5 readings arrives at the the storage service then all 5 will be sent to the notification service as a single block.

The next service in the edge control path is the notification service, this is perhaps the most complex step in the journey. The behavior of the notification service is very dependent upon how each individual notification instance has been configured, factors that are important are the notification type, the retrigger interval and the evaluation data options.

The notification type is used to determine when notifications are delivered to the delivery channel, in the case of edge control this might be the *setpoint* plugin or the *operation* plugin. Fledge implements three options for the notification type

- **One shot:** A one shot notification is sent once when the notification triggers but will not be resent again if the notification triggers on successive evaluations. Once the evaluation does not trigger, the notification is cleared and will be sent again the next time the notification rule triggers. One shot notifications may be further tailored with a maximum repeat frequency, e.g. no more than once in any 15 minute period.
- **Toggle:** A toggle notification is sent when the notification rule triggers and will not be resent again until the rule fails to trigger, in exactly the same way as a one shot trigger. However in this case when the notification rule first stops triggering a cleared notification is sent. Again this may be modified by the addition of a maximum repeat frequency.
- **Retriggered:** A retriggered notification will continue to be sent when a notification rule triggers. The rate at which the notification is sent can be controlled by a maximum repeat frequency, e.g. send a notification every 5 minutes until the condition fails to trigger.

It is very important to choose the right type of notification in order to ensure the data delivered in your set point control path is what you require. The other factor that comes into play is the *Retrigger Time*, this defines a dead period during which notifications will not be sent regardless of the notification type.

Setting a retrigger time that is too high will mean that data that you expect to be sent will not be sent. For example if you a new value you wish to be updated once every 5 seconds then you should use a retrigger type notification and set the retrigger time to less than 5 seconds.

It is very important to understand however that the retrigger time defines when notifications can be delivered, it does not related to the interval between readings. As an example, assume we have a retrigger time of 1 second and a reading that arrives every 2 seconds that causes a notification to be sent.

- If the south service is left with the default buffering configuration it will send the readings in a block to the storage service every 5 seconds, each block containing 2 readings.
- These are sent to the notification service in a single block of two readings.
- The notification will evaluate the rule against the first reading in the block.
- If the rule triggers the notification service will send the notification via the set point plugin.
- The notification service will now evaluate the rule against the second readings.
- If the rule triggers the notification service will note that it has been less than 1 second since it sent the last notification and it will not deliver another notification.



Therefore, in this case you appear to see only half of the data points you expect being delivered to you set point notification. In order to rectify this you must alter the tuning parameters of the south service to send data more frequently to the storage service.

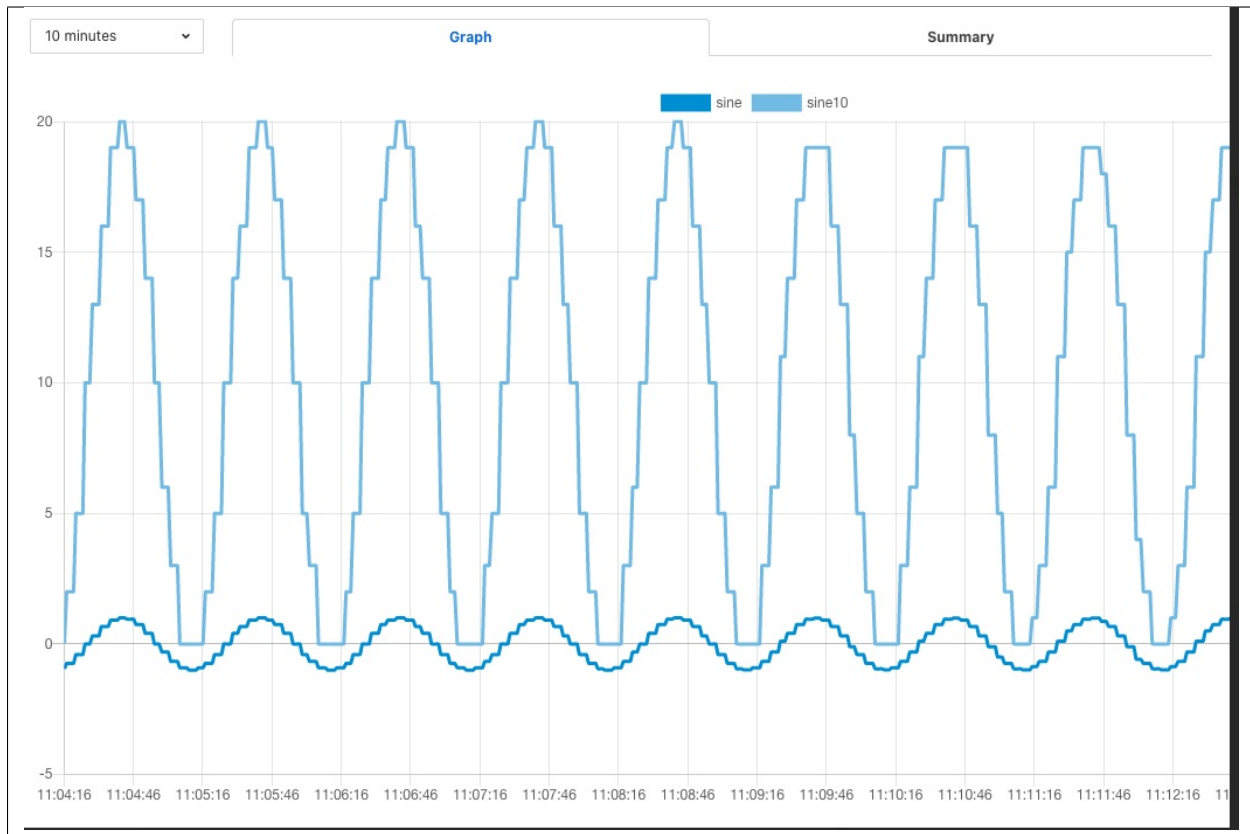
The final hop in the edge control path is the call from the notification service to the south service and the delivery via the plugin in the south service. This is done using the south service interface and is run on a separate thread in the south service. The result would normally be expected to be very low latency, however it should be noted that plugins commonly protect against simultaneous ingress and egress, therefore if the south service being used to deliver the data to the end device is also reading data from that device, there may be a requirement for the current read to complete before the write operation can commence.

To illustrate how the buffering in the south service might impact the data sent to the set point control service we will use a simple example of sine wave data being created by a south plugin and have every reading sent to a modbus device and then read back from the modbus device. The input data as read at the south service gathering the data is a smooth sine wave,



The data observed that is written to the modbus device is not however a clean sine wave as readings have been missed due to the retrigger time eliminating data that arrived in the same buffer.

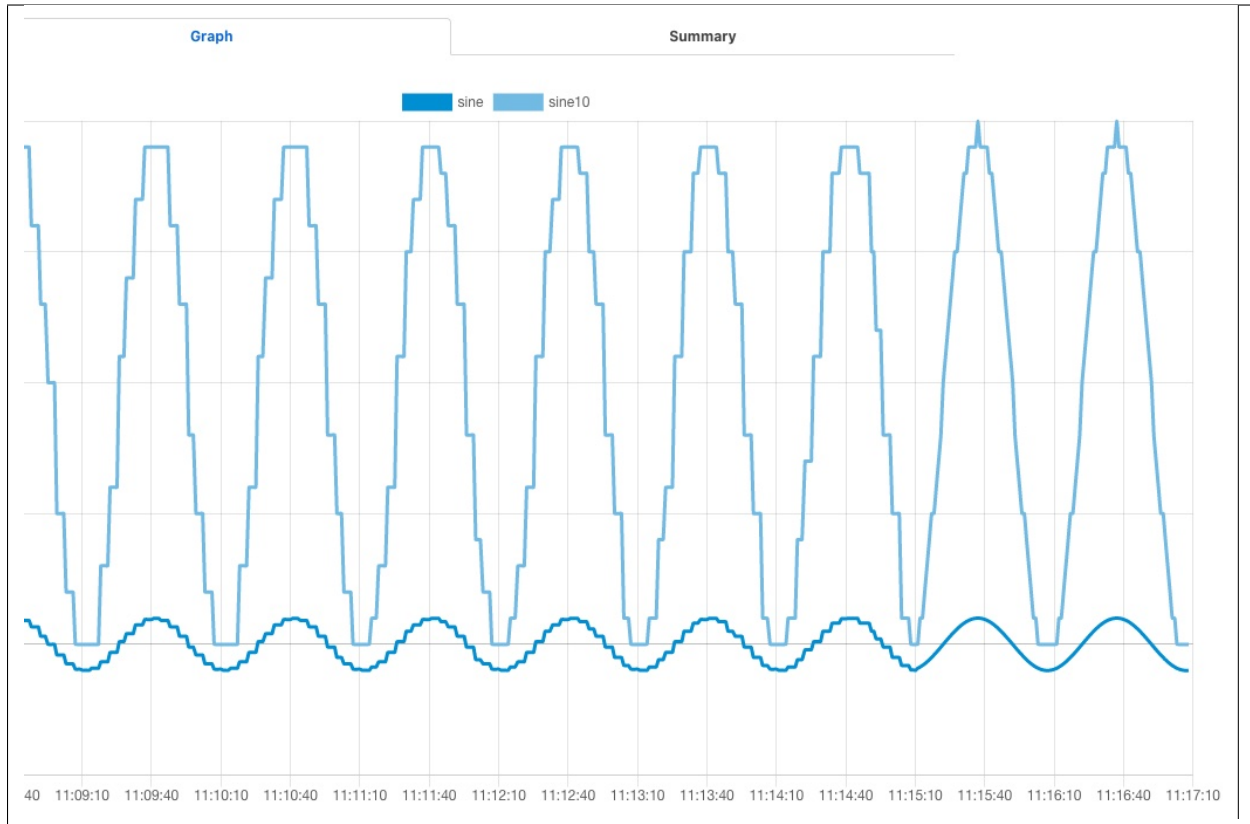




Some jitter caused by occasional differences in the readings that arrive in a single block can be seen in the data as well.

Changing the buffering on the south service to only buffer a single reading results in a much smooth sine wave as can be seen below as the data is seen to transition from one buffering policy to the next.

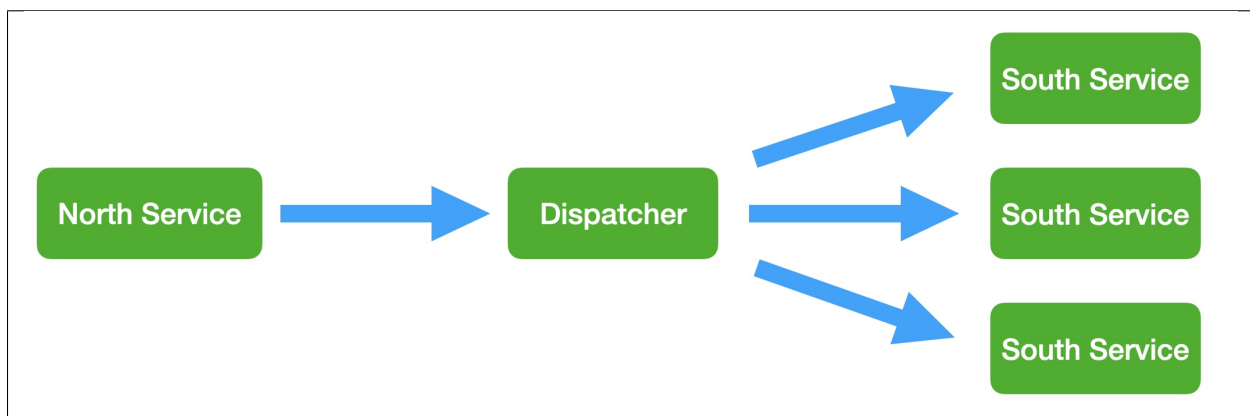




At the left end of the graph the south service is buffering 5 readings before sending data onward, on the right end it is only buffering one reading.

## 7.2.2 End to End Control

The end to end control path in Fledge is a path that allows control messages to enter the Fledge system from the north and flow through to the south. Both the north and south plugins involved in the path must support control operations, a dedicated service, the control dispatcher, is used to route the control messages from the source of the control input, the north service to the objects of the control operations, via the south service and plugins. Multiple south services may receive control inputs as a result of a single north control input.





It is the job of the north plugin to define how the control input is received, as this is specific to the protocol of device to the north of Fledge. The plugin then takes this input and maps it to a control message that can be routed by the dispatcher. The way this mappings is defined is specific to each of the north plugins that provide control input.

The control messages that the dispatcher is able to route are defined by the following set

- Write one or more values to a specified south service
- Write one or more values to the south service that ingests a specified asset
- Write one or more values to all south services supporting control
- Run an automation script within the dispatcher service
- Execution an operation on a specified south service
- Execute an operation on the south service that ingests a specified asset
- Execute an operation on all the south services that support control

An example of how a north plugin might define this mapping is shown below

Control Map

```

1 {
2   "nodes": [
3     {
4       "name": "test",
5       "type": "integer",
6       "destination" : "service",
7       "asset" : "fan0213"
8     }
9   ]
10  }

```

In this case we have an OPCUA north plugin that offers a writable node called *test*, we have defined this as accepting integer values and also set a destination of *service* and a name of *fan0213*. When the OPCUA node *test* is written the plugin will send a control message to the dispatcher to ask it to perform a write operation on the named service.

Alternately the dispatcher can send the request based on the assets that the south service is ingesting. In the following example, again taken from the OPCUA north plugin, we send a value of *EngineSpeed* which is an integer within the OPCUA server that Fledge presents to the service that is ingesting the asset *pump0014*.

Control Map

```

1 {
2   "nodes": [
3     {
4       "name": "EngineSpeed",
5       "type": "integer",
6       "asset" : "pump0014"
7     }
8   ]
9   }

```



If browsing the OPCUA server which Fledge is offering via the north service you will see a node with the browse name *EngineSpeed* which when written will cause the north plugin to send a message to the dispatcher service and ultimately cause the south service ingesting *pump0014* to have that value written to its *EngineSpeed* item. That south service need not be an OPCUA service, it could be any south service that supports control.

The screenshot shows the Fledge OPCUA interface. At the top, a status bar indicates 'Running' with the command `opc.tcp://foglamp-18.local:4840/fledge/server -- urn://fledge.dianomic.com`. On the left, a search pane shows a tree view with 'Objects' expanded, containing 'Booth1' and 'Control'. Under 'Control', 'EngineSpeed' is selected. The main pane, titled 'Attributes and References', displays a table of node properties.

Attribute	Value
NodeId	ns=2;i=2001
NodeClass	Variable
BrowseName	2:EngineSpeed
DisplayName	EngineSpeed
Description	EngineSpeed
WriteMask	1
UserWriteMask	1
Value	7
StatusCode	GOOD (0x00000000) ""
ServerTimestamp	02/09/22 15:01:47.48336...
SourceTimestamp	null
ServerPicoSeconds	0
SourcePicoSeconds	0
Value	7
Data Type	Int32
ValueRank	Scalar
ArrayDimensions	null
AccessLevel	CurrentRead
UserAccessLevel	CurrentRead
MinimumSamplingInterval	1.0
Historizing	false

It is also possible to get the dispatcher to send the control request to all services that support control. In the case of the OPCUA north plugin this is specified by omitting the other types of destination.



Control Map

```
1 {  
2   "nodes": [  
3     {  
4       "name": "EngineSpeed",  
5       "type": "integer"  
6     }  
7   ]  
8 }
```

All south services that support control will be sent the request, these may be of many different types and are free to ignore the request if it can not be mapped locally to a resource to update. The semantics of how the request is treated is determined by the south plugin, each plugin receiving the request may take different actions.

The dispatcher can also be instructed to run a local automation script, these are discussed in more detail below, when a write occurs on the OPCUA node via this north plugin. In this case the control map is passed a script key and name to execute. The script will receive the value *EngineSpeed* as a parameter of the script.

Control Map

```
1 {  
2   "nodes": [  
3     {  
4       "name": "EngineSpeed",  
5       "type": "integer",  
6       "script" : "SetPumpSpeed"  
7     }  
8   ]  
9 }
```

Note, this is an example and does not mean that all or any plugins will use the exact syntax for mapping described above, the documentation for your particular plugin should be consulted to confirm the mapping implemented by the plugin.

## 7.3 Control Dispatcher Service

The *control dispatcher* service is a service responsible for receiving control messages from other components of the Fledge system and taking the necessary actions against the south services in order to achieve the request result. This may be as simple as forwarding the write or operation request to one to more south services or it may require the execution of an automation script by the *dispatcher service*.



### 7.3.1 Forwarding Requests

The *service dispatcher* supports three forwarding regimes which may be used to either forward write requests or operation requests, these are;

- Forward to a single service using the name of the service. The caller of the dispatcher must provide the name of the service to which the request will be sent.
- Forward to a single service that is responsible for ingesting a named asset into the Fledge system. The caller of the dispatcher must provide the name of an asset, the *service dispatcher* will then look this asset up in the asset tracker database to determine which service ingested the named asset. The request will then be forwarded to that service.
- Forward the request to all south services that are currently running and that support control operations. Note that if a service is not running then the request will not be buffered for later sending.

### 7.3.2 Automation Scripts

The control dispatcher service supports a limited scripting designed to allow users to easily create sequences of operations that can be executed in response to a single control write operation. Scripts are created within Fledge and named externally to any control operations and may be executed by more than one control input. These scripts consist of a linear set of steps, each of which results in one of a number of actions, the actions supported are

- Perform a write request. A new write operation is defined in the step and it may take the form of any of the three styles of forwarding supported by the dispatcher; write to a named service, write to a service providing an asset or write to all south services.
- Perform an operation request on one or all south services. As with the write request above the three forwards of defining the target south service are defined.
- Delay the execution of a script. Add a delay between execution of the script steps.
- Update the Fledge configuration. Change the value of a configuration item within the system.
- Execute another script. A mechanism for calling another named script, the named script is executed and then the calling script will continue.

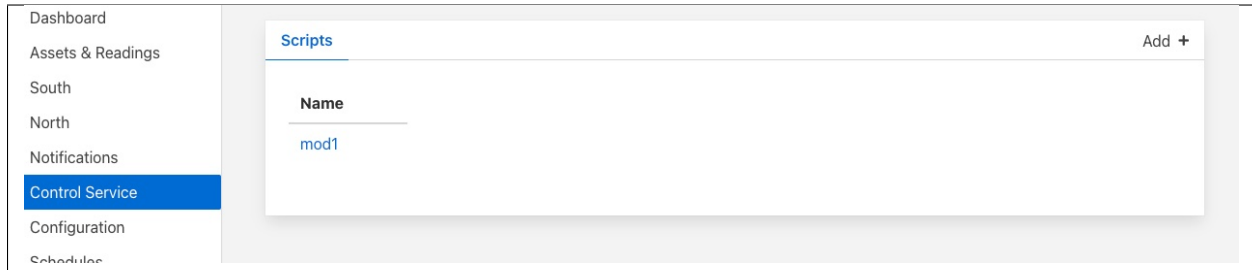
The same data substitution rules described above can also be used within the steps of an automation script. This allows data that is sent to the write or operation request in the dispatcher to be substituted in the steps themselves, for example a request to run a script with the values *param1* set to *value1* and *param2* set to *value2* would result in a step that wrote the value *\$param1\$* to a south service actually writing the value *value1*, i.e the value of *param1*.

Each step may also have associated with it a condition, if specified that condition must evaluate to true for the step to be executed. If it evaluates to false then the step is not executed and execution moves to the next step in the script.

### Graphical Interface

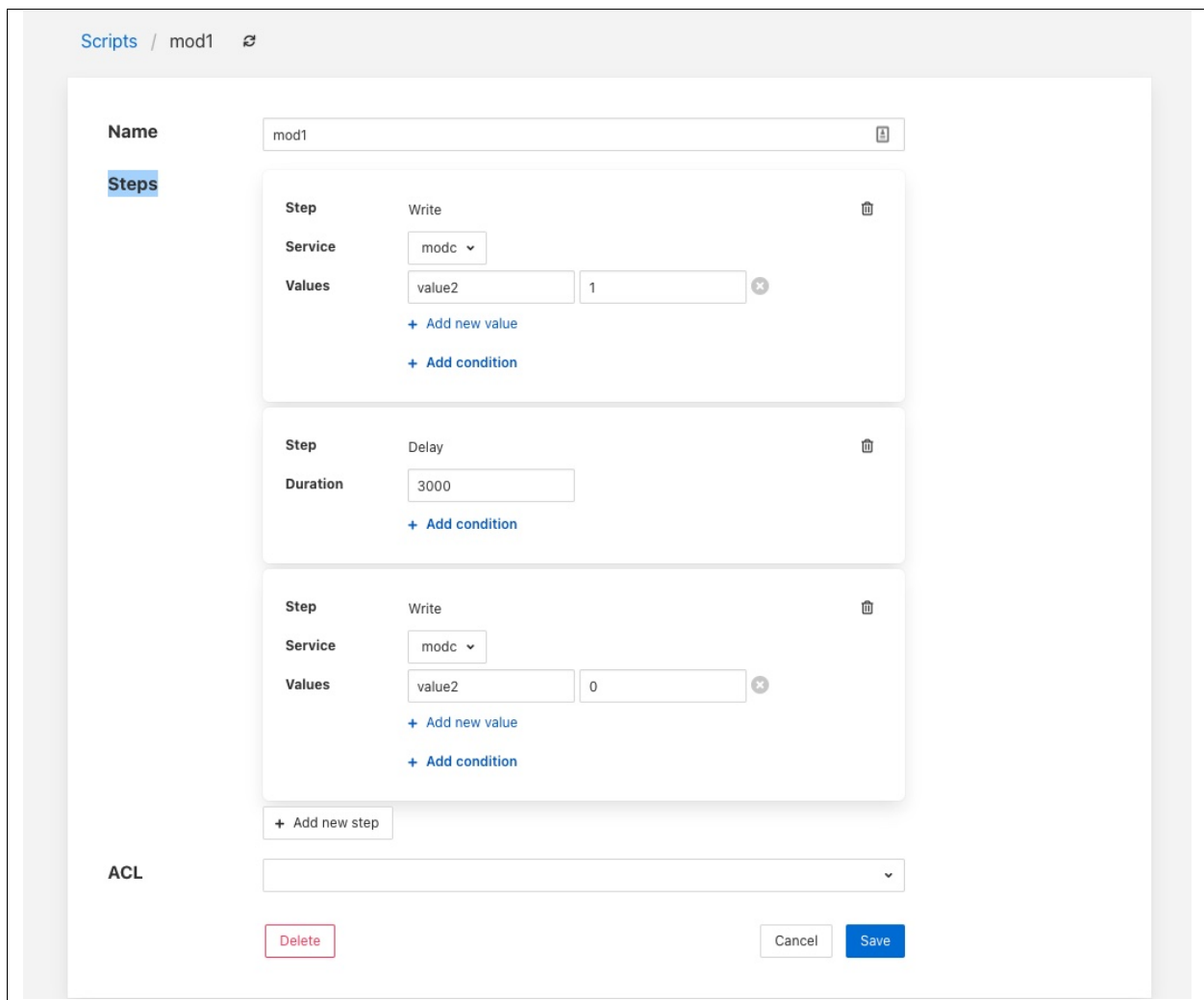
The automation scripts are available via the *Control Service* menu item in the left hand menu panel. Selecting this will give you access to the user interface associated with the control functions of Fledge. Click on the *Scripts* tab to select the scripts, this will display a list of scripts currently defined within the system and also show an add button icon in the top right corner.





## Viewing & Editing Existing Scripts

Simply click on the name of a script to view the script

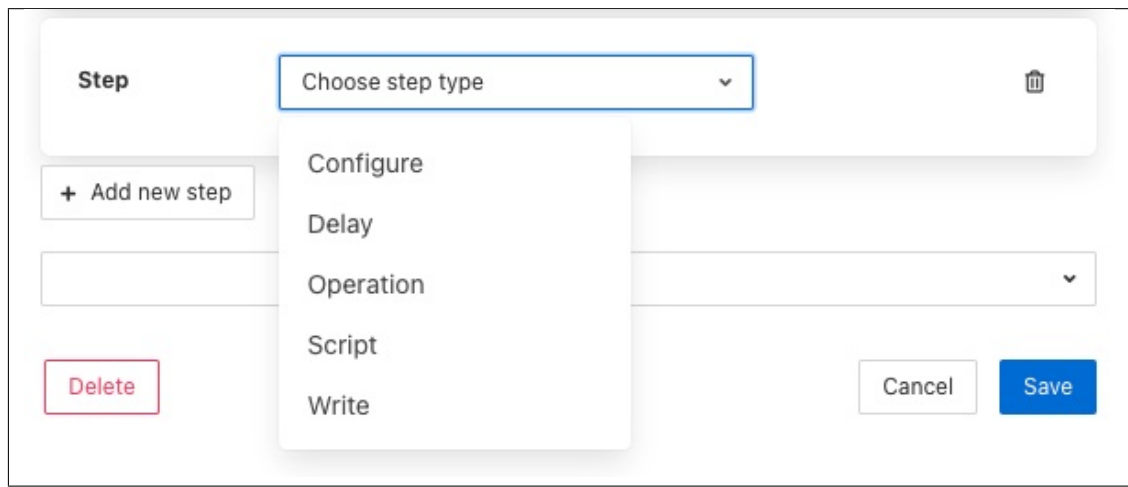


The steps within the script are each displayed within a panel for that step. The user is then able to edit the script provided they have permission on the script.

There are then a number of options that allow you to modify the script, note however it is not possible to change the type of a step in the script. The user must add a new step and remove the old step they wish to replace.

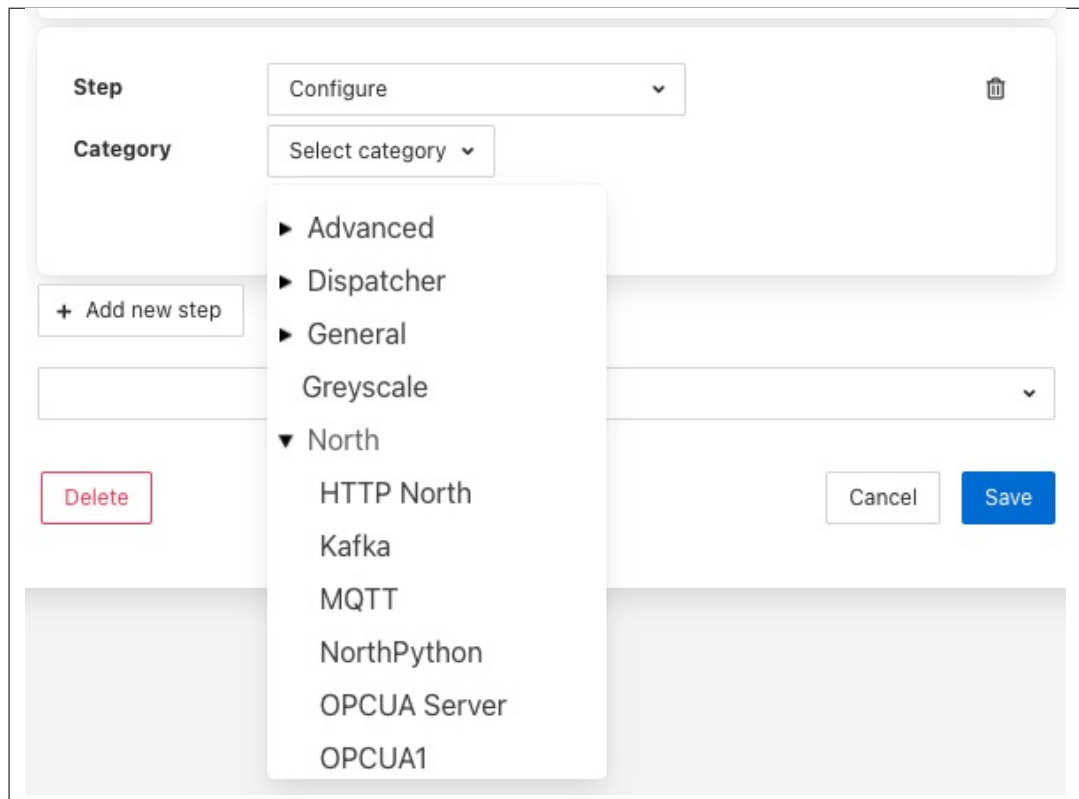


- To add a new step to a script click on the *Add new step* button
  - The new step be created in a new panel and will prompt for the user to select the step type



The screenshot shows a dialog box for adding a new step. At the top, there's a 'Step' label and a dropdown menu currently showing 'Choose step type'. A dropdown menu is open below it, listing five options: 'Configure', 'Delay', 'Operation', 'Script', and 'Write'. To the left of the dropdown is a '+ Add new step' button. Below that is a 'Delete' button. To the right of the dropdown menu are 'Cancel' and 'Save' buttons.

- The next step in to process will depend on the type of automation step chosen.
  - \* A *Configure* step will request the configuration category to update to be chosen. This is displayed in a drop down menu.



The screenshot shows a dialog box for configuring a step. The 'Step' dropdown is set to 'Configure'. Below it is a 'Category' dropdown set to 'Select category'. A tree structure is shown for categories: 'Advanced', 'Dispatcher', 'General', 'Greyscale', and 'North' (which is expanded). Under 'North', there are several sub-items: 'HTTP North', 'Kafka', 'MQTT', 'NorthPython', 'OPCUA Server', and 'OPCUA1'. To the left of the tree is a '+ Add new step' button. Below that is a 'Delete' button. To the right of the tree are 'Cancel' and 'Save' buttons.

The configuration categories are shown as a tree structure, allowing the user to navigate to the configuration category they wish to change.

Once chosen the user is presented with the items in that configuration category from which to choose.



The screenshot shows a configuration window for a step named 'Configure'. The 'Category' is set to 'Kafka'. The 'Config Item' dropdown menu is open, displaying a list of options: 'Simple plugin to send data to a Kafka topic' (which is highlighted), 'Bootstrap Brokers', 'Kafka Topic', 'Send JSON', 'Data Source', 'A switch that can be used to e...', 'Identifies the specific stream...', and 'Filter pipeline'. To the right of the dropdown, there is a text box containing the word 'Kafka'. Below the configuration fields, there is a '+ Add new step' button, a 'Delete' button, and 'Cancel' and 'Save' buttons.

Selecting an item will give you a text box with the current value of that item. Simply type the new value that should be assigned to that item when this step of the script runs into that text box.

- \* A *Delay* step will request the duration of the delay. The *Duration* is merely typed into the text box and is expressed in milliseconds.
- \* An *Operation* step will request you to enter the name of the operation to perform and then select the service to which the operation request should be sent



The screenshot shows a configuration window for a step. The 'Step' dropdown is set to 'Operation'. The 'Name' field contains 'Name'. The 'Service' dropdown is open, showing a list of services: modc, Sine, modbus2, Coolant, HTTPIn, Lathe, MQTTTest, Open62451, S2OPCUA, and Spinnaker. The 'Parameters' section is empty. At the bottom, there is a '+ Add new step' button, a 'Delete' button, and 'Cancel' and 'Save' buttons.

Operations can be passed zero or more parameters, to add parameters to an operation click on the *Add parameter* option. A pair of text boxes will appear allowing you to enter the key and value for the parameter.

The screenshot shows the same configuration window, but now the 'Service' dropdown is set to 'Lathe'. The 'Name' field contains 'shutdown'. The 'Parameters' section now has two input fields: 'key' and 'value'. Below these fields are two buttons: '+ Add new parameter' and '+ Add condition'.

To add another parameter simply press the *Add parameter* option again.

- \* A *Script* step will request you to choose the name of the script to run from a list of all the currently defined scripts.



The screenshot shows a configuration panel for a script step. The 'Step' dropdown is set to 'Script'. The 'Name' dropdown is open, showing 'Select script'. The 'Execution' dropdown is open, showing 'Unlock Lathe'. The 'Parameters' section has two links: '+ Add parameter' and '+ Add condition'. A trash icon is visible in the top right corner.

Note that the script that you are currently editing is not included in this list of scripts. You can then choose if you want the execution of this script to block the execution of the current script or to run in parallel with the execution of the current script.

The screenshot shows the same configuration panel, but the 'Execution' dropdown is now open, showing 'Select execution type'. The 'Name' dropdown is now closed, showing 'Unlock Lathe'. The 'Parameters' section is empty.

Scripts may also have parameters added by choosing the *Add parameter* option.

- \* A *Write* step will request you to choose the service to which you wish to send the write request. The list of available services is given in a drop down selection.

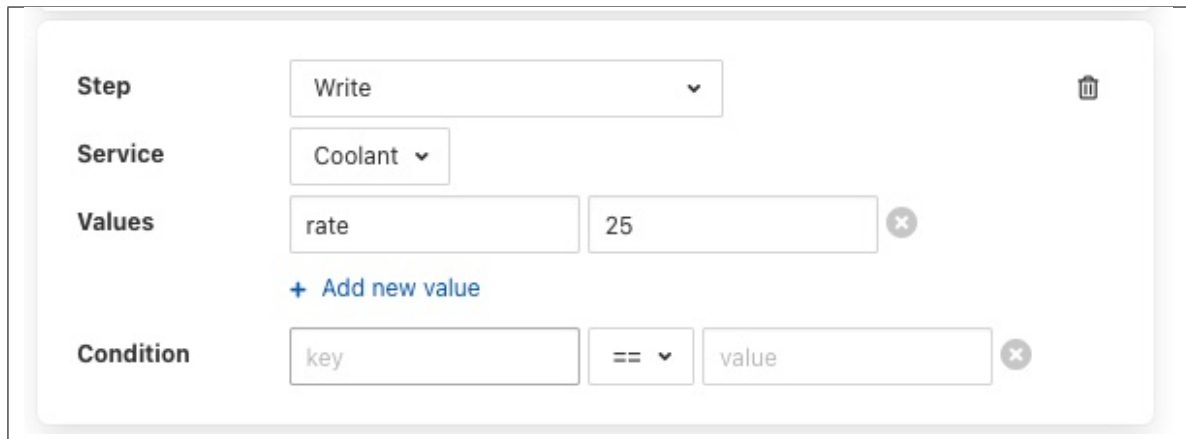


Values are added to the write request by clicking on the *Add new value* option. This will present a pair of text boxes in which the key and value of the write request value can be typed.

Multiple values can be sent in a single write request, to add another value simply click on the *Add new value* option again.

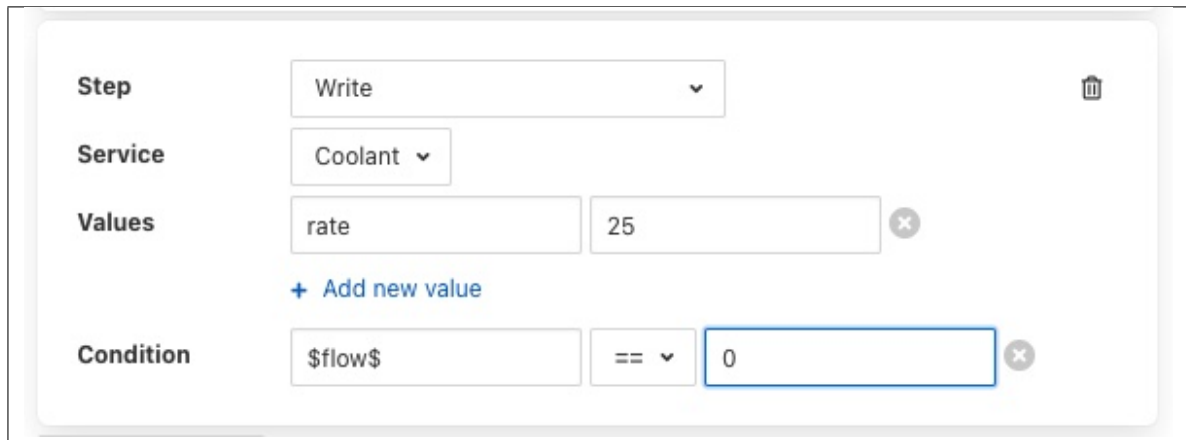
- Any step type may have a condition added to it. If a step has a condition associated with it, then that condition must evaluate to true if the step is to be run executed. If it does not evaluate to true the step is skipped and the next step is executed. To add a condition to a step click on the *Add condition* option within the step's panel.





The screenshot shows a configuration panel for a 'Write' step. The 'Service' is set to 'Coolant'. Under 'Values', there is a single entry with the key 'rate' and the value '25'. Below this, there is a link '+ Add new value'. Under 'Condition', there is a text box containing 'key', a dropdown menu showing '==', and a text box containing 'value'. A trash icon is visible in the top right corner of the panel.

A key and a value text box appears, type the key to test, this is usually a script parameter and the value to test. Script parameters are referenced using the \$ character to enclose the name of the script parameter.



This screenshot shows the same configuration panel as the previous one, but with changes to the 'Condition' section. The 'key' text box now contains '\$flow\$'. The operator dropdown still shows '=='. The 'value' text box now contains '0' and is highlighted with a blue border. The '+ Add new value' link is still present. The trash icon remains in the top right corner.

A selection list is provided that allows the test that you wish to perform to be chosen.



The screenshot shows a configuration panel for a 'Write' step. The 'Service' is set to 'Coolant'. Under 'Values', there is a field 'rate' with a value of '25'. Below this is a link '+ Add new value'. Under 'Condition', there is a field '\$flow\$' followed by a dropdown menu showing comparison operators. The selected operator is '==', and the value is '0'. A dropdown menu is open, showing the following operators: '==', '!=', '<', '>', '<=', and '>='. At the bottom left is a 'Delete' button. At the bottom right are 'Cancel' and 'Save' buttons.

- To remove a step from a script click on the bin icon on the right of the step panel

The screenshot shows a configuration panel for a 'Write' step. The 'Service' is set to 'modc'. Under 'Values', there is a field 'value2' with a value of '1'. Below this is a link '+ Add new value'. At the bottom is a link '+ Add condition'. A red circle highlights the bin icon in the top right corner of the step panel.

- To reorder the steps in a script it is a simple case of clicking on one of the panels that contains a step and dragging and dropping the step into the new position within the script in which it should run.



The screenshot shows the 'Scripts / mod1' configuration page. At the top, the 'Name' field is set to 'mod1'. Below it, the 'Steps' section contains three steps:

- Step 1: Write**
  - Service: modc
  - Values: value2, 1
  - + Add new value
  - + Add condition
- Step 2: Delay**
  - Duration: 3000
  - + Add condition
- Step 3: (partially obscured)**
  - Service: modc
  - Values: value2, 0
  - + Add new value
  - + Add condition

Below the steps, there is a '+ Add new step' button. At the bottom, the 'ACL' field is empty, and there are 'Delete', 'Cancel', and 'Save' buttons.

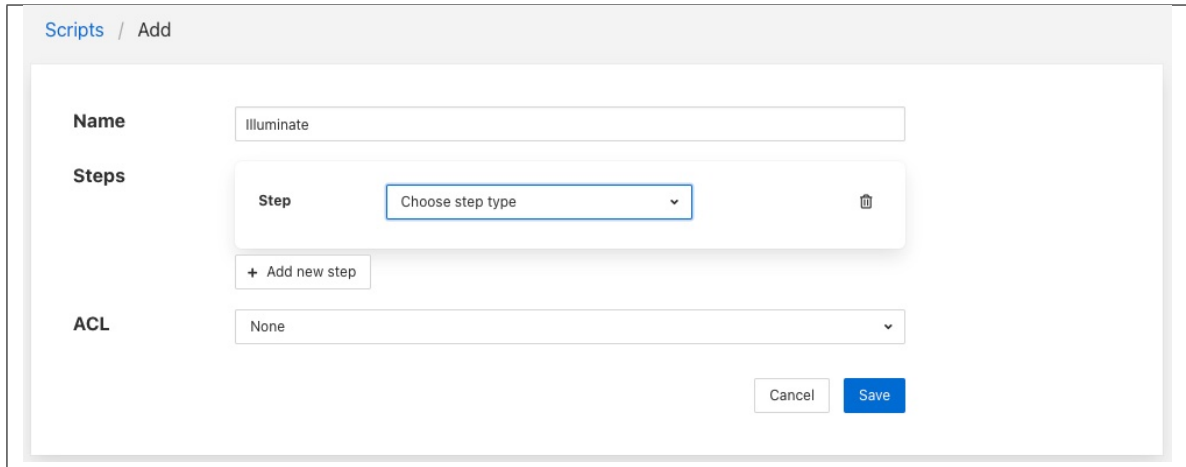
- A script may have an access control list associated to it. This controls how a script can be access, it allows the script to limit access to certain services, notifications or APIs. The creation of ACLs is covered elsewhere, to associate an ACL to a script simply select the name of the ACL from the ACL drop down at foot of the screen. If not ACL is assigned access to the script will not be limited.

## Adding a Script

The process for adding new scripts is similar to editing an existing script.

- To add a new script click on the *Add* option in the top right corner.
- Enter a name for the script in the text box that appears





- Now start to add the steps to your script in the same way as above when editing an existing script.
- Once you have added all your steps you may also add optional access control list
- Finally click on *Save* to save your script

### Step Conditions

The conditions that can be applied to a step allow for the checking of the values in the original request sent to the dispatcher. For example attaching a condition of the form

```
speed != 0
```

to a step, would result in the step being executed if the value in the parameter called *speed* that was in the original request to the dispatcher, had a value other than 0.

Conditions may be defined using the equals and not equals operators or for numeric values also greater than and less than.

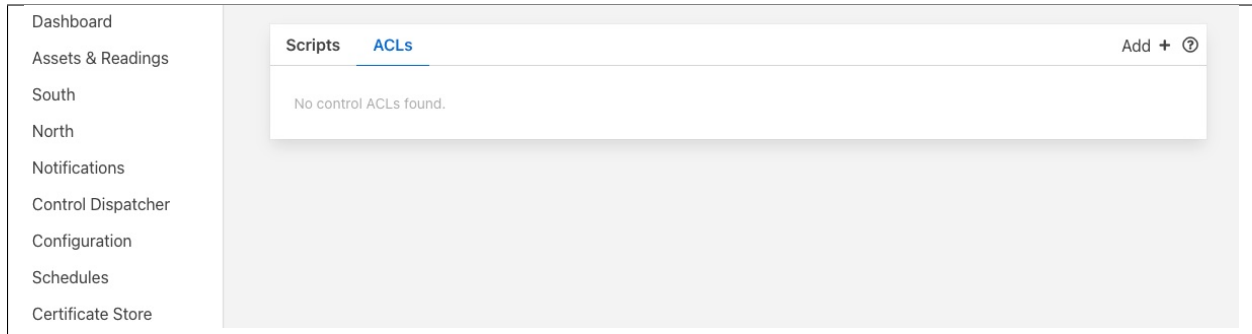
### 7.3.3 Access Control Lists

Control features within Fledge have the ability to add access control to every stage of the control process. Access control lists are used to limit which services have access to specific south services or scripts within the Fledge system.

#### Graphical Interface

A graphical interface is available to allow the creation and management of the access control lists used within the Fledge system. This is available within the *Control Dispatcher* menu item of the Fledge graphical interface.





## Adding An ACL

Click on the *Add* button in the top right corner of the ACL screen, the following screen will then be displayed.

You can enter a name for your access control list in the *Name* item on the screen. You should give each of your ACLs a unique name, this may be anything you like, but should ideally be descriptive or memorable as this is the name you will use when associating the ACL with services and scripts.

The *Services* section is used to define a set of services that this ACL is allowing access for. You may select services either by name or by service type. Multiple services may be granted access by a single ACL.



The screenshot shows the 'Services' configuration page. On the left, there are labels for 'Names', 'Types', 'URLs', and 'ACLs'. A dropdown menu is open, displaying a list of service names: 'Core', 'Fledge Core', 'Southbound', 'Sine', 'Press001', and 'Press002'. The 'Sine' service is currently selected and highlighted. To the right of the dropdown is a trash icon. At the bottom left, there is a button labeled '+ Add new URL'.

To add a named service to the ACL select the names drop down list and select the service name from those displayed. The display will change to show the service that you added to the ACL.

The screenshot shows the 'Services' configuration page. The 'Names' field now contains a tag 'x HTTP'. The 'Types' field has a dropdown menu with the text 'Select service type'.

More names may be added to the ACL by selecting the drop down again.

The screenshot shows the 'Services' configuration page. The 'Names' field now contains two tags: 'x HTTP' and 'x Press001'. The 'Types' dropdown menu is open, showing a list of service types: 'Sine', 'Press001', 'Press002', 'Press003', 'Northbound', and 'HTTP'. The 'Press001' service type is currently selected and highlighted.

If you wish to remove a named service from the list of services simply click on the small *x* to the left of the service name you wish to remove.

It is also possible to add a service type to an ACL. In this case all services of this type in the local Fledge instance will be given access via this ACL.



The screenshot shows the 'Services' configuration page. On the left, there are sections for 'Names', 'Types', 'URLs', and 'ACLs'. The 'Names' field has two entries: 'HTTP' and 'Press001'. The 'Types' dropdown menu is open, displaying a list of service types: 'Core', 'Storage', 'Southbound', 'Northbound', 'Notifications' (which is highlighted in blue), and 'Management'. Below the 'Types' dropdown is a button labeled '+ Add new URL'. The 'URLs' section on the left has input fields for 'URL' and 'ACLs'.

For example to create an ACL that allows all north services to have be granted access you would select *Northbound* in the *Services Types* drop down list.

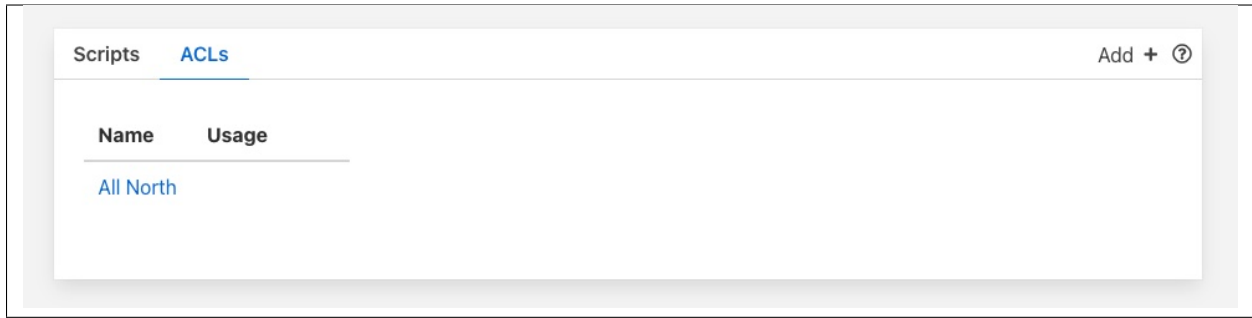
This screenshot shows the 'ACLs / Add' configuration page. The 'Name' field is filled with 'All North'. Under the 'Services' section, the 'Names' dropdown is set to 'Select service name' and the 'Types' dropdown is set to 'Select service type'. The 'Types' dropdown menu is open, showing the same list of service types as the previous image, with 'Northbound' highlighted in blue. The 'URLs' section on the left has input fields for 'URL' and 'ACLs', and a '+ Add new URL' button. At the bottom right, there are 'Cancel' and 'Save' buttons.

The *URLs* section of the ACL is used to grant access to specific URLs accessing the system.

**Note:** This is intended to allow control access via the REST API of the Fledge instance and is currently not implemented in Fledge.

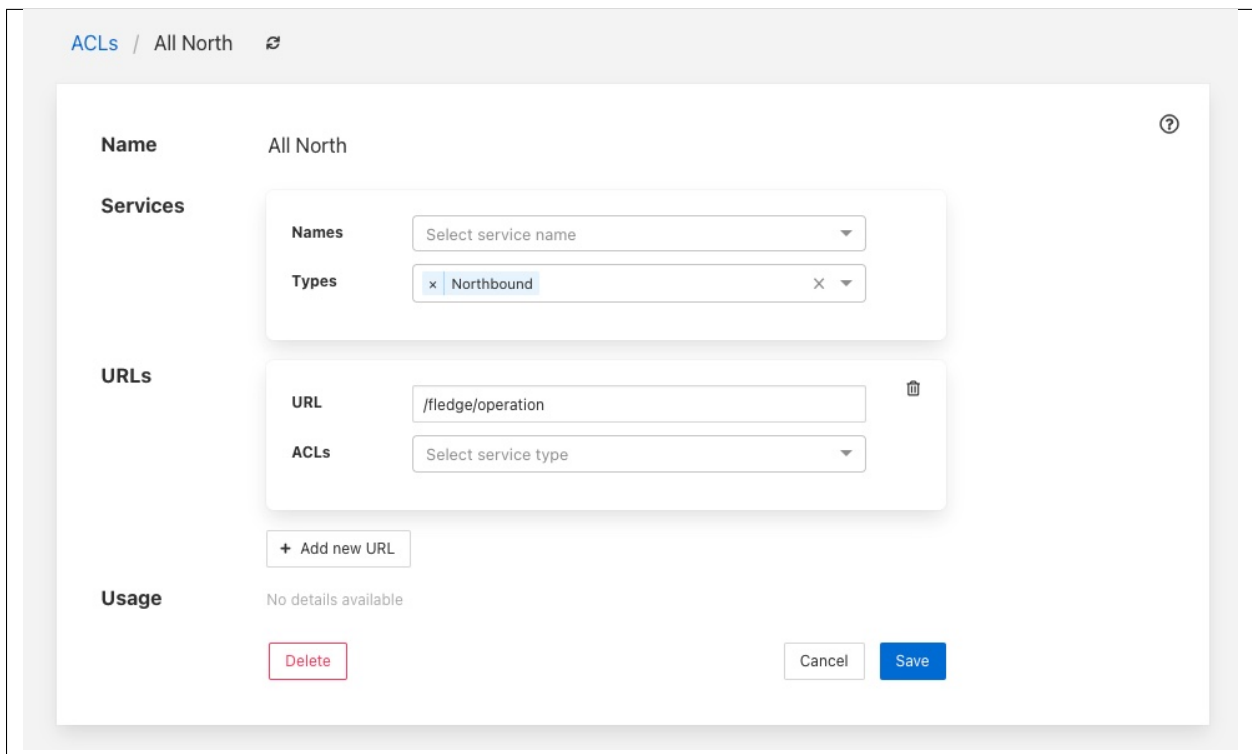
Once you are satisfied with the content of your access control list click on the *Save* button at the bottom of the page. You will be taken back to a display of the list of ACLs defined in your system.





## Updating an ACL

In the page that displays the set of ACLs in your system, click on the name of the ACL you wish to update, a page will then be displayed showing the current contents of the ACL.



To completely remove the ACL from the system click on the *Delete* button at the bottom of the page.

You may add and remove service names and types using the same procedure you used when adding the ACL.

Once you are happy with your updated ACL click on the *Save* button.



### 7.3.4 Configuration

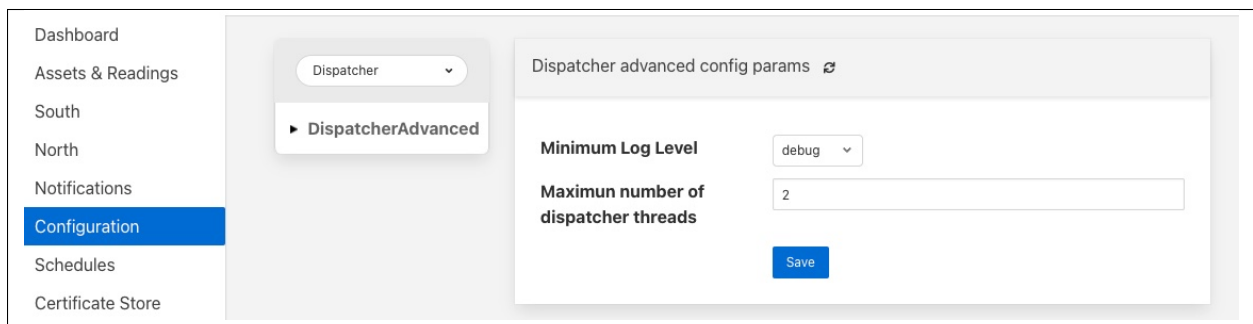
The *control dispatcher service* has a small number of configuration items that are available in the *Dispatcher* configuration category within the general Configuration menu item on the user interface.

Two subcategories exist, Server and Advanced.

#### Server Configuration

The server section contains a single option which can be used to either turn on or off the forwarding of control messages to the various services within Fledge. Clicking this option off will turn off all control message routing within Fledge.

#### Advanced Configuration



The screenshot displays the Fledge Configuration interface. On the left is a sidebar menu with the following items: Dashboard, Assets & Readings, South, North, Notifications, Configuration (highlighted in blue), Schedules, and Certificate Store. The main content area shows the 'Dispatcher' configuration category selected, with a sub-category 'DispatcherAdvanced' expanded. The 'Dispatcher advanced config params' section contains two settings: 'Minimum Log Level' set to 'debug' and 'Maximum number of dispatcher threads' set to '2'. A 'Save' button is located at the bottom right of the configuration panel.

- **Minimum Log Level:** Allows the minimum level at which logs will get written to the system log to be defined.
- **Maximum number of dispatcher threads:** Dispatcher threads are used to execute automation scripts. Each script utilizes a single thread for the duration of the execution of the script. Therefore this setting determines how many scripts can be executed in parallel.







## PLUGIN DOCUMENTATION

The following external plugins are currently available to extend the functionality of Fledge.

### 8.1 Fledge South Plugins

#### 8.1.1 AM2315 Temperature & Humidity Sensor



The *fledge-south-am2315* is a south plugin for a temperature and humidity sensor. The sensor connects via the I2C bus and can provide temperature data in the range  $-40^{\circ}\text{C}$  to  $+125^{\circ}\text{C}$  with an accuracy of  $0.1^{\circ}\text{C}$ .

The plugin will produce a single asset that has two data points; temperature and humidity.

---

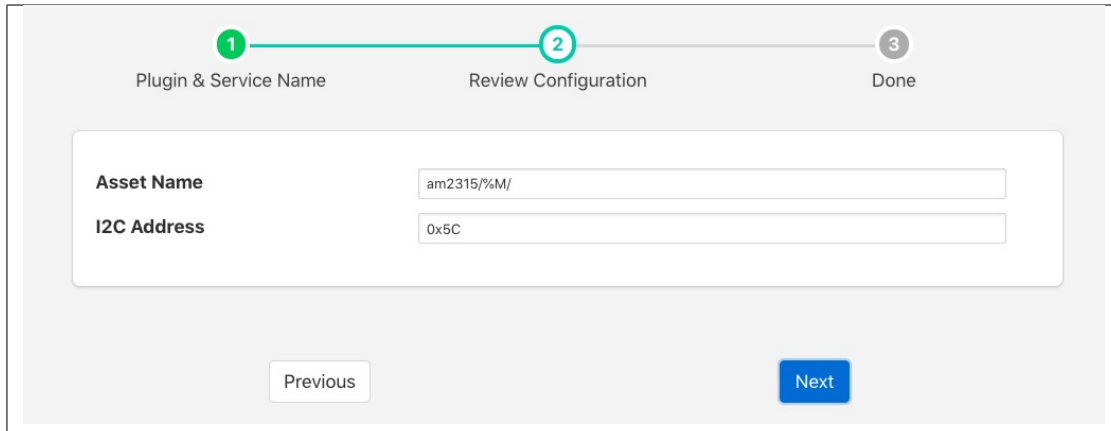
**Note:** The AM2315 is only available on the Raspberry Pi as it requires an I2C bus connection

---

To create a south service with the AM2315 plugin

- Click on *South* in the left hand menu bar
- Select *am2315* from the plugin list
- Name your service and click *Next*





1 Plugin & Service Name 2 Review Configuration 3 Done

Asset Name am2315/%M/

I2C Address 0x5C

Previous Next

- Configure the plugin
  - **Asset Name:** The name of the asset that will be created. To help when multiple AM2315 sensors are used a %M may be added to the asset name. This will be replaced with the I2C address of the sensor.
  - **I2C Address:** The I2C address of the sensor, this allows multiple sensors to be added to the same I2C bus.
- Click *Next*
- Enable the service and click on *Done*

## Wiring The Sensor

The following table details the four connections that must be made from the sensor to the Raspberry Pi GPIO connector.

Colour	Name	GPIO Pin	Description
Red	VDD	Pin 2 (5V)	Power (3.3V - 5V)
Yellow	SDA	Pin 3 (SDA)	Serial Data
Black	GND	Pin 6 (GND)	Ground
White	SCL	Pin 5 (SCL)	Serial Clock

### 8.1.2 CC2650 SensorTag





The *fledge-south-cc2650* is a plugin that connects using Bluetooth to a Texas Instruments . The SensorTag offers 10 sensors within a small, low powered package which may be read by this plugin and ingested into Fledge. These sensors include;

- ambient light
- magnetometer
- humidity
- pressure
- accelerometer
- gyroscope
- object temperature
- digital microphone

---

**Note:** The sensor requires that you have a Bluetooth low energy adapter available that supports at least BLE 4.0.

---

To create a south service with the

- Click on *South* in the left hand menu bar
- Select *cc2650* from the plugin list
- Name your service and click *Next*



1 Plugin & Service Name      2 Review Configuration      3 Done

Bluetooth Address	B0:91:22:EA:79:04
Asset Name Prefix	CC2650/%M/
Shutdown Threshold	10
Connection Timeout	3
Temperature Sensor	<input checked="" type="checkbox"/>
Temperature Sensor Name	temperature
Luminance Sensor	<input type="checkbox"/>
Luminance Sensor Name	luminance
Humidity Sensor	<input type="checkbox"/>
Humidity Sensor Name	humidity
Pressure Sensor	<input type="checkbox"/>
Pressure Sensor Name	pressure
Movement Sensor	<input type="checkbox"/>
Gyroscope Sensor Name	gyroscope
Accelerometer Sensor Name	accelerometer
Magnetometer Sensor Name	magnetometer
Battery Data	<input type="checkbox"/>
Battery Sensor Name	battery

Previous      Next

- Configure the plugin
  - **Bluetooth Address:** The Bluetooth MAC address of the device
  - **Asset Name Prefix:** A prefix to add to the asset name
  - **Shutdown Threshold:** The time in seconds allowed for a shutdown operation to complete
  - **Connection Timeout:** The Bluetooth connection timeout to use when attempting to connect to the device
  - **Temperature Sensor:** A toggle to include the temperature data in the data ingested
  - **Temperature Sensor Name:** The data point name to assign the temperature data
  - **Luminance Sensor:** Toggle to control the inclusion of the ambient light data
  - **Luminance Sensor Name:** The data point name to use for the luminance data
  - **Humidity Sensor:** A toggle to include the humidity data
  - **Humidity Sensor Name:** The data point name to use for the humidity data
  - **Pressure Sensor:** A toggle to control the inclusion of pressure data
  - **Pressure Sensor Name:** The name to be used for the data point that will contain the atmospheric pressure data



- **Movement Sensor:** A toggle that controls the inclusion of movement data gathered from the gyroscope, accelerometer and magnetometer
- **Gyroscope Sensor Name:** The data point name to use for the gyroscope data
- **Accelerometer Sensor Name:** The name of the data point that will record the accelerometer data
- **Magnetometer Sensor Name:** The name to use for the magnetometer data
- **Battery Data:** A toggle to control inclusion of the state of charge of the battery
- **Battery Sensor Name:** The data point name for the battery charge percentage
- Click *Next*
- Enable the service and click on *Done*

### 8.1.3 CoAP

The *fledge-south-coap* plugin implements a passive listener that will accept data from sensors implementing the CoAP protocol. CoAP is an Internet application protocol for constrained devices to send data over the internet, it is similar to HTTP but may be run over UDP or TCP and is considerably simplified to allow implementation in small footprint devices. CoAP stands for Constrained Application Protocol.

The plugin listens for POST requests to the URI defined in the configuration. It expects the content of this PUT request to be a CBOR payload which it will expand and create assets for the items read from the CBOR payload.

To create a south service with the CoAP plugin

- Click on *South* in the left hand menu bar
- Select *coap* from the plugin list
- Name your service and click *Next*

The screenshot shows the Fledge configuration interface for the CoAP plugin. At the top, there is a progress bar with three steps: 1. Plugin & Service Name, 2. Review Configuration, and 3. Done. Step 2 is currently active. Below the progress bar, there is a form with two input fields: 'Port' with the value '5683' and 'URI' with the value 'sensor-values'. At the bottom of the form, there are two buttons: 'Previous' and 'Next'.

- Configure the plugin
  - **Port:** The port on which the CoAP plugin will listen
  - **URI:** The URI the plugin expects to receive POST requests
- Click *Next*
- Enable the service and click on *Done*



### 8.1.4 Simple CSV Plugin

The *fledge-south-csv* plugin is a simple plugin for reading comma separated variable files and injecting them as if there were sensor data. There are a number of variants of plugin that support this functionality with varying degrees of sophistication. These may also be considered as simple examples of how to write plugin code.

This particular CSV reader supports single or multi-column CSV files, without timestamps in the file. It assumes every value is a data value. If the multi-column option is not set then it will read data from the file up until a newline or a comma character and make that as single data point in an asset and return that.

If the multi-column option is selected then each column in the CSV file becomes a data point within a single asset. It is assumed that every row of the CSV file will have the same number of values.

Upon reaching the end of the file the plugin will restart sending data from the beginning of the file.

To create a south service with the csv plugin

- Click on *South* in the left hand menu bar
- Select *Csv* from the plugin list
- Name your service and click *Next*

The screenshot shows the Fledge configuration interface for the CSV plugin. At the top, there is a progress bar with three steps: 1. Plugin & Service Name, 2. Review Configuration (which is the current step), and 3. Done. Below the progress bar, there is a configuration form with the following fields:

- Asset Name:** A text input field containing the value "Vibration".
- Datapoint:** A text input field containing the value "ch".
- Multi-Column:** A checkbox that is currently unchecked.
- Path Of File:** A text input field that is empty.

At the bottom of the form, there are two buttons: "Previous" and "Next". The "Next" button is highlighted in blue.

- Configure the plugin
  - **Asset Name:** The name of the asset that will be created
  - **Datapoint:** The name of the data point to insert. If multi-column is selected this becomes the prefix of the name, with the column number appended to create the full name
  - **Multi-Column:** If selected then each row of the CSV file is treated as a single asset with each column becoming a data point within that asset.
  - **Path Of File:** The file that should be read by the CSV plugin, this may be any location within the host operating system. The Fledge process should have permission to read this file.
- Click *Next*
- Enable the service and click on *Done*



### 8.1.5 CSV Playback

The plugin plays a csv file inside some given directory in file system (The default being FLEDGE\_ROOT/data). It converts the columns of csv file into readings which are datapoints of an output asset. The plugin plays readings at some configured rate.

We can also convert the columns of csv file into some other data type. For example from float to integer. The converted data will be part of reading not the CSV file.

The plugin has the ability to play the readings in either burst or continuous mode. In burst mode all readings are ingested into database at once and there is no adjustment of timestamp of a single reading. Whereas in continuous mode readings are ingested one by one and the timestamp of each reading is adjusted according to sampling rate. (For example if sampling rate is 8000 then the user\_ts of every reading differs by 125 micro seconds.)

We can also copy the timestamp if present in the CSV file. This time stamp becomes the user\_ts of a reading.

The plugin can also play the file in a loop which means it can start again if end of the file has reached.

The plugin can also play a file that has variable columns in every line.

The screenshot displays the 'Review Configuration' step of the Fledge CSV Playback plugin setup. The configuration parameters are as follows:

- Asset name:** vibration
- CSV directory name:** FLEDGE\_DATA
- CSV file pattern:** (empty)
- Header processing method:** do\_not\_skip
- Data point for header rows:** metadata
- Number of rows to skip or pass in datapoint:** 1
- Dynamic columns:** ☐
- Column processing method:** pick\_from\_file
- Auto generate prefix:** column
- Column names and data types for explicit:** (empty)

- **‘assetName’:** type: string default: **‘vibration’**: The output asset that contains the readings.
- **‘csvDirName’:** type: string default: **‘FLEDGE\_DATA’**: The directory where CSV file exists. Default is FLEDGE\_DATA or FLEDGE\_ROOT/data
- **‘csvFileName’:** type: string default: **‘’**: CSV file name or pattern to search inside directory. Not necessarily an exact file name. If there are multiple files matching with the pattern, then the plugin will pick the first file in alphabetical order. If postProcessMethod is rename or delete then it will rename or delete the played file and pick the next one and so on.
- **‘headerMethod’:** type: enumeration default: **‘do\_not\_skip’**: The method for processing the header of csv file.
  1. skip\_rows : If this is selected then the plugin will skip a given number of rows. The number of rows should be given in noOfRows config parameter given below.



2. `pass_in_datapoint` : If this is selected then the given number of rows will be combined into a string. This string will be present inside some given datapoint. Useful in cases where we want to ingest meta data along with readings from the csv file.
  3. `do_not_skip`: This option will not take any action on the header.
- **‘datapointForCombine’**: **type: string default: ‘metadata’**: If header method is `pass_in_datapoint` then it is the datapoint name where the given number of rows will get combined.
  - **‘noOfRows’**: **type: integer default: ‘1’**: No. of rows to skip or combine to single value. Used when header-Method is either `skip_rows` or `pass_in_datapoint`.
  - **‘variableCols’**: **type: boolean default: ‘false’**: It should be set true, when the columns in every row of CSV are not fixed. For example If you have a file like this

a,b,c

2,3,,23

4

Then you should set it true.

---

**Note:** Only one reading will be ingested at a time in this case. If you want to increase the rate then increase `readingPerSec` parameter in advanced plugin configuration.

---

- **‘columnMethod’**: **type: enumeration default: ‘pick\_from\_file’**: If variable Columns is false then it indicates how columns are considered.
  1. `pick_from_file` : The columns will be picked using a row index given.
  2. `explicit` : Specify the columns inside `useColumns` parameter.
- **‘autoGeneratePrefix’**: **type: string default: ‘column’**: If variable Columns is set true then data points will be generated using the prefix. For example if there is row like this 1,,2 and we chose `autoGeneratePrefix` to be `column`, then we will get data points like this `column_1: 1, column_3: 2`. Empty values will be ignored.
- **‘useColumns’**: **type: string default: ‘:’**: Format **column1:type,column2:type**

The data types supported are: int, float, str, datetime, bool

We can perform three tasks with this config parameter.

1. The column name will get renamed in the reading if different name is used other than present in CSV file.
2. We can select a subset of columns from total columns.
3. We can convert the data type of each column.

Example if the file is like the following

id,value,status

1,2.5,'OK'

2,2.7,'OK'

Then we can give

1. `id:int,temperature:float,status:str`

The column value will be renamed to temperature.

2. `id:int,value:float`



Only two columns will be selected here.

3. id:int,temperature:int,status:str

The data type will be converted to integer. Also column will be renamed.

Row index for column names	<input type="text" value="0"/>
Ingest mode	<input type="button" value="burst"/>
Sample rate	<input type="text" value="8000"/>
Burst interval (ms)	<input type="text" value="1000"/>
Timestamp processing mode	<input type="button" value="current time"/>
Timestamp column name	<input type="text" value=""/>
Timestamp format	<input type="text" value="%Y-%m-%d %H:%M:%S.%f%z"/>
Ignore or report for NaN	<input type="button" value="ignore"/>
Post process method	<input type="button" value="continue_playing"/>
Suffix name	<input type="text" value=".tmp"/>

- **‘rowIndexForColumnNames’**: type: integer default: **‘0’**: If column method is pick\_from\_file then it is the index from where column names are taken.
- **‘ingestMode’**: type: enumeration default: **‘burst’**: Burst or continuous mode for ingestion.
- **‘sampleRate’**: type: integer default: **‘8000’**: No of readings per second to ingest.
- **‘burstInterval’**: type: integer default: **‘1000’**: Used for burst mode. Time interval between consecutive bursts in milliseconds.
- **‘timestampStyle’**: type: enumeration default: **‘current time’**: Controls how to give timestamps to reading. Works in four ways:
  1. current time: The timestamp in the readings is whatever the local time in the machine.
  2. copy csv value: Copy the timestamp present in the CSV file.
  3. move csv value: Used when we do not want to include timestamps from files in actual readings.
  4. use csv sample delta: Pick the delta between two readings in the file and construct the timestamp of reading using this delta. Assuming the delta remains constant through out the file.)
- **‘timestampCol’**: type: string default: **‘’**: The timestamp column to pick from the file. Used only when timestampStyle is not ‘current time’.
- **‘timestampFormat’**: type: string default: **‘%Y-%m-%d %H:%M:%S.%f%z’**: The timestamp format that will be used to parse the time stamps present in the file. Used only when timestampStyle is not ‘current time’.
- **‘ignoreNaN’**: type: enumeration default: **ignore**: Pandas takes the white spaces and missing values as NaN’s. These NaN’s cause problem while ingesting into database. It is left to the user to ensure there



are no missing values in CSV file. However if the option selected is report. Then plugin will check for NaN's and report error to user. This can serve as a way to check the CSV file for missing values. However the user has to take action on what to do with NaN values. The default action is to ignore them. When error is reported the user must delete the south service and try again with clean CSV file.

- **'postProcessMethod': type: enumeration default: 'continue\_playing'**: It is the method to process the CSV file once all rows are ingested. It could be:
  1. **continue\_playing**  
Play the file again if finished.
  2. **delete**  
Delete the played file once finished.
  3. **rename**  
Rename the file with suffix after playing.
- **'suffixName': type: string default: '.tmp'**: The suffix name for renaming the file if postProcess method is rename.

### Execution

Assuming you have a csv file named vibration.csv inside FLEDGE\_ROOT/data/csv\_data (Can give a pattern like vib. The plugin will search for all the files starting with vib and therefore find out the file named vibration.csv). The csv file has fixed number of columns per row. Also assuming the column names are present in the first line. The plugin will rename the file with suffix .tmp after playing. Here is the cURL command for that.

```
res=$(curl -sX POST http://localhost:8081/fledge/service -d @- << EOF | jq
↪ '. '
{
  "name": "csv_player",
  "type": "south",
  "plugin": "csvplayback",
  "enabled": false,
  "config": {
    "assetName": {"value": "My_csv_asset"},
    "csvDirName": {"value": "FLEDGE_DATA/csv_data"},
    "csvFileName": {"value": "vib"},
    "headerMethod": {"value": "do_not_skip"},
    "variableCols": {"value": "false"},
    "columnMethod": {"value": "pick_from_file"},
    "rowIndexForColumnNames": {"value": "0"},
    "ingestMode": {"value": "burst"},
    "sampleRate": {"value": "8000"},
    "postProcessMethod": {"value": "rename"},
    "suffixName": {"value": ".tmp"}
  }
}
EOF
)

echo $res
```



## Poll Vs Async

The plugin also works in async mode. Though the default mode is poll. The async mode is faster but suffers with memory growth when sample rate is too high for the machine configuration.

Use the following sed operation for async and start the plugin again. The second sed operation, in similar way, can be used if you want to revert back to poll mode. Restart for the plugin service is required.

```
plugin_path=$FLEDGE_ROOT/python/fledge/plugins/south/csvplayback/csvplayback.py
value='s/POLL_MODE=True/POLL_MODE=False/'
sudo sed -i $value $plugin_path

# for reverting back to poll the commands will be
plugin_path=$FLEDGE_ROOT/python/fledge/plugins/south/csvplayback/csvplayback.py
value='s/POLL_MODE=False/POLL_MODE=True/'
sudo sed -i $value $plugin_path
```

## Behaviour under various modes

Table 1: Behaviour of CSV playback plugin

Plugin mode	Ingest mode	Behaviour
poll	burst	No memory growth. Resembles the way sensors give data in real life. However the timestamps of readings won't differ by a fixed delta.
poll	continuous	No memory growth. Readings differ by a constant delta. However it is slow in performance.
async	continuous	Similar to poll continuous but faster. However memory growth is observed over time.
async	burst	Similar to poll burst. Not used generally.

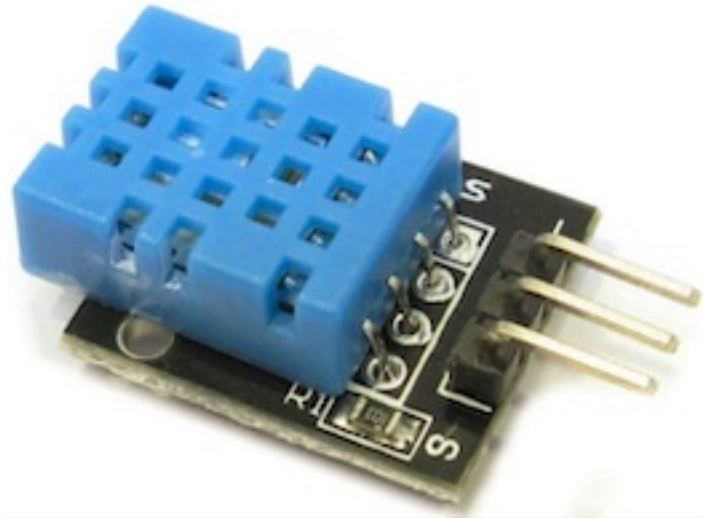
For using poll mode in continuous setting increase the readingPerSec category to the sample rate.

```
sampling_rate=8000
curl -sX PUT http://localhost:8081/fledge/category/csv_playerAdvanced -d '{
  ↪ "bufferThreshold": "'$sampling_rate'", "readingsPerSec": "'$sampling_rate'" }' | jq
```

It is advisable to increase the buffer threshold to atleast half the sample rate for good performance. (As done in above command)



### 8.1.6 DHT11 (C version)



The *fledge-south-dht* plugin implements a temperature and humidity sensor using the DHT11 sensor module. Two versions of plugins for the DHT11 are available and are used as the example for . The other DHT11 plugin is *fledge-south-dht11* and is a .

The DHT11 and the associated DHT22 sensors may be used, however they have slightly different characteristics;

	DHT11	DHT22
Voltage	3 to 5 Volts	3 to 5 Volts
Current	2.5mA	2.5mA
Humidity Range	0-50 % humidity 5% accuracy	0-100% humidity 2.5% accuracy
Temperature Range	0-50 +/- 2 degrees C	-40 to 80 +/- 0.5 degrees C
Sampling Frequency	1Hz	0.5Hz

---

**Note:** Due to the requirement for attaching to GPIO pins this plugin is only available for the Raspberry Pi platform.

---

To create a south service with the DHT11 plugin

- Click on *South* in the left hand menu bar
- Select *dht11\_V2* from the plugin list
- Name your service and click *Next*



1 Plugin & Service Name      2 Review Configuration      3 Done

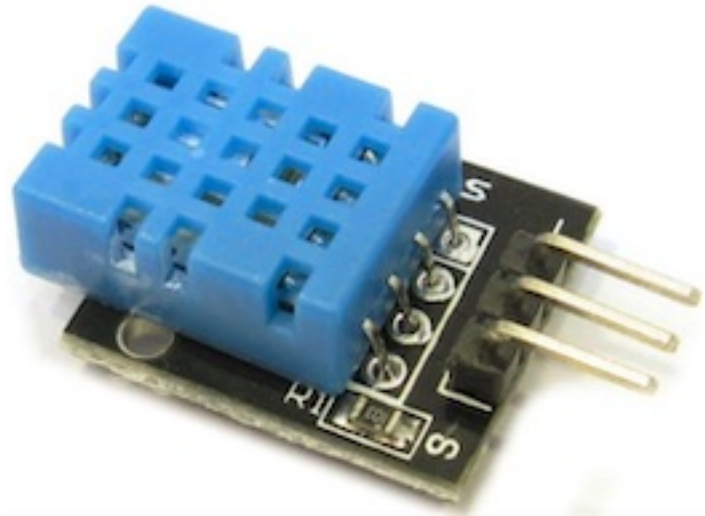
Asset Name: dht11

Rpi Pin: 7

Previous      Next

- Configure the plugin
  - **Asset Name:** The asset name which will be used for all data read.
  - **Rpi Pin:** The GPIO pin on the Raspberry Pi to which the DHT11 serial pin is connected.
- Click *Next*
- Enable the service and click on *Done*

### 8.1.7 DHT11 (Python version)



The *fledge-south-dht11* plugin implements a temperature and humidity sensor using the DHT11 sensor module. Two versions of plugins for the DHT11 are available and are used as the example for . The other DHT11 plugin is *fledge-south-dht* and is a .

The DHT11 and the associated DHT22 sensors may be used, however they have slightly different characteristics;

	DHT11	DHT22
Voltage	3 to 5 Volts	3 to 5 Volts
Current	2.5mA	2.5mA
Humidity Range	0-50 % humidity 5% accuracy	0-100% humidity 2.5% accuracy
Temperature Range	0-50 +/- 2 degrees C	-40 to 80 +/- 0.5 degrees C
Sampling Frequency	1Hz	0.5Hz



**Note:** Due to the requirement for attaching to GPIO pins this plugin is only available for the Raspberry Pi platform.

---

To create a south service with the DHT11 plugin

- Click on *South* in the left hand menu bar
- Select *dht11* from the plugin list
- Name your service and click *Next*

The screenshot shows a three-step configuration process. Step 1, 'Plugin & Service Name', is active and highlighted with a green circle. Step 2, 'Review Configuration', is also highlighted with a green circle. Step 3, 'Done', is marked with a grey circle. The configuration form contains two fields: 'Asset Name' with the value 'dht11' and 'GPIO Pin' with the value '4'. At the bottom, there are 'Previous' and 'Next' buttons.

- Configure the plugin
  - **Asset Name:** The asset name which will be used for all data read.
  - **GPIO Pin:** The GPIO pin on the Raspberry Pi to which the DHT11 serial pin is connected.
- Click *Next*
- Enable the service and click on *Done*

### 8.1.8 DNP3 Master Plugin

The *fledge-south-dnp3* allows Fledge to act as a DNP3 master and gather data from a DNP3 Out Station. The plugin will fetch all data types from the DNP3 Out Station and create assets for each in Fledge. The DNP3 plugin also handles unsolicited messages transmitted by the outstation.



1 Plugin & Service Name      2 Review Configuration      3 Done

**Asset Name prefix**

**Master link Id**

**Outstation address**

**Outstation port**

**Outstation link Id**

**Data scan** ☐

**Scan interval**

**Network timeout**

[Previous](#) [Next](#)

- **Asset Name prefix:** An asset name prefix that is prepended to the DNP3 objects retrieved from the DNP3 outstations to create the Fledge asset name.
- **Master link id:** The master link id Fledge uses when implementing the DNP3 protocol.
- **Outstation address:** The IP address of the DNP3 Out Station to be connected.
- **Outstation port:** The port on the Out Station to which the connection is established.
- **Outstation link Id:** The Out Station link id.
- **Data scan:** Enable or disable the scanning of all objects and values in the Out Station. This is the Integrity Poll for all Classes.
- **Scan interval:** The interval between data scans of the Out Station.
- **Network timeout:** Timeout for fetching data from the Out Station expressed in seconds.

## DNP3 Out Station Testing

The `opendnp3` package contains a demo Out Station that can be used for test purposes. After building the `opendnp3` package on your machine run the demo program as follows;

```
$ cd opendnp3/build
$ ./outstation-demo
```

This demo application listens on any IP address, port 20001 and has link Id set to 10. It also assumes master link Id is 1. Configuring your Fledge plugin with these parameters should allow Fledge to connect to this Out Station.

Once started it logs traffic and waits for use input to send unsolicited messages:



```
Enter one or more measurement changes then press <enter>
c = counter, b = binary, d = doublebit, a = analog, o = octet string, 'quit' = exit
```

Another option is the use of a DNP3 Out Station simulator, as an example:

<http://freyrscada.com/dnp3-ieee-1815-Client-Simulator.php#Download-DNP3-Development-Bundle>

Once the bundle has been downloaded, the **DNPOutstationSimulator.exe** application under the “Simulator” folder can be installed and run on a Windows 32bit platform.

### 8.1.9 Expression South Plugin

The *fledge-south-expression* plugin is a plugin that is used to generate synthetic data using a mathematical expression to generate data that changes over time. The user may configure the plugin with an expression of their choice and define a period in terms of samples per period of the output and the increment between each sample.

The screenshot shows a configuration window for the Expression South Plugin. At the top, a progress bar indicates three steps: 1. Plugin & Service Name, 2. Review Configuration (the current step, highlighted with a green circle), and 3. Done. The main configuration area is a light gray box containing five labeled text input fields: 'Asset Name' (empty), 'Expression' (containing the formula `clamp(-1.0, sin(2 * pi * x) + cos(x / 2 * pi), +1.0)`), 'Minimum Value' (containing `-5`), 'Maximum Value' (containing `5`), and 'Step Value' (containing `0.001`). At the bottom of the window, there are two buttons: 'Previous' (disabled, light gray) and 'Next' (active, blue).

The parameters that can be configured are;

- **Asset Name:** The name of the asset to be created inside Fledge.
- **Expression:** The expression that should be evaluated to create the asset value, see below.
- **Minimum Value:** The minimum value of  $x$ , where  $x$  is the value that sweeps over time.
- **Maximum Value:** The maximum value of  $x$ , where  $x$  is the value that sweeps over time.
- **Step Value:** The step in  $x$  for each call to the expression evaluation.



## Expression Support

The *fledge-south-expression* plugin makes use of the library to do run time expression evaluation. This library provides a rich mathematical operator set, the most useful of these in the context of this plugin are;

- Mathematical operators (+, -, \*, /, %, ^)
- Functions (min, max, avg, sum, abs, ceil, floor, round, roundn, exp, log, log10, logn, pow, root, sqrt, clamp, inrange, swap)
- Trigonometry (sin, cos, tan, acos, asin, atan, atan2, cosh, cot, csc, sec, sinh, tanh, d2r, r2d, d2g, g2d, hyp)

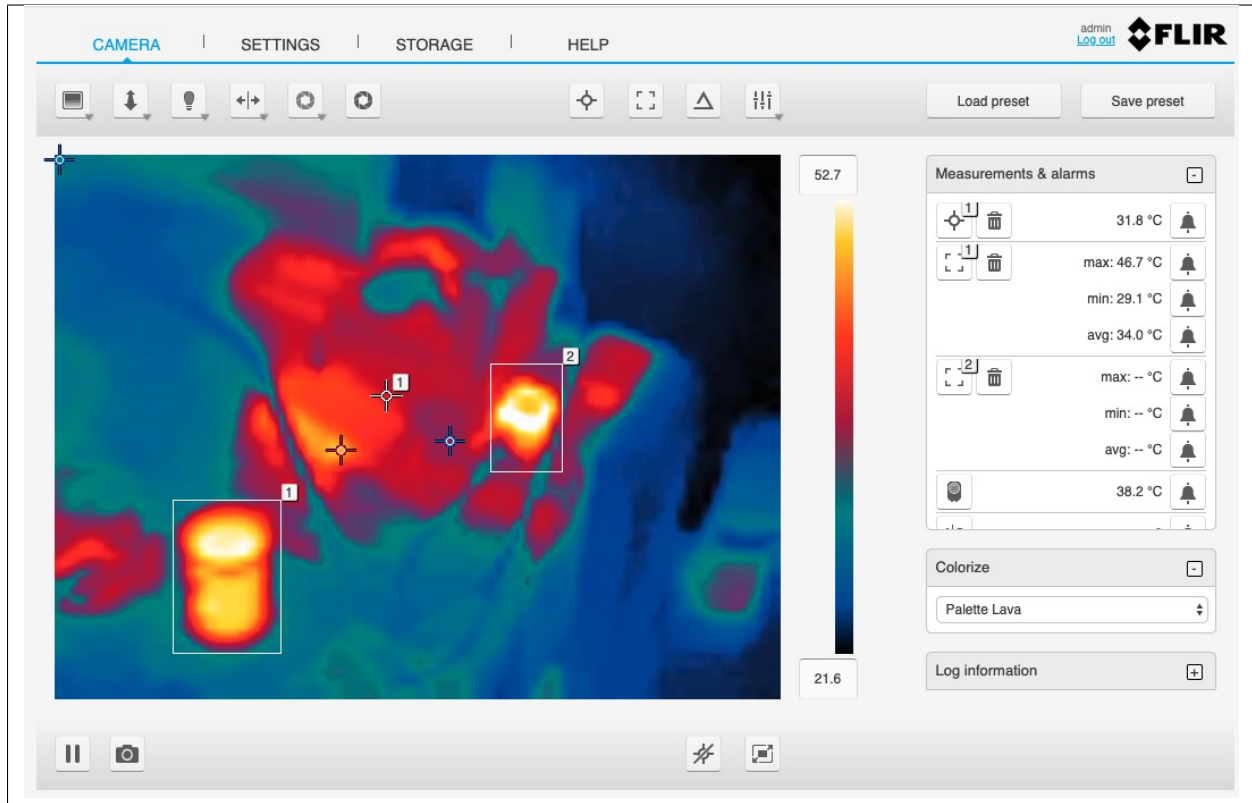
### 8.1.10 Flir AX8 Thermal Imaging Camera



The *fledge-south-FlirAX8* plugin is a south plugin that enables temperature data to be collected from Flir Thermal Imaging Devices, in particular the AX8 and other A Series cameras. The camera provides a number of temperatures for both spots and boxes defined within the field of view of the camera. In addition it can also provide deltas between two temperature readings.

The bounding boxes and spots to read are configured by connecting to the web interface of the camera and dropping the spots on a thermal imaging or pulling out rectangles for the bounding boxes. The camera will return a minimum, maximum and average temperature within each bounding box.





In order to configure a south service to obtain temperature data from a Flir camera select the *South* option from the left-hand menu bar and click on the Add icon in the top right corner of the South page that appears. Select the *FlirAX8* plugin from the list of south plugins, name your service and click on *Next*.

The screen that appears is the configuration screen for the *FlirAX8* plugin.

1 Plugin & Service Name 2 Review Configuration 3 Done

**Asset Name** AX8

**Server Address** 192.168.0.48

**Port** 502

**Slave ID** 1

Previous Next



There are four configuration parameters that can be set, usually it is only necessary to change the first two however;

- **Asset Name:** This is the asset name that the temperature data will be written to Fledge using. A single asset is used that will contain all of the values read from the camera.
- **Server Address:** This is the address of the Modbus server within the camera. This is the same IP address that is used to connect to the user interface of the camera.
- **Port:** The TCP port on which the cameras listens for Modbus requests. Unless changed in the camera the default port of 502 should be used.
- **Slave ID:** The Modbus Slave ID of the camera. By default the cameras are supplied with this set to 1, if changed within your camera setup you must also change the value here to match.

Once entered click on *Next*, enable the service on the next page and click on *Done*.

This will create a single asset that contains values for all boxes and spots that may be define. A filter *fledge-filter-FlirValidity* can be added to the south service to remove data for boxes and spots not switched on in the camera user interface. See . This filter also allows you to name the boxes and hence have more meaningful names in the data points within the asset.

### 8.1.11 South HTTP

The *fledge-south-http* plugin allows data to be received from another Fledge instance or external system using a REST interface. The Fledge which is sending the data to the corresponding north task with the HTTP north plugin installed. There are two options for the HTTP north or , these serve the dual purpose of providing a data path between Fledge instances and also as examples of how other systems might use the REST interface from C/C++ or Python. The plugin supports both HTTP and HTTPS transport protocols and sends a JSON payload of reading data in the internal Fledge format.

The primary purpose of this plugin is for Fledge to Fledge communication however, there is no reason to prevent other applications that wish to send data into a Fledge system to not use this plugin also. The only requirement is that the application that is sending the data uses the same JSON payload structure as Fledge uses for passing reading data between different instances. Data should be sent to the URL defined in the configuration of the plugin using a POST request. The caller may choose to send one or many readings within a single POST request and those readings may be for multiple assets.

To create a south service you, as with any other south plugin

- Select *South* from the left hand menu bar.
- Click on the + icon in the top left
- Choose *http\_south* from the plugin selection list
- Name your service
- Click on *Next*
- Configure the plugin



1 Plugin & Service Name      2 Review Configuration      3 Done

Host: 0.0.0.0

Port: 6683

URI: sensor-reading

Asset Name Prefix: http-

Enable HTTP: ☒

HTTPS Port: 6684

Certificate Name: fledge

Previous      Next

- **Host:** The host name or IP address to bind to. This may be left as default, in which case the plugin binds to any address. If you have a machine with multiple network interfaces you may use this parameter to select one of those interfaces to use.
- **Port:** The port to listen for connection from another Fledge instance.
- **URL:** URI that the plugin accepts data on. This should normally be left to the default.
- **Asset Name Prefix:** A prefix to add to the incoming asset names. This may be left blank if you wish to preserve the same asset names.
- **Enable HTTP:** This toggle specifies if HTTP connections should be accepted or not. If the toggle is off then only HTTPS connections can be used.
- **Certificate Name:** The name of the certificate to use for the HTTPS encryption. This should be the name of a certificate that is stored in the Fledge .

- Click *Next*
- Enable your service and click *Done*

## JSON Payload

The payload that is expected by this plugin is a simple JSON presentation of a set of reading values. A JSON array is expected with one or more reading objects contained within it. Each reading object consists of a timestamp, an asset name and a set of data points within that asset. The data points are represented as name value pair JSON properties within the reading property.

The fixed part of every reading contains the following

Name	Description
times-tamp	The timestamp as an ASCII string in ISO 8601 extended format. If no time zone information is given it is assumed to indicate the use of UTC.
asset	The name of the asset this reading represents.
read-ings	A JSON object that contains the data points for this asset.



The content of the *readings* object is a set of JSON properties, each of which represents a data value. The type of these values may be integer, floating point, string, a JSON object or an array of floating point numbers.

A property

```
"voltage" : 239.4
```

would represent a numeric data value for the item *voltage* within the asset. Whereas

```
"voltageUnit" : "volts"
```

Is string data for that same asset. Other data may be presented as arrays

```
"acceleration" : [ 0.4, 0.8, 1.0 ]
```

would represent acceleration with the three components of the vector, x, y, and z. This may also be represented as an object

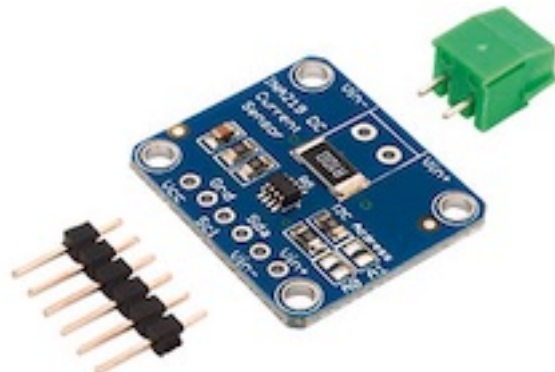
```
"acceleration" : { "X" : 0.4, "Y" : 0.8, "Z" : 1.0 }
```

both are valid formats within Fledge.

An example payload with a single reading would be as shown below

```
[
  {
    "timestamp" : "2020-07-08 16:16:07.263657+00:00",
    "asset"      : "motor1",
    "readings"   : {
      "voltage"  : 239.4,
      "current"  : 1003,
      "rpm"      : 120147
    }
  }
]
```

### 8.1.12 INA219 Voltage & Current Sensor



The *fledge-south-ina219* plugin is a south plugin that uses an INA219 breakout board to measure current and voltage. The Texas Instruments INA219 is capable of measuring voltages up to 26 volts and currents up to 3.2 Amps. It connects via the I2C bus of the host and multiple sensors may be daisy chain on a single I2C bus. Breakout boards that mount the chip and its associate shunt resistor and connectors and easily available and attached to hosts with I2C buses.



The INA219 support three voltage/current ranges

- 32 Volts, 2 Amps
- 32 Volts, 1 Amp
- 16 Volts, 400 mAmps

Choosing the smallest range that is sufficient for your application will give you the best accuracy.

---

**Note:** This plugin is only available for the Raspberry Pi as it requires to be interfaced to the I2C bus on the Raspberry Pi GPIO header socket.

---

To create a south service with the INA219

- Click on *South* in the left hand menu bar
- Select *ina219* from the plugin list
- Name your service and click *Next*

The screenshot shows a configuration window for the 'ina219' plugin. At the top, there is a progress bar with three steps: 1. Plugin & Service Name, 2. Review Configuration (currently active), and 3. Done. Below the progress bar, there are three input fields: 'Asset Name' with the value 'electrical', 'I2C Address' with the value '64', and 'Voltage Range' with the value '32V2A'. At the bottom of the window, there are two buttons: 'Previous' and 'Next'.

- Configure the plugin
  - **Asset Name:** The asset name of the asst that will be written
  - **I2C Address:** The address of the INA219 device
  - **Voltage Range:** The voltage range that is to be used. This may be one of 32V2A, 32V1A or 16V400mA
- Click *Next*
- Enable the service and click on *Done*



## Wiring The Sensor

The INA219 uses the I2C bus on the Raspberry PI, which requires two wires to connect the bus, it also requires power taking the total to four wires

INA219 Pin	Raspberry Pi Pin
Vin	3V3 pin 1
GND	GND pin 9
SDA	SDA pin 3
SCL	SCL pin 5

### 8.1.13 Lathe Simulation

The *fledge-south-lathesim* plugin is a south plugin that simulates a lathe with a number of attached sensors. The purpose of this plugin is for test and demonstration only as it does not attach to any real device.

The plugin simulates four sensor devices attached to the virtual lathe

- The PLC controlling the lathe that gives details such as cutting depth, tool position, motor speed
- A current sensor that measures the current draw from the lathe
- A vibration sensor giving the RMS value of the vibration and the dominant vibration frequency
- A thermal imaging device that takes temperature readings every second from the motor, gearbox, headstock, tailstock and tool on the lathe

The vibration sensor reports at half the rate of the other sensors attached to the lathe in order to simulate handling data that is related to the same physical device but not available at the same rate as the other sensors.

The simulation runs a repeated pattern of operations;

- A spin-up period where the lathe spins up to speed from idle.
- A period where the lathe is doing some cutting of a work piece.
- A spin-down period where the lathe is slowing to a stop.
- An idle period where the work piece is removed and replaced with a new billet.

During the spin up period the lathe speed, expressed in revolutions per minute, will linearly increase from 0 to the maximum defined.

When the lathe is cutting the speed will remain predominantly constant, with a small random variation, whilst the depth of cut and X position of the cutting tool will change.

The lathe then spins down to rest and will remain idle for a short time whilst the worked item is removed and a new billet of material is installed.

During the cutting period the current draw and vibration will alter as load is applied to the piece.



## Configuring the PLC

There are a number of configuration options that can be applied to the simulation.

Lathe1 South Service

Lathe Name: lathe

Spin up time: 5

Cutting time: 45

Idle time: 15

Spin down time: 6

RPM: 500

Current: 750

Enabled: ☒

[Show Advanced Config](#)

Applications +

Cancel Save

- **Lathe Name:** The name of the lathe in this configuration. This name is used to derive the assets returned from the three sets of sensors. The PLC data is returned with an asset name that machines the lathe name. The current data has *Current* appended to the lathe name and the asset id of the vibration name is the lathe name with *Vibration* appended to it. The temperature data uses the asset with the name of the lathe and *IR* appended to it.
- **Spin up time:** The time in seconds it takes the lathe to spin up to working speed from idle.
- **Cutting time:** The time in seconds for which the lathe is cutting material.
- **Spin Down time:** The time in seconds for which the lathe is spinning down from operating speed to stop.
- **Idle time:** The time in seconds for which the lathe is idle between jobs.
- **RPM:** The operating speed of the lathe, expressed in revolutions per minute.
- **Current:** The nominal operating current draw of the lathe.

### 8.1.14 Modbus South Plugin

The *fledge-south-modbus-c* plugin is a south plugin that supports both TCP and RTU variants of Modbus. The plugin provides support for reading Modbus coils, input bits, registers and input registers, a flexible mechanism is provided to create a mapping between the Modbus registers and coils and the assets within Fledge. Multiple registers can be combined to allow larger values than register width to be mapped from devices that represent data in this way. Support is also included for floating point representation within the Modbus registers.



## Configuration Parameters

A Modbus south service is added in the same way as any other south service in Fledge,

- Select the *South* menu item
- Click on the + icon in the top right
- Select *ModbusC* from the plugin list
- Enter a name for your Modbus service
- Click *Next*
- You will be presented with the following configuration page



Asset Name

modbus

Protocol

RTU

Server Address

127.0.0.1

Port

2222

Device

Baud Rate

9600

Number Of Data Bits

8

Number Of Stop Bits

1

Parity

none

Slave ID

1

Register Map

1

{

2

"values": [

3

{

4

"name": "temperature",

5

"slave": 1,

6

"assetName": "Booth1",

7

"register": 0,

8

"scale": 0.1,

9

"offset": 0

10

},

11

{

12

"name": "humidity",

}

}

Timeout

0.5

Control

None

Control Map

1

{

2

"values": [ ]

3

}

- **Asset Name:** This is the name of the asset that will be used for the data read by this service. You can override this within the Modbus Map, so this should be treated as the default if no override is given.



- **Protocol:** This allows you to select either the *RTU* or *TCP* protocol. Modbus RTU is used whenever you have a serial connection, such as RS485 for connecting to your device. The TCP variant is used where you have a network connection to your device.
- **Server Address:** This is the network address of your Modbus device and is only valid if you selected the *TCP* protocol.
- **Port:** This is the port to use to connect to your Modbus device if you are using the TCP protocol.
- **Device:** This is the device to open if you are using the RTU protocol. This would be the name of a Linux device in `/dev`, for example `/dev/SERIAL0`
- **Baud Rate:** The baud rate used to communicate if you are using a serial connection with Modbus RTU.
- **Number of Data Bits:** The number of data bits to send on serial connections.
- **Number of Stop Bits:** The number of stop bits to send on the serial connections.
- **Parity:** The parity setting to use on the serial connection.
- **Slave ID:** The slave ID of the Modbus device from which you wish to pull data.
- **Register Map:** The register map defines which Modbus registers and coils you read, and how to map them to Fledge assets. The map is a complex JSON object which is described in more detail below.
- **Timeout:** The request timeout when communicating with a Modbus TCP client. This can be used to increase the timeout when a slow Modbus device or network is used.
- **Control:** Which register map should be used for mapping control entities to modbus registers.



If no control is required then this may be set to *None*. Setting this to *Use Register Map* will cause all the registers that are being read to also be targets for control. Setting this to *Use Control Map* will cause the separate *Control Map* to be used to map the control set points to modbus registers.

- **Control Map:** The register map that is used to map the set point names into Modbus registers for the purpose of set point control. The control map is the same JSON format document as the register map and uses the same set of properties.



## Register Map

The register map is the most complex configuration parameter for this plugin and over time has supported a number of different variants. We will only document the latest of these here although previous variants are still supported. This latest variant is the most flexible to date and is thus the recommended approach to adopt.

The map is a JSON object with a single array *values*, each element of this array is a JSON object that defines a single item of data that will be stored in Fledge. These objects support a number of properties and values, these are

Property	Description
name	The name of the value that we are reading. This becomes the name of the data point with the asset. This may be either the default asset name defined plugin or an individual asset if an override is given.
slave	The Modbus slave ID of the device if it differs from the global Slave ID defined for the plugin. If not given the default Slave ID will be used.
asset-Name	This is an optional property that allows the asset name define for the plugin to be overridden on an individual basis. Multiple values in the values array may share the same AssetName, in which case the values read from the Modbus device are placed in the same asset. Note: This is unused in a control map.
register	This defines the Modbus register that is read. It may be a single register, in which case the value is the register number or it may be multiple registers in which case the value is a JSON array of numbers. If an array is given then the registers are read in the order of that array and combined into a single value by shifting each value up 16 bits and performing a logical OR operation with the next register in the array.
coil	This defines the number of the Modbus coil to read. Coils are single bit Modbus values.
input	This defines the number of the Modbus discrete input. Coils are single bit Modbus values.
inputRegister	This defines the Modbus input register that is read. It may be a single register, in which case the value is the register number or it may be multiple registers in which case the value is a JSON array of numbers. If an array is given then the registers are read in the order of that array and combined into a single value by shifting each value up 16 bits and performing a logical OR operation with the next register in the array.
scale	A scale factor to apply to the data that is read. The value read is multiplied by this scale. This is an optional property.
offset	An optional offset to add to the value read from the Modbus device.
type	This allows data to be cast to a different type. The only support type currently is <i>float</i> and is used to interpret data read from the one or more of the 16 bit registers as a floating point value. This property is optional.
swap	This is an optional property used to byte swap values read from a Modbus device. It may be set to one of <i>bytes</i> , <i>words</i> or <i>both</i> to control the swapping to apply to bytes in a 16 bit value, 16 bit words in a 32 bit value or both bytes and words in 32 bit values.

Every *value* object in the *values* array must have one and only one of *coil*, *input*, *register* or *inputRegister* included as this defines the source of the data in your Modbus device. These are the Modbus object types and each has an address space within a typical Modbus device.

Object Type	Size	Address Space	Map Property
Coil	1 bit	00001 - 09999	coil
Discrete Input	1 bit	10001 - 19999	input
Input Register	16 bits	30001 - 39999	inputRegister
Holding Register	16 bits	40001 - 49999	register

The values in the map for coils, inputs and registers are relative to the base of the address space for that object type rather than the global address space and each is 0 based. A map value that has the property *"coil" : 10* would return



the values of the tenth coil and “*register*” : 10 would return the tenth register.

## Example Maps

In this example we will assume we have a cooling fan that has a Modbus interface and we want to extract three data items of interest. These items are

- Current temperature that is in Modbus holding register 10
- Current speed of the fan that is stored as a 32 bit value in Modbus holding registers 11 and 12
- The active state of the fan that is stored in a Modbus coil 1

The Modbus Map for this example would be as follow:

```
{
  "values" : [
    {
      "name"      : "temperature",
      "register"   : 10
    },
    {
      "name"      : "speed",
      "register"   : [ 11, 12 ]
    },
    {
      "name"      : "active",
      "coil"      : 1
    }
  ]
}
```

Since none of these values have an `assetName` defined all there values will be stored in a single asset, the name of which is the default asset name defined for the plugin as a whole. This asset will have three data points within it; *temperature*, *speed* and *active*.

## Function Codes

The *fledge-south-modbus-c* plugin attempts to make as few calls as possible to the underlying modbus device in order to collect the data. This is done in order to minimise the load that is placed on the modbus server. The modbus function codes used to read each coil or register type are as follows;

Object Type	Function Code	Size	Address Space	Map Property
Coil	01 Read Coils	1 bit	00001 - 09999	coil
Discrete Input	02 Read Discrete inputs	1 bit	10001 - 19999	input
Input Register	04 Read register	16 bits	30001 - 39999	inputRegister
Holding Register	16 Read multiple registers	16 bits	40001 - 49999	register



## Set Point Control

The *fledge-south-modbus-c* plugin supports the Fledge set point control mechanisms and allows a register map to be defined that maps the set point attributes to the underlying modbus registers. As an example a control map as follows

```
{
  "values" : [
    {
      "name" : "active",
      "coil" : 1
    }
  ]
}
```

Defines that a set point write operation can be instigated against the set point named *active* and this will map to the Modbus coil 1.

Set points may be defined for Modbus coils and registers, the read only input bits and input registers can not be used for set point control.

The *Control Map* can use the same swapping, scaling and offset properties as modbus *Register Map*, it can also map multiple registers to a single set point and floating point values.

## Error Messages

The following are messages that may be produced by the *fledge-south-modbus-c* plugin, these messages are written to the system log file and may be viewed by the *System* menu item in the Fledge user interface. This display may be filtered on the name of a particular south service in order to view just the messages that originate from that south service.

**The value of slave in the modbus map should be an integer** When a modbus slave identifier is defined within the JSON modbus map it should always be given as a integer value and should not be enclosed in quotes

```
"slave" : 0
```

**The value of slave for item 'X' in the modbus map should be an integer** A name entity in the modbus map is defined as a string and must be enclosed in double quotes. This error would indicate that a non-string value has been given.

```
"name" : "speed"
```

**Each item in the modbus map must have a name property** Each of the modbus entities that is read must define a name property for the entity.

```
"name" : "speed"
```

**The value of assetName for item 'X' in the modbus map should be a string** The optional property *assetName* must always be provided as a string in the modbus map.

```
"assetName" : "pumpSpeed"
```

**The value of scale for item 'X' in the modbus map should be a floating point number** The optional property *scale* must always be expressed as a numeric value in the JSON of the modbus map, and should not be enclosed in quotes.

```
"scale" : 1.4
```



**The value of offset for item ‘X’ in the modbus map should be a floating point number** The optional property *offset* must always be given as a numeric value in the JSON definition of the modbus item, and should not be enclosed in quotes.

```
"offset" : 2.0
```

**The value of coil for item ‘X’ in the modbus map should be a number** The coil number given in the modbus map of an item must be an integer number, and should not be enclosed in quotes.

```
"coil" : 22
```

**The value of input for item ‘X’ in the modbus map must be either an integer** The input number given in the modbus map of an item must be an integer number, and should not be enclosed in quotes.

```
"input" : 22
```

**The value of register for item ‘X’ in the modbus map must be either an integer or an array** The register to read for an entity must be either an integer number or in the case of values constructed from multiple registers it may be an array of integer numbers. Numeric values should not be enclosed on quotes.

```
"register" : 22
```

Or, if two registers are being combined

```
"register" : [ 18, 19 ]
```

**The register array for item ‘X’ in the modbus map contain integer values** When giving an array as the value of the register property for a modbus item, that array must only contain register numbers expressed as numeric values. Register numbers should not be enclosed in quotes.

```
"register" : [ 18, 19 ]
```

**The value of inputRegister for item ‘X’ in the modbus map must be either an integer or an array** The input register to read for an entity must be either an integer number or in the case of values constructed from multiple input registers it may be an array of integer numbers. Numeric values should not be enclosed on quotes.

```
"inputRegister" : 22
```

Or, if two input registers are being combined

```
"inputRegister" : [ 18, 19 ]
```

**The type property of the item ‘X’ in the modbus map must be a string** The optional *type* property for a modbus entity must be expressed as a string enclosed in double quotes.

```
"type" : "float"
```

**The type property ‘Y’ of the item ‘X’ in the modbus map is not supported** The *type* property of the item is not supported by the plugin. Only the type *float* is currently supported.

**The swap property ‘Y’ of item ‘X’ in the modbus map must be one of bytes, words or both** An unsupported option has been supplied as the value of the swap property, only *bytes*, *words* or *both* are supported values.

**The swap property of the item ‘X’ in the modbus map must be a string** The optional *swap* property of a modbus item must be given as a string in double quotes and must be one of the supported swap options.

```
"swap" : "bytes"
```



**Item ‘X’ in the modbus map must have one of coil, input, register or inputRegister properties** Each modbus item to be read from the modbus server must define how that item is addressed. This is done by adding a modbus property called *coil*, *input*, *register* or *inputRegister*.

**Item ‘X’ in the modbus map must only have one of coil, input, register or inputRegister properties** Each modbus item to be read from the modbus server must define how that item is addressed. This is done by adding a modbus property called *coil*, *input*, *register* or *inputRegister*, these are mutually exclusive and only one of them may be given per item in the modbus map.

**N errors encountered in the modbus map** A number of errors have been detected in the modbus map. These must be correct in order for the plugin to function correctly.

**Parse error in modbus map, the map must be a valid JSON object.** The modbus map JSON document has failed to parse. An additional text will be given that describes the error that has caused the parsing of the map to fail.

**Parse error in control modbus map, the map must be a valid JSON object.** The modbus control map JSON document has failed to parse. An additional text will be given that describes the error that has caused the parsing of the map to fail.

**Failed to connect to Modbus device** The plugin has failed to connect to a modbus device. In the case of a TCP modbus connection this could be because the address or port have been misconfigured or the modbus device is not currently reachable on the network. In the case of a modbus RTU device this may be a misconfiguration or a permissions issue on the entry in `/dev` for the device. Additional information will be given in the error message to help identify the issue.

### 8.1.15 South MQTT

The *fledge-south-mqtt-readings* plugin allows to create an MQTT subscriber service. MQTT Subscriber reads messages from topics on the MQTT broker.

To create a south service you, as with any other south plugin

- Select *South* from the left hand menu bar
- Click on the + icon in the top right
- Choose mqtt-readings from the plugin selection list
- Name your service
- Click on *Next*
- Configure the plugin

The screenshot shows a configuration window for the MQTT plugin. At the top, there is a progress bar with three steps: 1. Plugin & Service Name, 2. Review Configuration (currently active), and 3. Done. The configuration fields are as follows:

Field	Value
MQTT Broker host	localhost
MQTT Broker Port	1883
Keep Alive Interval	60
Topic To Subscribe	Room1/conditions
QoS Level	0
Asset Name	mqtt-

At the bottom of the window, there are two buttons: "Previous" and "Next".



- **MQTT Broker host:** Hostname or IP address of the broker to connect to.
  - **MQTT Broker Port:** The network port of the broker.
  - **Keep Alive Interval:** Maximum period in seconds allowed between communications with the broker. If no other messages are being exchanged, this controls the rate at which the client will send ping messages to the broker.
  - **Topic To Subscribe:** The subscription topic to subscribe to receive messages.
  - **QoS Level:** The desired quality of service level for the subscription.
  - **Asset Name:** Name of Asset.
- Click *Next*
  - Enable your service and click *Done*

## Message Payload

The content of the message payload published to the topic, to which the service is configured to subscribe, should be parsable to a JSON object.

e.g. `{“humidity”: 93.29, “temp”: 16.82}`

```
$ mosquitto_pub -h localhost -t "Room1/conditions" -m '{"humidity": 93.29, "temp": 16.82}'
```

The `mosquitto_pub` client utility comes with the `mosquitto` package, and a great tool for conducting quick tests and troubleshooting. [https://mosquitto.org/man/mosquitto\\_pub-1.html](https://mosquitto.org/man/mosquitto_pub-1.html)

### 8.1.16 MQTT Sparkplug B

The `fledge-south-mqtt-sparkplug` plugin implements the Sparkplug B payload format with an MQTT (Message Queue Telemetry Transport) transport. The plugin will subscribe to a configured topic and will process the Sparkplug B payloads, creating Fledge assets from those payloads. Sparkplug is an open source software specification of a payload format and set of conventions for transporting sensor data using MQTT as the transport mechanism.

---

**Note:** Sparkplug is bi-directional, however this plugin will only read data from the Sparkplug device.

---

To create a south service with the MQTT Sparkplug B plugin

- Click on *South* in the left hand menu bar
- Select `mqtt_sparkplug` from the plugin list
- Name your service and click *Next*



1 Plugin & Service Name      2 Review Configuration      3 Done

Asset Name: mqtt

MQTT Host: chariot.groov.com

MQTT Port: 1883

Username: opto

Password: .....

Topic: spBv1.0/Opto22/DDATA/groovEPIC\_workshop/Strategy

Previous      Next

- Configure the plugin
  - **Asset Name:** The asset name which will be used for all data read.
  - **MQTT Host:** The MQTT host to connect to, this is the host that is running the MQTT broker.
  - **MQTT Port:** The MQTT port, this is the port the MQTT broker uses for unencrypted traffic, usually 1883 unless modified.
  - **Username:** The user name to be used when authenticating with the MQTT subsystem.
  - **Password:** The password to use when authenticating with the MQTT subsystem.
  - **Topic:** The MQTT topic to which the plugin will subscribe.
- Click *Next*
- Enable the service and click on *Done*

### 8.1.17 OPC/UA South Plugin

The *fledge-south-opcua* plugin allows Fledge to connect to an OPC/UA server and subscribe to changes in the objects within the OPC/UA server.

A south service to collect OPC/UA data is created in the same way as any other south service in Fledge.

- Use the *South* option in the left hand menu bar to display a list of your South services
- Click on the + add icon at the top right of the page
- Select the *opcua* plugin from the list of plugins you are provided with
- Enter a name for your south service
- Click on *Next* to configure the OPC/UA plugin



1 Plugin & Service Name 2 Review Configuration 3 Done

Asset Name

OPCUA Server URL

OPCUA Object Subscriptions

```

1 {
2   "subscriptions": [
3     "ns=5;s=85/0:Simulation"
4   ]
5 }

```

Subscribe By ID ☒

Asset Name Source

Asset Path Delimiter

Min Reporting Interval

Previous Next

The configuration parameters that can be set on this page are;

- **Asset Name:** This is a prefix that will be applied to all assets that are created by this plugin. The OPC/UA plugin creates a separate asset for each data item read from the OPC/UA server. This is done since the OPC/UA server will deliver changes to individual data items only. Combining these into a complex asset would result in assets that contain only one of many data points in each update. This can cause problems in upstream systems with the ever-changing asset structure.
- **OPCUA Server URL:** This is the URL of the OPC/UA server from which data will be extracted. The URL should be of the form `opc.tcp://.../`
- **OPCUA Object Subscriptions:** The subscriptions are a set of locations in the OPC/UA object hierarchy that defined which data is subscribed to in the server and hence what assets get created within Fledge. A fuller description of how to configure subscriptions is shown below.
- **Subscribe By ID:** This toggle determines if the OPC/UA objects in the subscription are using names to identify the objects in the OPC/UA object hierarchy or using object ID's.
- **Asset Name Source:** This drop-down allows you to choose the source of the name for the Asset in Fledge. The choices are:
  - *NodeId*: the Node Id of the OPC UA node. This is the default.
  - *BrowseName*: the Browse Name of the OPC UA node.
  - *Subscription Path with NodeId*: the path to the node in the OPC/UA server's object hierarchy starting with the node specified in *Subscriptions*. Every node in the hierarchy is named with its Node Id. The Node Id is also used as the Data Point name.



- *Subscription Path with BrowseName*: same as *Subscription Path with NodeId* except that the Browse Name is used as the name of every node in the hierarchy. The Browse Name is also used as the Data Point name.
- *Full Path with NodeId*: the path to the node in the OPC/UA server's object hierarchy starting with the top-level *Objects* folder. Every node in the hierarchy is named with its Node Id. The *Objects* folder itself is not part of the full path. The Node Id is also used as the Data Point name.
- *Full Path with BrowseName*: same as *Full Path with NodeId* except that the Browse Name is used as the name of every node in the hierarchy. The Browse Name is also used as the Data Point name.
- **Asset Path Delimiter**: A character to separate segments of the Asset Path. The delimiter is a single character. If multiple characters are specified, the string will be truncated to one character. The default is the forward slash (“/”).
- **Min Reporting Interval**: This controls the minimum interval between reports of data changes in subscriptions. It sets an upper limit to the rate that data will be ingested into the plugin and is expressed in milliseconds.

## Subscriptions

Subscriptions to OPC/UA objects are stored as a JSON object that contains an array named “subscriptions”. This array is a set of OPC/UA nodes that will control the subscription to variables in the OPC/UA server.

The array may be empty, in which case all variables are subscribed to in the server and will create assets in Fledge. Note that simply subscribing to everything will return a lot of data that may not be of use.

If the *Subscribe By ID* option is set then this is an array of Node Id's. Each Node Id should be of the form *ns=...;s=...*. Where *ns* is a namespace index and *s* is the Node Id string identifier. A subscription will be created with the OPC/UA server for the object with the specified Node Id and its children, resulting in data change messages from the server for those objects. Each data change received from the server will create an asset in Fledge with the name of the object prepended by the value set for *Asset Name*. An integer identifier is also supported by using a Node Id of the form *ns=...;i=...*.

If the *Subscribe By ID* option is not set then the array is an array of Browse Names. The format of the Browse Names is *<namespace>:<name>*. If the namespace is not required then the name can simply be given, in which case any name that matches in any namespace will have a subscription created. The plugin will traverse the node tree of the server from the *ObjectNodes* root and subscribe to all variables that live below the named nodes in the subscriptions array.

## Configuration examples

```
{ "subscriptions": [ "5:Simulation", "2:MyLevel" ] }
```

We subscribe to

- 5:Simulation is a node name under ObjectsNode in namespace 5
- 2:MyLevel is a variable under ObjectsNode in namespace 2

```
{ "subscriptions": [ "5:Sinusoid1", "2:MyLevel", "5:Sawtooth1" ] }
```

We subscribe to

- 5:Sinusoid1 and 5:Sawtooth1 are variables under ObjectsNode/Simulation in namespace 5
- 2:MyLevel is a variable under ObjectsNode in namespace 2

```
{ "subscriptions": [ "2:Random.Double", "2:Random.Boolean" ] }
```

We subscribe to



- Random.Double and Random.Boolean are variables under ObjectsNode/Demo both in namespace 2

It's also possible to specify an empty subscription array:

```
{"subscriptions":[]}
```

**Note:** Depending on OPC/UA server configuration (number of objects, number of variables) this empty configuration might take a long time to create the subscriptions and hence delay the startup of the south service. It will also result in a large number of assets being created within Fledge.

Object names, variable names and NamespaceIndexes can be easily retrieved browsing the given OPC/UA server using OPC UA clients, such as .

### 8.1.18 Person Detection Plugin

The *fledge-south-person-detection* detects a person on a live video feed from either a camera or on a network stream. It uses Google's Mobilenet SSD v2 to detect a person. The bounding boxes and confidence scores are displayed on the same video frame itself. Also FPS (frames per second) are also displayed on the same frame. The detection results are also converted into readings. The readings have mainly three things:

1. *Count* : The number of people detected.
2. *Coordinates* : It consists of coordinates (x,y) of top-left and bottom right corners of bounding box for each detected person.
3. *Confidence* : Confidence with which the model detected each person.

The screenshot shows a configuration window titled "rtrr South Service". It contains the following settings:

- TFlite Model File:** detect.tflite
- Labels File:** coco\_labels.txt
- Asset Name:** person\_detection\_4
- Enable Edge TPU:** ☐
- Minimum Confidence Threshold:** 0.5
- Source For Detection Camera/Stream:** stream
- Stream URL:** rtsp://192.168.0.109:8554/clip
- Opencv Backend:** ffmpeg
- Stream Protocol:** udp
- Camera ID:** 0
- Enable Detection Window:** ☐

- **TFlite Model File:** This is the name of the tflite model file that should be placed in python/fledge/plugins/south/person\_detection/model directory. Its default value is detect\_edgetpu.tflite. If a Coral Edge TPU is not being used, the file name will be different (i.e. detect.tflite).
- **Labels File:** This is the name of the labels file that was used when training the above model, this file should also be placed in same directory as the model.



- **Asset Name:** The name of the asset used for the readings generated by this plugin.
- **Enable Edge TPU:** Indicates whether to use edge TPU for inference. If you don't want to use Coral Edge TPU then disable this configuration parameter. Also ensure to change the name of the model file to detect.tflite if disabled. Default is set to enabled.
- **Minimum Confidence Threshold:** The detection results from the model will be filtered out, if the score is below this value.
- **Source:** Either use a stream over a network or use a local camera device. Default is set to stream.
- **Streaming URL:** The URL of the RTSP stream, if stream is to be used. Only RTSP streams are supported for now.
- **OpenCV Backend:** The backend required by OpenCV to process the stream, if stream is to be used. Default is set to ffmpeg.
- **Streaming Protocol:** The protocol over which live frames are being transported over the network, if stream is to be used. Default is set to udp.
- **Camera ID:** The number associated with your video device. See /dev in your filesystem you will see video0 or video1. It is required when source is set to camera. Default is set to 0.
- **Enable Detection Window:** Show detection results in a native window. Default is set to disabled.

rtrr South Service

Stream Protocol

udp

Camera ID

0

Enable Detection Window

☐

Enable Web Streaming

☒

Web Streaming Port

8085

Enabled

☐

[Show Advanced Config](#)

Applications

Cancel

Save

?

Export Readings

Delete Service

- **Enable Web Streaming:** Whether to stream the detected results in a browser or not. Default is set to enabled.
- **Web Streaming Port:** Port number where web streaming server should run, if web streaming is enabled. Default is set to 8085.



## Installation

### 1. First run requirements.sh

There are two ways to get the video feed.

#### 1. Camera

To see the supported configuration of the camera run the following command.

```
$ v4l2-ctl --list-formats-ext --device /dev/video0
You will see something like
'YUYV' (YUYV 4:2:2)
  Size: Discrete 640x480
    Interval: Discrete 0.033s (30.000 fps)
  Size: Discrete 720x480
    Interval: Discrete 0.033s (30.000 fps)
  Size: Discrete 1280x720
    Interval: Discrete 0.033s (30.000 fps)
  Size: Discrete 1920x1080
    Interval: Discrete 0.067s (15.000 fps)
    Interval: Discrete 0.033s (30.000 fps)
  Size: Discrete 2592x1944
    Interval: Discrete 0.067s (15.000 fps)
  Size: Discrete 0x0
```

Above example uses Camera ID 0 to indicate use of /dev/video0 device, please use the applicable value for your setup

#### 2. Network RTSP stream

To create a network stream follow the following steps

##### 1. Install vlc

```
$ sudo add-apt-repository ppa:videolan/master-daily
$ sudo apt update
$ apt show vlc
$ sudo apt install vlc qtwayland5
$ sudo apt install libavcodec-extra
```

##### 2. Download some sample files from here.

```
$ git clone https://github.com/intel-iot-devkit/sample-videos.git
```

##### 3. Either stream a file using the following

```
$ vlc <name_of_file>.mp4 --sout '#gather:transcode{vcodec=h264,vb=512,
↪scale=Auto,width=640,height=480,acodec=none,scodec=none}:rtp{sdp=rtsp://
↪<ip_of_machine_streaming>:8554/clip}' --no-sout-all --sout-keep --loop --
↪no-sout-audio --sout-x264-profile=baseline
```

Note : fill the <ip\_of\_the\_machine> with ip of the machine which will be used to stream video. Also fill <name\_of\_file> with the name of mp4 file.

##### 4. You can also stream from a camera using the following

```
$ vlc v4l2:///dev/video<index_of_video_device> --sout '#gather:transcode
↪{vcodec=h264,vb=512,scale=Auto,width=<supported_width_of_camera_image>,
↪height=<supported_height_of_camera_image>,acodec=none,scodec=none}:rtp
↪{sdp=rtsp://<ip_of_the_machine>:8554/clip}' --no-sout-all --sout-keep --
↪no-sout-audio --sout-x264-profile=baseline
```

(continues on next page)



(continued from previous page)

Fill the following :

<index\_of\_video\_device> The index with which you ran the v4l2 command mentioned above. for example video0.

<supported\_height\_of\_camera\_image> Height you get when you ran v4l2 command mentioned above. For example Discrete 640x480. Here 480 is height.

<supported\_width\_of\_camera\_image> Width you get when you ran v4l2 command mentioned above. For example Discrete 640x480. Here 640 is width.

<ip\_of\_the\_machine> ip of the machine which will be used to stream video.

Once you have run the plugin by filling appropriate parameters Now go to your browser and enter *ip\_where\_fledge\_is\_running:the\_port\_for\_web\_streaming*

### 8.1.19 Playback Plugin

The *fledge-south-playback* plugin is a feature rich plugin for playing back comma separated variable (CSV) files. It supports features such as;

- Header rows
- User defined column names
- Use of historic or current timestamps
- Multiple timestamp formats
- Pick and optionally rename columns
- Looped or single pass readings of the data

To create a south service with the playback plugin

- Click on *South* in the left hand menu bar
- Select *playback* from the plugin list
- Name your service and click *Next*



1 Plugin & Service Name      2 Review Configuration      3 Done

Asset Name

CSV file name with extension

Header Row ☒

Header columns

Cherry pick column with same/new name

1	{}
---	----

Historic timestamps ☐

Pick timestamp delta from file ☐

Timestamp column name

Timestamp format

Ingest mode

Sample Rate

Burst Interval (ms)

Burst size

Read file in a loop ☐

- Configure the plugin
  - **Asset Name:** An asset name to use for the content of the file.
  - **CSV file name with extension:** The name of the file that is to be processed, the file must be located in the fledge data directory.
  - **Header Row:** Toggle to indicate the first row is a header row that contains the names that should be used for the data points within the asset.
  - **Header Columns:** Only used if *Header Row* is not enabled. This parameter should a column separated list of column names that will be used to name the data points within the asset.
  - **Cherry pick column with same/new name:** This is a JSON document that can define a set of columns to include and optionally names to give those columns. If left empty then all columns, are included.
  - **Historic timestamps:** A toggle field to control if the timestamp data should be the current time or a date and time taken from the file itself.
  - **Pick timestamp delta from file:** If current timestamps are used then this option can be used to maintain the same relative times between successive timestamps added to the data as it is ingested.
  - **Timestamp column name:** The name of the column that should be used for reading timestamp value. This must be given if either historic timestamps are used or the interval between readings



is to be maintained.

- **Timestamp format:** The format of the timestamp within the file.
  - **Ingest mode:** Determine if ingest should be in batch or burst mode. In burst mode data is ingested as a set of bursts of rows, defined by *Burst size*, every *Burst Interval*, this allows simulation of sensors that have internal buffering within them. Batch mode is the normal, regular rate ingest of data.
  - **Sample Rate:** The data sampling rate that should be used, this is defined in readings per second.
  - **Burst Interval (ms):** The time interval between consecutive bursts when burst mode is used.
  - **Burst size:** The number of readings to be sent in each burst.
  - **Read file in a loop:** Once the end of the file is reached then the plugin will go back to the start and resend the data if this toggle is on.
- Click *Next*
  - Enable the service and click on *Done*

### Picking Columns

The *Cherry pick column with same/new name* entry is a JSON document with a set of key/value pairs. The key is the name of the column in the file and the value is the name which should appear in the final asset. To illustrate this let's assume we have a CSV file as follows

```
X, Y, Z, Amps, Volts
1.3, 0.1, 0.3, 2.1, 240
1.2, 0.3, 0.2, 2.2, 235
....
```

We want to create an asset that has the *X* and *Y* values, *Amps* and *Volts* but we want to name them *X*, *Y*, *Current*, *Voltage*. We can do this by creating a JSON document that maps the columns.

```
{
  "X" : "X",
  "Y" : "Y",
  "Amps" : "Current",
  "Volts" : "Voltage"
}
```

Since we only mention the columns *X*, *Y*, *Amps* and *Volts*, only these will be included in the asset and we will not include the column *Z*. We map the column name *X* to *X*, so it will be unchanged. As will the column *Y*, the column *Amps* will become the data point *Current* and *Volts* will become *Voltage*.



### 8.1.20 PT100 Temperature Sensor



The *fledge-south-pt100* is a south plugin for the PT-100 temperature sensor. The PT100 is a resistance temperature detectors (RTDs) consist of a fine wire (typically platinum) wrapped around a ceramic core, exhibiting a linear increase in resistance as temperature rises. The sensor connects via a MAX31865 converter to a GPIO pins for I2C bus and a chip select pin.

**Note:** This plugin is only available for the Raspberry Pi as it requires to be interfaced to the I2C bus on the Raspberry Pi GPIO header socket.

To create a south service with the PT100

- Click on *South* in the left hand menu bar
- Select *pt100* from the plugin list
- Name your service and click *Next*

A screenshot of the Fledge configuration interface for the PT100 plugin. The interface shows a progress bar with three steps: 1. Plugin & Service Name, 2. Review Configuration, and 3. Done. The current step is 2, Review Configuration. Below the progress bar, there are two input fields: 'Asset Name Prefix' with the value 'PT100/' and 'GPIO Pin' with the value '8'. At the bottom, there are two buttons: 'Previous' and 'Next'.

- Configure the plugin
  - **Asset Name Prefix:** A prefix to add to the asset name
  - **GPIO Pin:** The GPIO pin on the Raspberry PI to which the MAX31865 chip select is connected.
- Click *Next*
- Enable the service and click on *Done*

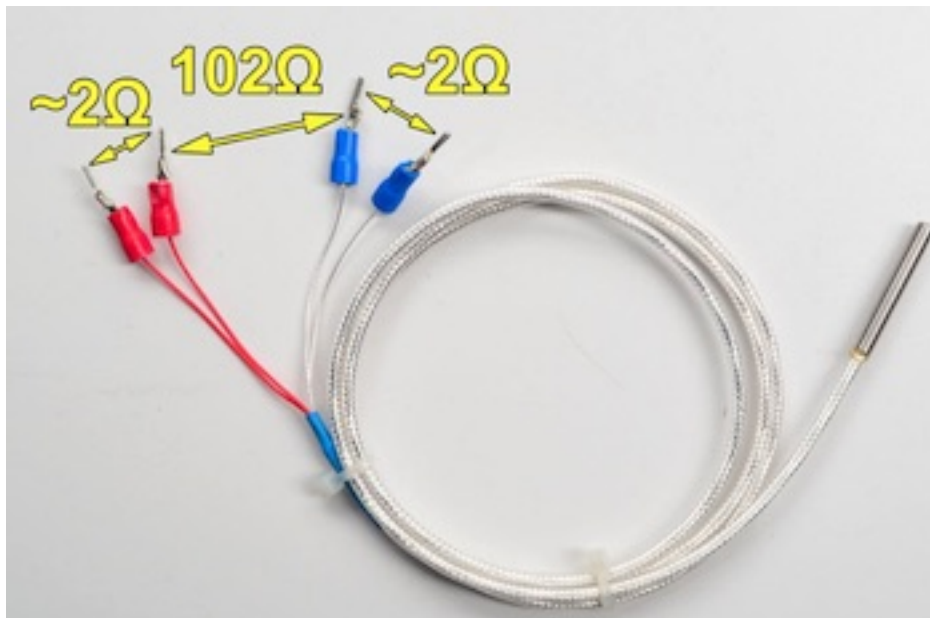


### Wiring The Sensor

The MAX31865 uses the I2C bus on the Raspberry PI, which requires three wires to connect the bus, it also requires a chip select pin to be wired to a general GPIO pin and power.

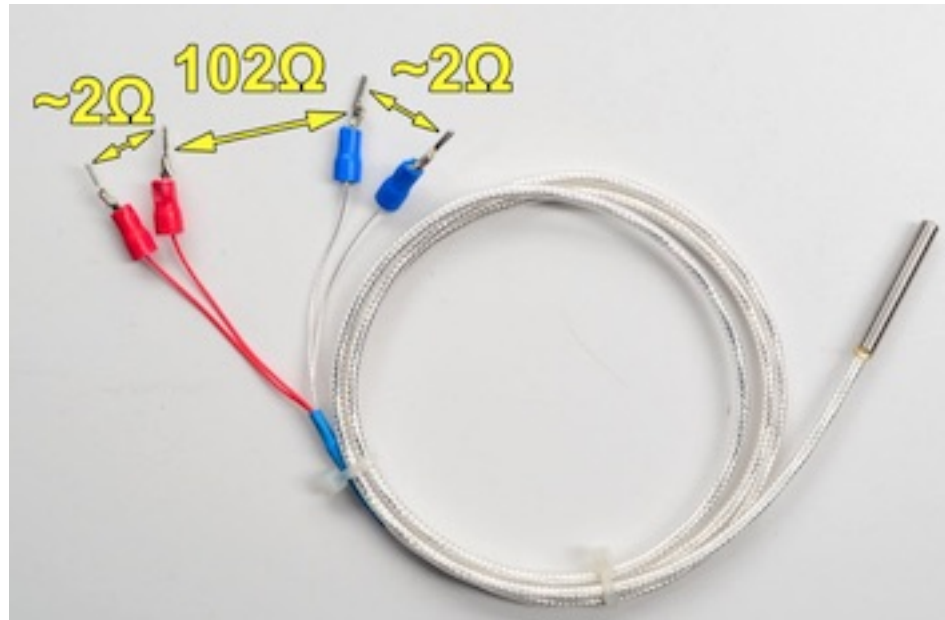
MAX 31865 Pin	Raspberry Pi Pin
Vin	3V3
GND	GND
SDI	MOSI
SDO	MISO
CLK	SCLK
CS	GPIO (default GPIO8)

There are two options for connecting a PT100 to the MAX31865, a three wire PT100 or a four wire PT100.



To connect a four wire PT100 to the MAX 31865 the wires are connected in pairs, the two red wires are connected to the RTD- connector pair on the MAX31865 and the two remaining wires are connected to the RTD+ connector pair. If your PT100 does not have red wires or you wish to verify the colours are correct use a multimeter to measure the resistance across the pair of wires. Each pair should show 2 ohms between them and the difference between the two pairs should be 102 ohms, but will vary with temperature.





To connect a three wire sensor connect the red pair of wires across the RTD+ pair of connectors and the third wire on the RTD- block. If your PT100 does not have a pair of red wires, or you wish to verify the colours and have access to a multimeter, the resistance between the red wires should be 2 ohms. ~The resistance to the third wire, from the red pair, will be approximately 102 ohms but will vary with temperature.

If using the 3 wire sensor you must also modify the jumpers on the MAX31865.



Create a solder bridge across the 2/3 Wire jumper, outlined in red in the picture above.



You must also cut the thin wire trace on the jumper block outlined in yellow that runs between the 2 and 4.

Then create a new connection between the 4 and 3 side of this jumper block. This is probably best done with a solder bridge.

### 8.1.21 Random

The *fledge-south-random* plugin is a plugin that will create random data.

To create a south service with the Random plugin

- Click on *South* in the left hand menu bar
- Select *Random* from the plugin list
- Name your service and click *Next*

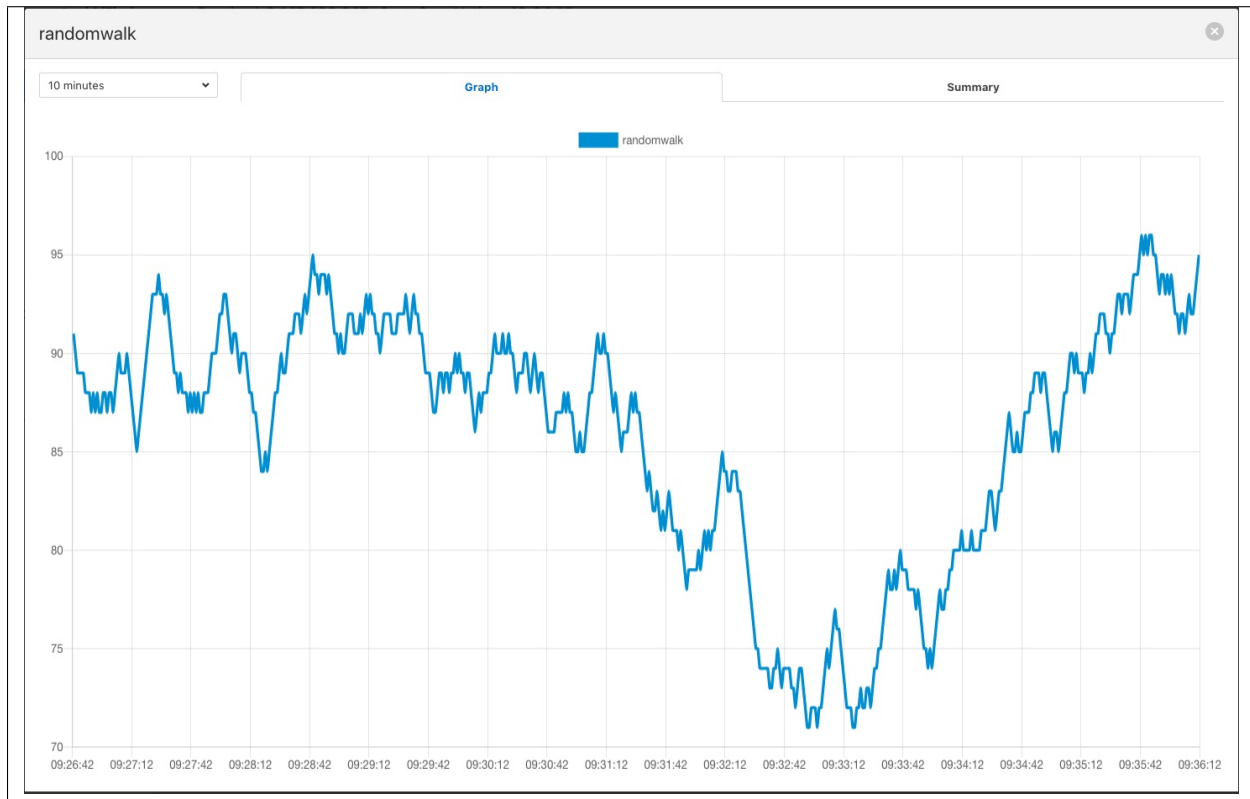
The screenshot shows a three-step configuration wizard. Step 1, 'Plugin & Service Name', is active and highlighted with a green circle and line. It contains a text input field labeled 'Asset Name' with the value 'Random' entered. Step 2, 'Review Configuration', is indicated by a green circle and line. Step 3, 'Done', is indicated by a grey circle and line. At the bottom, there are 'Previous' and 'Next' buttons. The 'Next' button is blue and highlighted.

- Configure the plugin
  - **Asset name:** The name of the asset that will be created
- Click *Next*
- Enable the service and click on *Done*



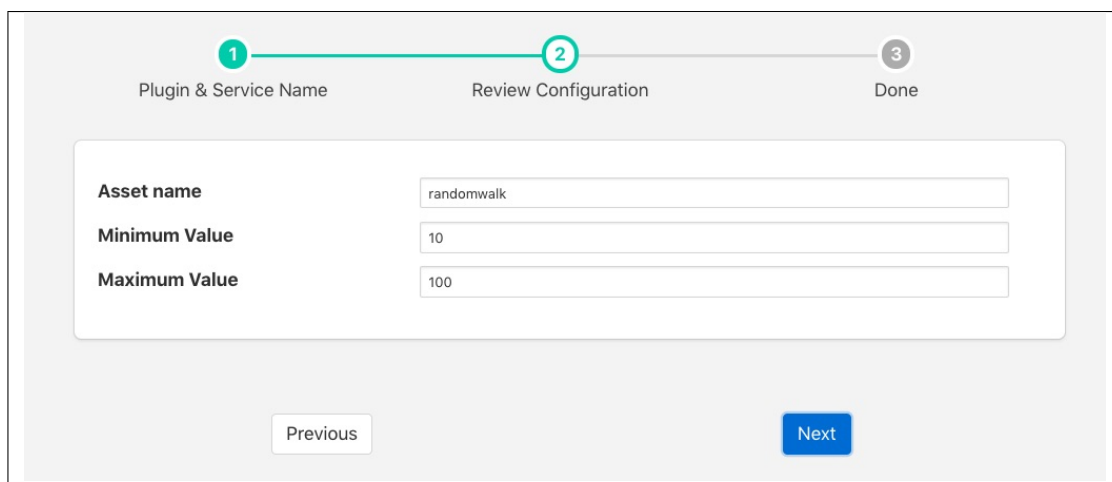
## 8.1.22 Random Walk

The *fledge-south-randomwalk* plugin is a plugin that will create random data between a pair of values. Each new value is based on a random increment or decrement of the previous. This results in an output that appears as follows



To create a south service with the Random Walk plugin

- Click on *South* in the left hand menu bar
- Select *randomwalk* from the plugin list
- Name your service and click *Next*



1 Plugin & Service Name 2 Review Configuration 3 Done

Asset name randomwalk

Minimum Value 10

Maximum Value 100

Previous Next



- Configure the plugin
  - **Asset name:** The name of the asset that will be created
  - **Minimum Value:** The minimum value to include in the output
  - **Maximum Value:** The maximum value to include in the output
- Click *Next*
- Enable the service and click on *Done*

### 8.1.23 Enviro pHAT Plugin



The *fledge-south-rpienviro* is a plugin that uses the Pimoroni Enviro pHAT sensor board. The Enviro pHAT board is an environmental sensing board populated with multiple sensors, the plugin pulls data from the;

- RGB light sensor
- Magnetometer
- Accelerometer
- Temperature/pressure Sensor

Individual sensors can be enabled or disabled separately in the configuration. Separate assets are created for each sensor within Fledge with individual controls over the naming of these assets.

---

**Note:** The Enviro pHAT plugin is only available on the Raspberry Pi as it is specific the GPIO pins of that device.

---

To create a south service with the Enviro pHAT



- Click on *South* in the left hand menu bar
- Select *rpienviro* from the plugin list
- Name your service and click *Next*

The screenshot displays the 'Review Configuration' step of the Fledge plugin setup. A progress bar at the top indicates the current step is 2 of 3. The configuration form includes the following settings:

Field	Value
Asset Name Prefix	e_
RGB Sensor	<input checked="" type="checkbox"/>
RGB Sensor Name	rgb
Magnetometer Sensor	<input checked="" type="checkbox"/>
Magnetometer Sensor Name	magnetometer
Accelerometer Sensor	<input checked="" type="checkbox"/>
Accelerometer Sensor Name	accelerometer
Weather Sensor	<input checked="" type="checkbox"/>
Weather Sensor Name	weather

Navigation buttons at the bottom are 'Previous' and 'Next'.

- Configure the plugin
  - **Asset Name Prefix:** An optional prefix to add to the asset names. The asset names created by the plugin are; rgb, magnetometer, accelerometer and weather. Using the prefix you can add an identifier to the front of each such that it becomes easier to differentiate between multiple instances of the sensor.
  - **RGB Sensor:** A toggle control to turn on or off collection of RGB light level information
  - **RGB Sensor Name:** Set a name for the RGB sensor asset
  - **Magnetometer Sensor:** A toggle control to turn on or off collection of magnetometer data
  - **Magnetometer Sensor Name:** Set a name for the magnetometer sensor asset
  - **Accelerometer Sensor:** A toggle to turn on or off collection of accelerometer data
  - **Accelerometer Sensor Name:** Set a name for the accelerometer sensor asset
  - **Weather Sensor:** A toggle to turn on or off collection of weather data
  - **Weather Sensor Name:** Set a name for the weather sensor asset
- Click *Next*
- Enable the service and click on *Done*



### 8.1.24 OPC/UA Safe & Secure South Plugin

The *fledge-south-s2opcua* plugin allows Fledge to connect to an OPC/UA server and subscribe to changes in the objects within the OPC/UA server. This plugin is very similar to the *fledge-south-opcua* plugin but is implemented using a different underlying OPC/UA open source library, from Systemrel. The major difference between the two is the ability of this plugin to support secure endpoints with the OPC/UA server.

A south service to collect OPC/UA data is created in the same way as any other south service in Fledge.

- Use the *South* option in the left hand menu bar to display a list of your South services
- Click on the + add icon at the top right of the page
- Select the *s2opcua* plugin from the list of plugins you are provided with
- Enter a name for your south service
- Click on *Next* to configure the OPC/UA plugin



1

2

3

Plugin & Service NameReview ConfigurationDone

**Asset Name**

s2opcua

**OPCUA Server URL**

opc.tcp://localhost:53530/OPCUA/SimulationServer

**OPCUA Object Subscriptions**

1 ▾

{

2 ▾

"subscriptions": [

3

"ns=3;i=1001",

4

"ns=3;i=1002"

5

]

6

}

**Min Reporting Interval (millisec)**

1000

**Security Mode**

None ▾

**Security Policy**


None ▾

**User Authentication Policy**

anonymous ▾

**Username**

**Password**

password 

**CA Certificate Authority**

**Server Public Certificate**

**Client Public Certificate**

**Client Private Key**

**Certificate Revocation List**

**Debug Trace File**

☐

Previous

Next



The configuration parameters that can be set on this page are;

- **Asset Name:** This is a prefix that will be applied to all assets that are created by this plugin. The OPC/UA plugin creates a separate asset for each data item read from the OPC/UA server. This is done since the OPC/UA server will deliver changes to individual data items only. Combining these into a complex asset would result in assets that do only contain one of many data points in each update. This can cause upstream systems problems with the every changing asset structure.
- **OPCUA Server URL:** This is the URL of the OPC/UA server from which data will be extracted. The URL should be of the form `opc.tcp://.../`
- **OPCUA Object Subscriptions:** The subscriptions are a set of locations in the OPC/UA object hierarchy that defined which data is subscribed to in the server and hence what assets get created within Fledge. A fuller description of how to configure subscriptions is shown below.
- **Min Reporting Interval:** This control the minimum interval between reports of data changes in subscriptions. It sets an upper limit to the rate that data will be ingested into the plugin and is expressed in milliseconds.

The screenshot shows a configuration panel with three labels on the left: "Security mode", "Security policy", and "User authentication policy". On the right, there is a rounded rectangle containing three options: "None" (selected with a checkmark), "Sign", and "SignAndEncrypt". Below this rectangle is a dropdown menu currently showing "anonymous" with a downward arrow.

- **Security Mode:** Specify the OPC/UA security mode that will be used to communicate with the OPC/UA server.

The screenshot shows a configuration panel with two labels on the left: "Security policy" and "User authentication policy". On the right, there is a rounded rectangle containing three options: "None" (selected with a checkmark), "Basic256", and "Basic256Sha256".

- **Security Policy:** Specify the OPC/UA security policy that will be used to communicate with the OPC/UA server.

The screenshot shows a configuration panel with two labels on the left: "User authentication policy" and "Username". On the right, there is a rounded rectangle containing two options: "anonymous" (selected with a checkmark) and "username".

- **User Authentication Policy:** Specify the user authentication policy that will be used when authenticating the connection to the OPC/UA server.
- **Username:** Specify the username to use for authentication. This is only used if the *User authentication policy* is set to *username*.
- **Password:** Specify the password to use for authentication. This is only used if the *User authentication policy* is set to *username*.



- **CA Certificate Authority:** The name of the root certificate authorities certificate file in DER format. This is the certificate authority that forms the root of trust and signs the certificates that will be trusted. If using self-signed certificates this should be left blank.
- **Server Public Certificate:** The name of the public certificate of the OPC/UA server specified in the *OPCUA Server URL*. This must be a DER format certificate file. It must be signed by the certificate authority unless you are using self-signed certificates.
- **Client Public Certificate:** The name of the public certificate of the OPC/UA client application, that is, this plugin. This must be a DER format certificate file. It must be signed by the certificate authority unless you are using self-signed certificates.
- **Client Private Key:** The name of the private key of the client application, that is, the private key the plugin will use. This must be a PEM format key file.
- **Certificate Revocation List:** The name of the certificate authority's Certificate Revocation List. This is a DER format certificate. If using self-signed certificates this should be left blank.
- **Debug Trace File:** Enable the S2OPCUA OPCUA Toolkit trace file for debugging. If enabled, log files will appear in the directory */usr/local/fledge/data/logs*.

## Subscriptions

Subscriptions to OPC/UA objects are stored as a JSON object that contains an array named “subscriptions.” This array is a set of OPC/UA nodes that will control the subscription to variables in the OPC/UA server. Each element in the array is an OPC/UA node id, if that node is the id of a variable then that single variable will be added to the subscription list. If the node id is not a visible, then the plugin will recurse down the object tree below that node and add every variable it finds in this tree to the subscription list.

A subscription list which gives the root node of the OPC/UA server will cause all variables within the server to be added to the subscription list. Care however should be taken as this may be a large number of assets.

## Subscription examples

```
{ "subscriptions": [ "5:Simulation", "2:MyLevel" ] }
```

We subscribe to

- 5:Simulation is a node name under ObjectsNode in namespace 5
- 2:MyLevel is a variable under ObjectsNode in namespace 2

```
{ "subscriptions": [ "5:Sinusoid1", "2:MyLevel", "5:Sawtooth1" ] }
```

We subscribe to

- 5:Sinusoid1 and 5:Sawtooth1 are variables under ObjectsNode/Simulation in namespace 5
- 2:MyLevel is a variable under ObjectsNode in namespace 2

```
{ "subscriptions": [ "2:Random.Double", "2:Random.Boolean" ] }
```

We subscribe to

- Random.Double and Random.Boolean are variables under ObjectsNode/Demo both in namespace 2

Object names, variable names and namespace indices can be easily retrieved browsing the given OPC/UA server using OPC UA clients, such as .



### Certificate Management

OPC UA clients and servers use X509 certificates to confirm each other's identities and to enable digital signing and data encryption. Certificates are often issued by a Certificate Authority (CA) which means either the client or the server could reach out to the CA to confirm the validity of the certificate if it choses to.

The configuration described above uses the names of certificates that will be used by the plugin. These certificates must be loaded into the Fledge Certificate Store manually and named to match the names used in the configuration before the plugin is started. When entering certificate and key file names, do not include directory names or file extensions (*.der* or *.pem*).

Typically the Certificate Authorities certificate is retrieved and uploaded to the Fledge Certificate Store along with the certificate from the OPC/UA server that has been signed by that Certificate Authority. A public/private key pair must also be created for the plugin and signed by the Certificate Authority. These are uploaded to the Fledge Certificate Store.

[OpenSSL](#) may be used to generate and convert the keys and certificates required. An to do this is available as part of the underlying library.

### Certificate Requirements

Certificates must be X509 Version 3 certificates and must have the following field values:

Certificate Field	Value
Version	V3
Subject	This field must include a Common Name (CN=) which is a human-readable name such as <i>S2OPCUA South Plugin</i> . Do not use your device hostname.
Subject Alternative Name	URI= fledge:south:s2opcua, DNS= <i>deviceHostname</i>
Key Usage	Digital Signature, Key Encipherment, Non Repudiation, Data Encipherment
Extended Key Usage	Client Authentication

### Self-Signed Certificates

A common configuration is to use self-signed certificates which are issued by your own systems and cannot be validated against a CA. For this to work, the OPC UA client and server must each have a copy of the other's certificate in their Trusted Certificate stores. This task must be done by a system manager who is creating the device configuration. By copying certificates, the system manager is confirming that the client and server can legitimately communicate with each other.

### Creating a Self-Signed Certificate

There is a very useful online tool for creating self-signed certificates called [CertificateTools](#). You can watch a demonstration of CertificateTools on [YouTube](#). This section will walk you through the necessary steps to create a self-signed certificate for the S2OPCUA South plugin which is the OPC UA Client.

The [CertificateTools](#) main page is divided into sections. You can leave many of the sections at their default values. Here are the required entries for each section:



## Private Key

Leave the default values as-is: *Generate PKCS#8 RSA Private Key* and *2048 Bit*. Leave *Encrypt* unchecked.

## Subject Attributes

In *Common Names*, enter a human-readable name such as *S2OPCUA South Plugin*. Click *Add*.

Edit *Country*, *State*, *Locality* and *Organization* as you wish. We recommend:

- Country: US
- State: CA
- Locality: Menlo Park
- Organization: Dianomic

## Subject Alternative Name

Set the drop-down to *DNS*. Enter the hostname of your Fledge device. This can be an unqualified name, that is, the device hostname without domain name. Click *Add*.

Set the drop-down to *URI*. Enter *fledge:south:s2opcua*. Click *Add*.

## x509v3 Extensions

### Key Usage

Click the check boxes to enable *Critical*, *Digital Signature*, *Key Encipherment*, *Non Repudiation* and *Data Encipherment*.

### Extended Key Usage

Click the check boxes to enable *Critical* and *TLS Web Client Authentication*.

### Encoding Options

Leave at Default.

### CSR Options

Leave the first drop-down at *SHA256*. Change the second drop-down from *CSR Only* to *Self-Sign*. Doing this will expose drop-downs to set the self-signed certificate expiration time.



### Generating the Certificate and Private Key

Click *Submit*. This will create a new section marked by a blue bar labelled *Certificate 0*.

Open *Certificate 0*. This will reveal a subsection called *Download*. You will need only two of these files:

- PEM Certificate (filename *cert.crt*)
- PKCS#12 Certificate and Key (filename *cert.pfx*)

When you click the *PKCS#12 Certificate and Key* link, you will be prompted for a password for the private key. It is acceptable to click *Cancel* to proceed without a password. Download these two files to a working directory on any computer with OpenSSL installed (you will need OpenSSL to post-process the downloaded files). You do not need to do this on your Fledge device. You must do this on a machine that can run the Fledge GUI in a browser; you will need the browser to import the certificate and key into the Fledge Certificate Store.

---

**Note:** The CertificateTools webpage can show you the equivalent OpenSSL commands to perform the self-signed certificate and key generation. Look for *OpenSSL Commands* below the blue *Certificate 0* bar.

---

### Post-Processing the Certificate and Private Key

Use the OpenSSL command-line utility to convert the certificate and key files to the formats needed for the S2OPCUA South Plugin.

#### Converting the Certificate File

The *PEM Certificate* file (*cert.crt*) is in PEM format. It must be converted to DER format. The command is:

```
openssl x509 -inform pem -outform der -in cert.crt -out myclientcert.der
```

#### Converting the Private Key File

The *PKCS#12 Certificate and Key* file (*cert.pfx*) is in Public-Key Cryptography Standards [PKCS#12](#) format. It must be converted to PEM format. The command is:

```
openssl pkcs12 -in cert.pfx -out myclientkey.pem -nodes
```

This command will prompt for the Import Password. If you created a password when you downloaded the PKCS#12 Certificate and Key file, enter it now. If you did not create a password, hit Enter.

#### Importing the Certificate and Key Files

Launch the Fledge GUI. Navigate to the Certificate Store. In the upper right corner of the screen, click *Import*.



Upload Certificate

Key

Choose File

myclientkey.pem

Certificate

Choose File

myclientcert.der

☐ Overwrite  
 Overwrite allows to replace the existing key or certificate

Import

In the *Key* section, click *Choose File* and navigate to the location of the key file *myclientkey.pem*.

In the *Certificate* section, click *Choose File* and navigate to the location of the certificate file *myclientcert.der*.

Click *Import*.

You should use the Certificate Store in the Fledge GUI to import your OPC UA server certificate. In this case, enter the server certificate file name in the *Certificate* portion of the Import dialog and then click *Import*.

### 8.1.25 SenseHAT



The *fledge-south-sensehat* is a plugin that uses the Raspberry Pi Sense HAT sensor board. The Sense HAT has an 8×8 RGB LED matrix, a five-button joystick and includes the following sensors:

- Gyroscope
- Accelerometer
- Magnetometer
- Temperature



- Barometric pressure
- Humidity

In addition it has an 8x8 matrix for RGB LED's, these are not included in the devices the plugin supports.

Individual sensors can be enabled or disabled separately in the configuration. Separate assets are created for each sensor within Fledge with individual controls over the naming of these assets.

---

**Note:** The Sense HAT plugin is only available on the Raspberry Pi as it is specific the GPIO pins of that device.

---

To create a south service with the Sense HAT

- Click on *South* in the left hand menu bar
- Select *sensehat* from the plugin list
- Name your service and click *Next*

1 Plugin & Service Name			2 Review Configuration			3 Done		
Asset Name Prefix						sensehat/		
Pressure Sensor						<input checked="" type="checkbox"/>		
Pressure Sensor Name						pressure		
Temperature Sensor						<input checked="" type="checkbox"/>		
Temperature Sensor Name						temperature		
Humidity Sensor						<input checked="" type="checkbox"/>		
Humidity Sensor Name						humidity		
Gyroscope Sensor						<input checked="" type="checkbox"/>		
Gyroscope Sensor Name						gyroscope		
Accelerometer Sensor						<input checked="" type="checkbox"/>		
Accelerometer Sensor Name						accelerometer		
Magnetometer Sensor						<input checked="" type="checkbox"/>		
Magnetometer Sensor Name						magnetometer		
Joystick Sensor						<input checked="" type="checkbox"/>		
Joystick Sensor Name						joystick		
Previous						Next		

- Configure the plugin
  - **Asset Name Prefix:** An optional prefix to add to the asset names.
  - **Pressure Sensor:** A toggle control to turn on or off collection of pressure information
  - **Pressure Sensor Name:** Set a name for the Pressure sensor asset
  - **Temperature Sensor:** A toggle control to turn on or off collection of temperature information
  - **Temperature Sensor Name:** Set a name for the temperature sensor asset
  - **Humidity Sensor:** A toggle control to turn on or off collection of humidity information
  - **Humidity Sensor Name:** Set a name for the humidity sensor asset

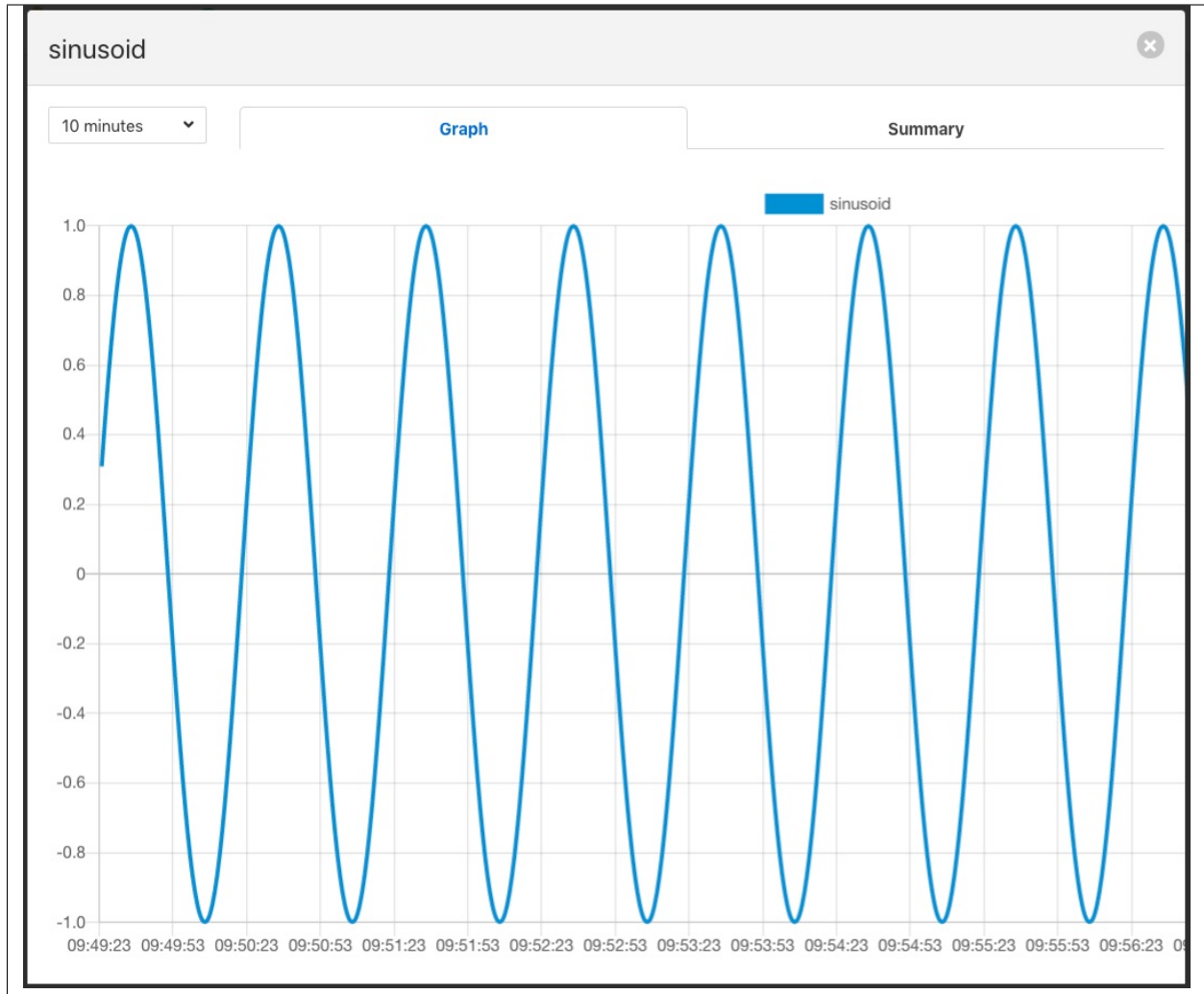


- **Gyroscope Sensor:** A toggle control to turn on or off collection of gyroscope information
  - **Gyroscope Sensor Name:** Set a name for the gyroscope sensor asset
  - **Accelerometer Sensor:** A toggle to turn on or off collection of accelerometer data
  - **Accelerometer Sensor Name:** Set a name for the accelerometer sensor asset
  - **Magnetometer Sensor:** A toggle control to turn on or off collection of magnetometer data
  - **Magnetometer Sensor Name:** Set a name for the magnetometer sensor asset
  - **Joystick Sensor:** A toggle control to turn on or off collection of joystick data
  - **Joystick Sensor Name:** Set a name for the joystick sensor asset
- Click *Next*
  - Enable the service and click on *Done*

### 8.1.26 Sinusoid

The *fledge-south-sinusoid* plugin is a south plugin that is primarily designed for testing purposes. It produces as it's output a simple sine wave, the period of which can be adjusted by changing the poll rate in the advanced settings of the south service into which it is loaded.





There is very little configuration required for the *sinusoid* plugin, merely the name of the asset that should be written. This can be useful if you wish to have multiple sinusoid in your Fledge system.

The screenshot shows the configuration wizard for the 'sinusoid' plugin. At the top, there is a progress bar with three steps: 1. Plugin & Service Name, 2. Review Configuration (current step), and 3. Done. Below the progress bar, there is a form with a label 'Asset name' and a text input field containing 'sinusoid'. At the bottom, there are two buttons: 'Previous' and 'Next'.

The frequency of the sinusoid can be adjusted by changing the poll rate of the sinusoid plugin. To do this select the



*South* item from the left-hand menu bar and then click on the name of your sinusoid service. You will see a link labeled *Show Advanced Config*, click on this to reveal the advanced configuration.

The screenshot shows a configuration window titled "Sine South Service" with a close button (X) in the top right corner. The window contains several configuration fields:









































- Asset name:** A text input field containing "sinusoid".
- Maximum Reading Latency (mS):** A text input field containing "5000".
- Maximum buffered Readings:** A text input field containing "100".
- Reading Rate:** A spin box with "10" selected.
- Throttle:** A checkbox that is currently unchecked.
- Reading Rate Per:** A dropdown menu showing "second".
- Minimum Log Level:** A dropdown menu showing "warning".
- Enabled:** A checkbox that is currently checked.

Below these fields is a section titled "Applications" with a plus icon. At the bottom right of the window are "Cancel" and "Save" buttons.

Amongst the advanced setting you will see one labeled *Reading Rate*. This defaults to 1 per second. The sinusoid takes 60 samples to complete one cycle of the sine wave, therefore it has a periodicity of 1 minute, or 0.0166Hz. If the *Reading Rate* is set to 60, then the frequency of the output becomes 1Hz.



### 8.1.27 System Information

Asset	Readings		
system/cpuUsage_all	94		
system/diskTraffic_loop0	94		
system/diskTraffic_loop1	94		
system/diskTraffic_loop2	94		
system/diskTraffic_sda	94		
system/diskTraffic_sdb	94		
system/diskUsage_dev/loop0	94		
system/diskUsage_dev/loop1	94		
system/diskUsage_dev/sda2	94		
system/diskUsage_dev/sdb1	94		
system/diskUsage_tmpfs	470		
system/diskUsage_udev	94		
system/hostname	94		
system/loadAverage	94		
system/memInfo	94		
system/networkTraffic_enp0s3	94		
system/networkTraffic_lo	94		
system/pagingAndSwappingEvents	94		
system/platform	94		
system/processes	94		
system/uptime	94		

The *fledge-south-systeminfo* plugin implements a that collects data about the machine that the Fledge instance is running on. The plugin is designed to allow the monitoring of the edge devices themselves to be included in the monitoring of the equipment involved in processing environment.

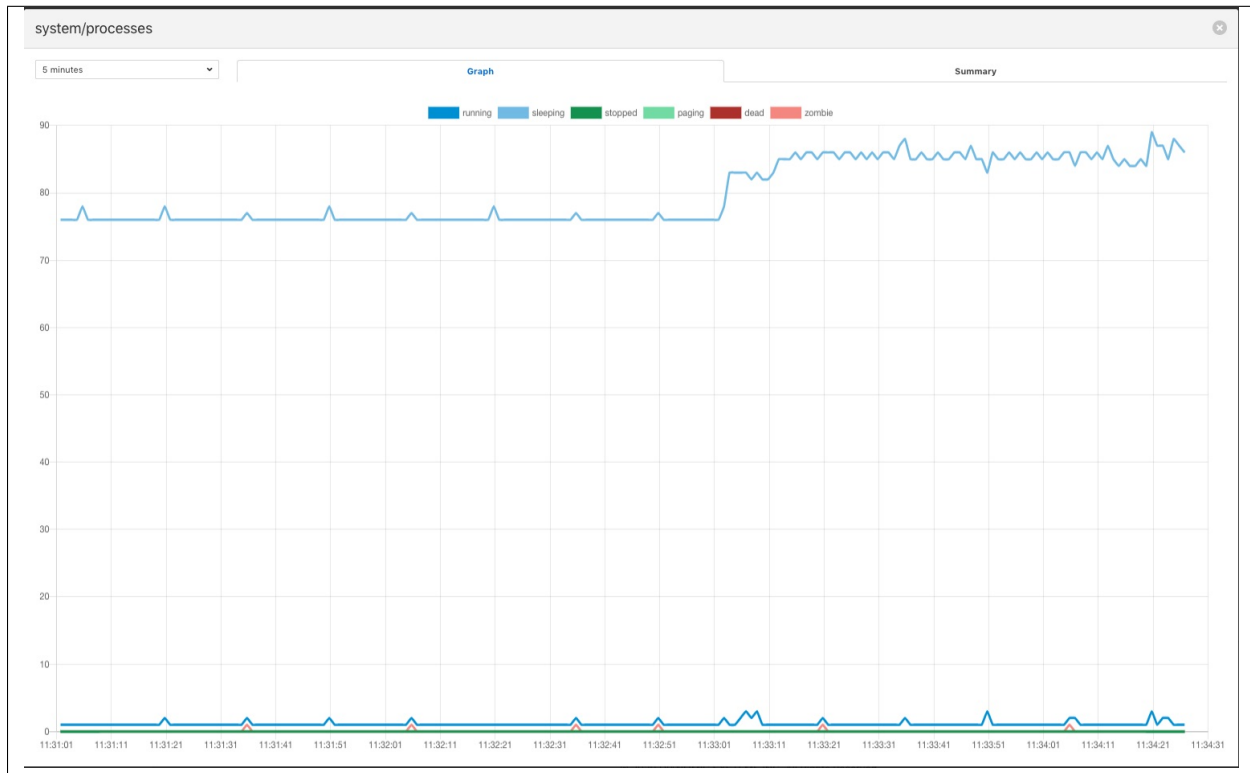
The plugin will create a number of assets, in general there are one or more assets per device connected in the case of disks and network interfaces. There are also some generic assets to measure;

- CPU Usage
- Host name
- Load Average
- Memory Usage



- Paging and swapping
- Process information
- System Uptime

A typical output for one of these assets, in this case the processes asset is shown below



To create a south service with the systeminfo plugin

- Click on *South* in the left hand menu bar
- Select *systeminfo* from the plugin list
- Name your service and click *Next*

- Configure the plugin
  - **Asset Name Prefix:** The asset name prefix for the assets created by this plugin. The plugin will create a number of assets, the exact number is dependent on the number of devices attached to



the machine.

- Click *Next*
- Enable the service and click on *Done*

### 8.1.28 Advantech USB-4704



The *fledge-south-usb4704* plugin is a south plugin that is designed to gather data from an Advantech USB-4704 data acquisition module. The module supports 8 digital inputs and 8 analogue inputs. It is possible to configure the plugin to combine multiple digital input to create a single numeric data point or have each input as a boolean data point. Each analogue input, which is a 14 bit analogue to digital converter, becomes a single numeric data point in the range 0 to 16383, although a scale and offset may be applied to these values.

To create a south service with the USB-4704

- Click on *South* in the left hand menu bar
- Select *usb4704* from the plugin list
- Name your service and click *Next*



1 Plugin & Service Name      2 Review Configuration      3 Done

**Asset Name**

**Connections**

```

1 {
2   "analogue_example": {
3     "type": "analogue",
4     "pin": "AI0",
5     "name": "value1",
6     "scale": 0.1
7   },
8   "digital_example": {
9     "type": "digital",
10    "pins": [
11      "DI0",
12      "DI1",
13      "DI2"

```

- Configure the plugin
  - **Asset Name:** The name of the asset that will be created with the values read from the USB-4704
  - **Connections:** A JSON document that describes the connections to the USB-4704 and the data points within the asset that they map to. The JSON document is a set of objects, one per data point. The objects contain a number of key/value pairs as follow

Key	Description
type	The type of connection, this may be either digital or analogue.
pin	The analogue pin used for the connection.
pins	An array of pins for a digital connection, the first element in the array becomes the most significant bit of the numeric value created.
name	The data point name within the asset.
scale	An optional scale value that may be applied to the value.

- Click on *Next*
- Enable your service and click on *Done*

## 8.2 Fledge North Plugins

### 8.2.1 OMF

The *OMF* north plugin is included in all distributions of the Fledge core and provides the north bound interface to the OSIsoft data historians in all it forms; PI Server, Edge Data Store and OSIsoft Cloud Services.



### PI Web API OMF Endpoint

To use the PI Web API OMF endpoint first ensure the OMF option was included in your PI Server when it was installed.

Now go to the Fledge user interface, create a new North instance and select the “OMF” plugin on the first screen. The second screen will request the following information:



?

Endpoint

PI Web API

Send full structure

☒

Naming Scheme

Concise

Server hostname

localhost

Server port, 0=use the default

0

Producer Token

omf\_north\_0001

Data Source

readings

Static Data

Location: Palo Alto, Company: Dianomic

Sleep Time Retry

1

Maximum Retry

3

HTTP Timeout

10

Integer Format

int64

Number Format

float64

Compression

☒

Default Asset Framework Location

/fledge/data\_piwebapi/default

Asset Framework hierarchy rules

1

{}

PI Web API Authentication Method

anonymous

PI Web API User Id

user\_id

PI Web API Password

\*\*\*\*\*

PI Web API Kerberos keytab file

piwebapi\_kerberos\_https.keytab

OCS Namespace

name\_space

OCS Tenant ID

ocs\_tenant\_id

OCS Client ID

ocs\_client\_id

OCS Client Secret

\*\*\*\*\*



Select PI Web API from the Endpoint options.

- **Basic Information**

- **Endpoint:** This is the type of OMF endpoint. In this case, choose PI Web API.
- **Send full structure:** Used to control if Asset Framework structure messages are sent to the PI Server. If this is turned off then the data will not be placed in the Asset Framework.
- **Naming scheme:** Defines the naming scheme to be used when creating the PI points in the PI Data Archive. See [Naming Scheme](#).
- **Server hostname:** The hostname or address of the PI Web API server. This is normally the same address as the PI Server.
- **Server port:** The port the PI Web API OMF endpoint is listening on. Leave as 0 if you are using the default port.
- **Data Source:** Defines which data is sent to the PI Server. Choices are: readings or statistics (that is, Fledge’s internal statistics).
- **Static Data:** Data to include in every reading sent to PI. For example, you can use this to specify the location of the devices being monitored by the Fledge server.

- **Asset Framework**

- **Default Asset Framework Location:** The location in the Asset Framework hierarchy into which the data will be inserted. All data will be inserted at this point in the Asset Framework hierarchy unless a later rule overrides this. Note this field does not include the name of the target Asset Framework Database; the target database is defined on the PI Web API server by the PI Web API Admin Utility.
- **Asset Framework Hierarchies Rules:** A set of rules that allow specific readings to be placed elsewhere in the Asset Framework. These rules can be based on the name of the asset itself or some metadata associated with the asset. See [Asset Framework Hierarchy Rules](#).

- **PI Web API authentication**

- **PI Web API Authentication Method:** The authentication method to be used: anonymous, basic or kerberos. Anonymous equates to no authentication, basic authentication requires a user name and password, and Kerberos allows integration with your single signon environment.
- **PI Web API User Id:** For Basic authentication, the user name to authenticate with the PI Web API.
- **PI Web API Password:** For Basic authentication, the password of the user we are using to authenticate.
- **PI Web API Kerberos keytab file:** The Kerberos keytab file used to authenticate.

- **Connection management (These should only be changed with guidance from support)**

- **Sleep Time Retry:** Number of seconds to wait before retrying the HTTP connection (Fledge doubles this time after each failed attempt).
- **Maximum Retry:** Maximum number of times to retry connecting to the PI Server.
- **HTTP Timeout:** Number of seconds to wait before Fledge will time out an HTTP connection attempt.

- **Other (Rarely changed)**

- **Integer Format:** Used to match Fledge data types to the data type configured in PI. This defaults to int64 but may be set to any OMF data type compatible with integer data, e.g. int32.
- **Number Format:** Used to match Fledge data types to the data type configured in PI. The default is float64 but may be set to any OMF datatype that supports floating point values.



- **Compression:** Compress the readings data before sending them to the PI Web API OMF endpoint. This setting is not related to data compression in the PI Data Archive.

### **Edge Data Store OMF Endpoint**

To use the OSIsoft Edge Data Store first install Edge Data Store on the same machine as your Fledge instance. It is a limitation of Edge Data Store that it must reside on the same host as any system that connects to it with OMF.

Now go to the Fledge user interface, create a new North instance and select the “OMF” plugin on the first screen. The second screen will request the following information:



?

Endpoint

Edge Data Store

Send full structure

☒

Naming Scheme

Concise

Server hostname

localhost

Server port, 0=use the default

0

Producer Token

omf\_north\_0001

Data Source

readings

Static Data

Location: Palo Alto, Company: Dianomic

Sleep Time Retry

1

Maximum Retry

3

HTTP Timeout

10

Integer Format

int64

Number Format

float64

Compression

☒

Default Asset Framework Location

/fledge/data\_piwebapi/default

Asset Framework hierarchy rules

1

{ }

PI Web API Authentication Method

anonymous

PI Web API User Id

user\_id

PI Web API Password

.....

PI Web API Kerberos keytab file

piwebapi\_kerberos\_https.keytab

OCS Namespace

name\_space

OCS Tenant ID

ocs\_tenant\_id

OCS Client ID

ocs\_client\_id

OCS Client Secret

.....



Select Edge Data Store from the Endpoint options.

- **Basic Information**

- **Endpoint:** This is the type of OMF endpoint. In this case, choose Edge Data Store.
- **Naming scheme:** Defines the naming scheme to be used when creating the PI points within the PI Server. See [Naming Scheme](#).
- **Server hostname:** Normally the hostname or address of the OMF endpoint. For Edge Data Store, this must be *localhost*.
- **Server port:** The port the Edge Data Store is listening on. Leave as 0 if you are using the default port.
- **Data Source:** Defines which data is sent to the Edge Data Store. Choices are: readings or statistics (that is, Fledge's internal statistics).
- **Static Data:** Data to include in every reading sent to PI. For example, you can use this to specify the location of the devices being monitored by the Fledge server.

- **Connection management (These should only be changed with guidance from support)**

- **Sleep Time Retry:** Number of seconds to wait before retrying the HTTP connection (Fledge doubles this time after each failed attempt).
- **Maximum Retry:** Maximum number of times to retry connecting to the PI server.
- **HTTP Timeout:** Number of seconds to wait before Fledge will time out an HTTP connection attempt.

- **Other (Rarely changed)**

- **Integer Format:** Used to match Fledge data types to the data type configured in PI. This defaults to int64 but may be set to any OMF data type compatible with integer data, e.g. int32.
- **Number Format:** Used to match Fledge data types to the data type configured in PI. The default is float64 but may be set to any OMF datatype that supports floating point values.
- **Compression:** Compress the readings data before sending them to the Edge Data Store.



### OSIsoft Cloud Services OMF Endpoint

Go to the Fledge user interface, create a new North instance and select the “OMF” plugin on the first screen. The second screen will request the following information:



Endpoint

OSIsoft Cloud Services

Send full structure

☒

Naming Scheme

Concise

Server hostname

localhost

Server port, 0=use the default

0

Producer Token

omf\_north\_0001

Data Source

readings

Static Data

Location: Palo Alto, Company: Dianomic

Sleep Time Retry

1

Maximum Retry

3

HTTP Timeout

10

Integer Format

int64

Number Format

float64

Compression

☒

Default Asset Framework Location

/fledge/data\_piwebapi/default

Asset Framework hierarchy rules

1

{}

PI Web API Authentication Method

anonymous

PI Web API User Id

user\_id

PI Web API Password

.....

PI Web API Kerberos keytab file

piwebapi\_kerberos\_https.keytab

OCS Namespace

name\_space

OCS Tenant ID

ocs\_tenant\_id

OCS Client ID

ocs\_client\_id

OCS Client Secret

.....



Select OSIsoft Cloud Services from the Endpoint options.

- **Basic Information**

- **Endpoint:** This is the type of OMF endpoint. In this case, choose OSIsoft Cloud Services.
- **Naming scheme:** Defines the naming scheme to be used when creating the PI points within the PI Server. See [Naming Scheme](#).
- **Data Source:** Defines which data is sent to OSIsoft Cloud Services. Choices are: readings or statistics (that is, Fledge’s internal statistics).
- **Static Data:** Data to include in every reading sent to OSIsoft Cloud Services. For example, you can use this to specify the location of the devices being monitored by the Fledge server.

- **Authentication**

- **OCS Namespace:** Your namespace within OSIsoft Cloud Services.
- **OCS Tenant ID:** Your OSIsoft Cloud Services Tenant ID for your account.
- **OCS Client ID:** Your OSIsoft Cloud Services Client ID for your account.
- **OCS Client Secret:** Your OSIsoft Cloud Services Client Secret.

- **Connection management (These should only be changed with guidance from support)**

- **Sleep Time Retry:** Number of seconds to wait before retrying the HTTP connection (Fledge doubles this time after each failed attempt).
- **Maximum Retry:** Maximum number of times to retry connecting to the PI server.
- **HTTP Timeout:** Number of seconds to wait before Fledge will time out an HTTP connection attempt.

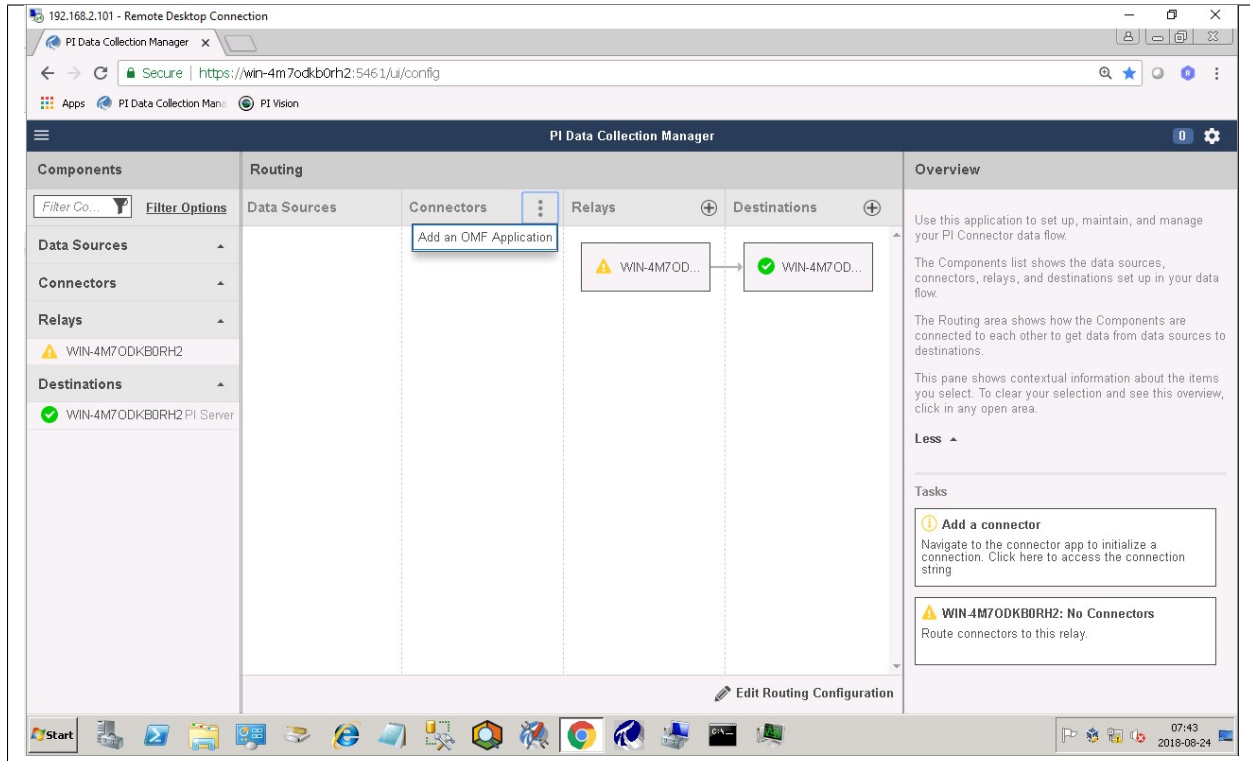
- **Other (Rarely changed)**

- **Integer Format:** Used to match Fledge data types to the data type configured in PI. This defaults to int64 but may be set to any OMF data type compatible with integer data, e.g. int32.
- **Number Format:** Used to match Fledge data types to the data type configured in PI. The default is float64 but may be set to any OMF datatype that supports floating point values.
- **Compression:** Compress the readings data before sending them to OSIsoft Cloud Services.

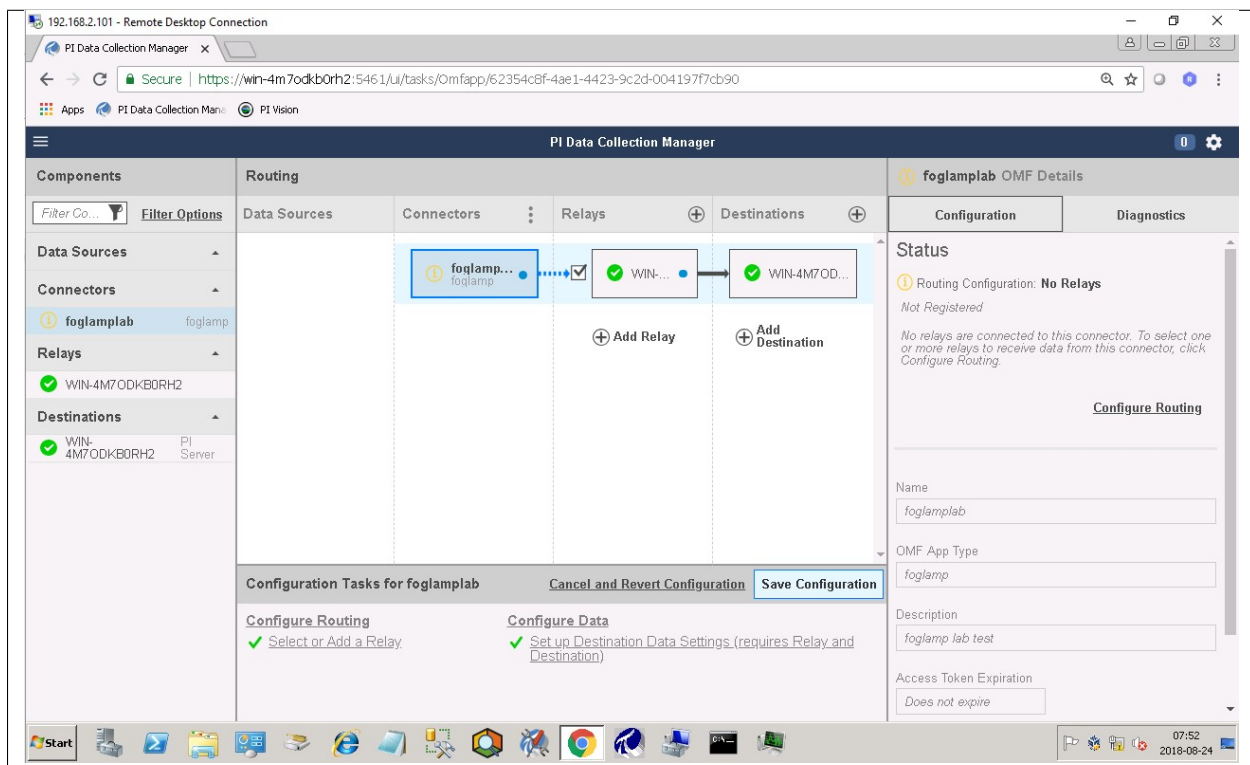
## PI Connector Relay

**The PI Connector Relay has been discontinued by OSIsoft.** All new deployments should use the PI Web API endpoint. Existing installations will still be supported. The PI Connector Relay was the original mechanism by which OMF data could be ingesting into a PI Server. To use the PI Connector Relay, open and sign into the PI Relay Data Connection Manager.



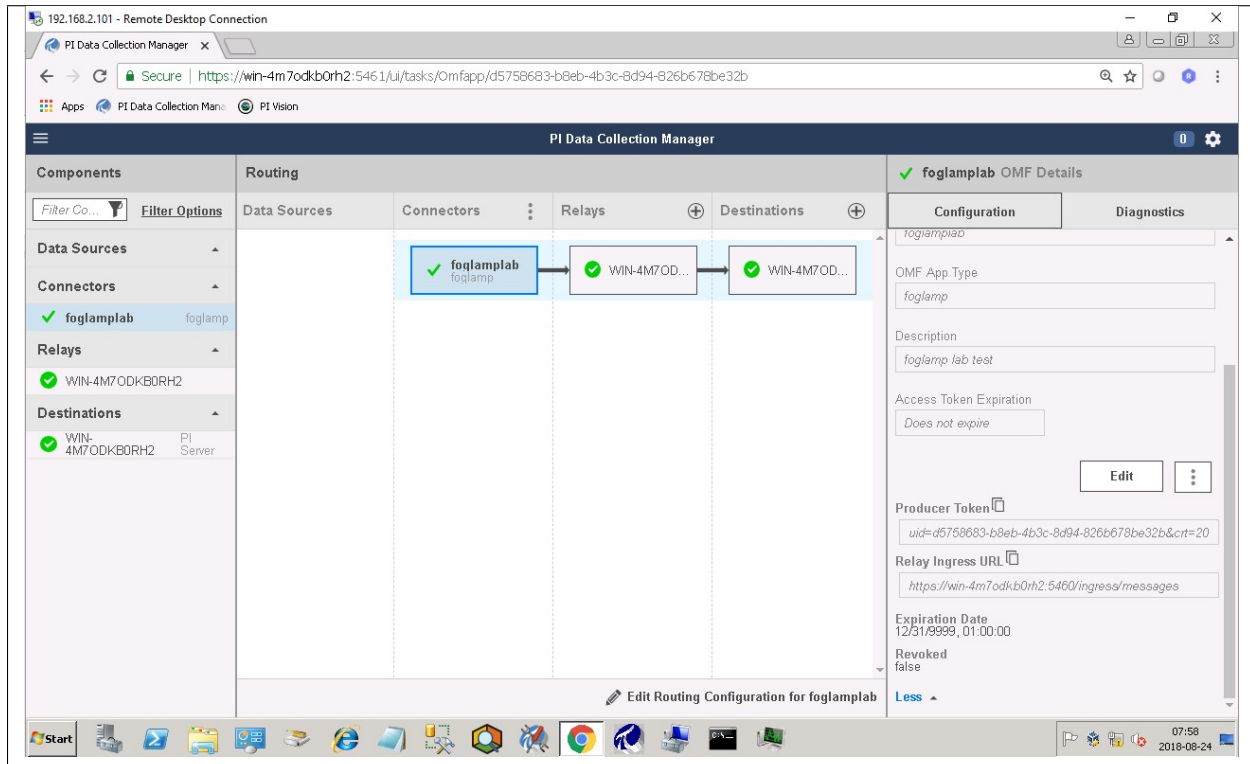


To add a new connector for the Fledge system, click on the drop down menu to the right of “Connectors” and select “Add an OMF application”. Add and save the requested configuration information.





Connect the new application to the PI Connector Relay by selecting the new Fledge application, clicking the check box for the PI Connector Relay and then clicking “Save Configuration”.



Finally, select the new Fledge application. Click “More” at the bottom of the Configuration panel. Make note of the Producer Token and Relay Ingress URL.

Now go to the Fledge user interface, create a new North instance and select the “OMF” plugin on the first screen. The second screen will request the following information:



?

Endpoint

Connector Relay

Send full structure

☒

Naming Scheme

Concise

Server hostname

localhost

Server port, 0=use the default

0

Producer Token

omf\_north\_0001

Data Source

readings

Static Data

Location: Palo Alto, Company: Dianomic

Sleep Time Retry

1

Maximum Retry

3

HTTP Timeout

10

Integer Format

int64

Number Format

float64

Compression

☒

Default Asset Framework Location

/fledge/data\_piwebapi/default

Asset Framework hierarchy rules

1

{}

PI Web API Authentication Method

anonymous

PI Web API User Id

user\_id

PI Web API Password

.....

PI Web API Kerberos keytab file

piwebapi\_kerberos\_https.keytab

OCS Namespace

name\_space

OCS Tenant ID

ocs\_tenant\_id

OCS Client ID

ocs\_client\_id

OCS Client Secret

.....



- **Basic Information**

- **Endpoint:** This is the type of OMF endpoint. In this case, choose Connector Relay.
- **Server hostname:** The hostname or address of the PI Connector Relay.
- **Server port:** The port the PI Connector Relay is listening on. Leave as 0 if you are using the default port.
- **Producer Token:** The Producer Token provided by the PI Relay Data Connection Manager.
- **Data Source:** Defines which data is sent to the PI Connector Relay. Choices are: readings or statistics (that is, Fledge's internal statistics).
- **Static Data:** Data to include in every reading sent to PI. For example, you can use this to specify the location of the devices being monitored by the Fledge server.

- **Connection management (These should only be changed with guidance from support)**

- **Sleep Time Retry:** Number of seconds to wait before retrying the HTTP connection (Fledge doubles this time after each failed attempt).
- **Maximum Retry:** Maximum number of times to retry connecting to the PI server.
- **HTTP Timeout:** Number of seconds to wait before Fledge will time out an HTTP connection attempt.

- **Other (Rarely changed)**

- **Integer Format:** Used to match Fledge data types to the data type configured in PI. This defaults to int64 but may be set to any OMF data type compatible with integer data, e.g. int32.
- **Number Format:** Used to match Fledge data types to the data type configured in PI. The default is float64 but may be set to any OMF datatype that supports floating point values.
- **Compression:** Compress the readings data before sending it to the PI System.

## Naming Scheme

The naming of objects in the Asset Framework and of the attributes of those objects has a number of constraints that need to be understood when storing data into a PI Server using OMF. An important factor in this is the stability of your data structures. If you have objects in your environment that are likely to change, you may wish to take a different naming approach. Examples of changes are a difference in the number of attributes between readings, and a change in the data types of attributes.

This occurs because of a limitation of the OMF interface to the PI Server. Data is sent to OMF in a number of stages. One of these is the definition of the Types used to create AF Element Templates. OMF uses a Type to define an AF Element Template but once defined it cannot be changed. If an updated Type definition is sent to OMF, it will be used to create a new AF Element Template rather than changing the existing one. This means a new AF Element Template is created each time a Type changes.

The OMF plugin names objects in the Asset Framework based upon the asset name in the reading within Fledge. Asset names are typically added to the readings in the south plugins, however they may be altered by filters between the south ingest and the north egress points in the data pipeline. Asset names can be overridden using the *OMF Hints* mechanism described below.

The attribute names used within the objects in the PI System are based on the names of the datapoints within each Reading within Fledge. Again *OMF Hints* can be used to override this mechanism.

The naming used within the objects in the Asset Framework is controlled by the *Naming Scheme* option:



**Concise** No suffix or prefix is added to the asset name and property name when creating objects in the Asset Framework and PI Points in the PI Data Archive. However, if the structure of an asset changes a new AF Element Template will be created which will have the suffix `-type*x*` appended to it.

**Use Type Suffix** The AF Element names will be created from the asset names by appending the suffix `-type*x*` to the asset name. If the structure of an asset changes a new AF Element name will be created with an updated suffix.

**Use Attribute Hash** AF Attribute names will be created using a numerical hash as a prefix.

**Backward Compatibility** The naming reverts to the rules that were used by version 1.9.1 and earlier of Fledge: both type suffixes and attribute hashes will be applied to the name.

## Asset Framework Hierarchy Rules

The Asset Framework rules allow the location of specific assets within the Asset Framework to be controlled. There are two basic types of hint:

- Asset name placement: the name of the asset determines where in the Asset Framework the asset is placed,
- Meta data placement: metadata within the reading determines where the asset is placed in the Asset Framework.

The rules are encoded within a JSON document. This document contains two properties in the root of the document: one for name-based rules and the other for metadata based rules.

```
{
  "names" :
  {
    "asset1" : "/Building1/EastWing/GroundFloor/Room4",
    "asset2" : "Room14"
  },
  "metadata" :
  {
    "exist" :
    {
      "temperature" : "temperatures",
      "power" : "/Electrical/Power"
    },
    "nonexist" :
    {
      "unit" : "Uncalibrated"
    }
    "equal" :
    {
      "room" :
      {
        "4" : "ElecticalLab",
        "6" : "FluidLab"
      }
    }
    "notequal" :
    {
      "building" :
      {
        "plant" : "/Office/Environment"
      }
    }
  }
}
```

(continues on next page)



(continued from previous page)

```
}  
}
```

The name type rules are simply a set of asset name and Asset Framework location pairs. The asset names must be complete names; there is no pattern matching within the names.

The metadata rules are more complex. Four different tests can be applied:

- **exists:** This test looks for the existence of the named datapoint within the asset.
- **nonexist:** This test looks for the lack of a named datapoint within the asset.
- **equal:** This test looks for a named datapoint having a given value.
- **notequal:** This test looks for a name datapoint having a value different from that specified.

The *exist* and *nonexist* tests take a set of name/value pairs that are tested. The name is the datapoint name to examine and the value is the Asset Framework location to use. For example

```
"exist" :  
{  
    "temperature" : "temperatures",  
    "power"       : "/Electrical/Power"  
}
```

If an asset has a datapoint called *temperature* it will be stored in the AF hierarchy *temperatures*, if the asset had a datapoint called *power* the asset will be placed in the AF hierarchy */Electrical/Power*.

The *equal* and *notequal* tests take an object as a child, the name of the object is datapoint to examine, the child nodes are sets of values and locations. For example

```
"equal" :  
{  
    "room" :  
    {  
        "4" : "ElectricalLab",  
        "6" : "FluidLab"  
    }  
}
```

In this case if the asset has a datapoint called *room* with a value of *4* then the asset will be placed in the AF location *ElectricalLab*, if it has a value of *6* then it is placed in the AF location *FluidLab*.

If an asset matches multiple rules in the ruleset it will appear in multiple locations in the hierarchy, the data is shared between each of the locations.

If an OMF Hint exists within a particular reading this will take precedence over generic rules.

The AF location may be a simple string or it may also include substitutions from other datapoints within the reading. For example if the reading has a datapoint called *room* that contains the room in which the readings were taken, an AF location of */BuildingA/\${room}* would put the reading in the Asset Framework using the value of the room datapoint. The reading

```
"reading" : {  
    "temperature" : 23.4,  
    "room"        : "B114"  
}
```

would be put in the AF at */BuildingA/B114* whereas a reading of the form



```
"reading" : {
  "temperature" : 24.6,
  "room"       : "2016"
}
```

would be put at the location */BuildingA/2016*.

It is also possible to define defaults if the referenced datapoint is missing. In our example above if we used the location */BuildingA/\${room:unknown}* a reading without a *room* datapoint would be placed in */BuildingA/unknown*. If no default is given and the data point is missing then the level in the hierarchy is ignored. E.g. if we use our original location */BuildingA/\${room}* and we have the reading

```
"reading" : {
  "temperature" : 22.8,
}
```

this reading would be stored in */BuildingA*.

## OMF Hints

The OMF plugin also supports the concept of hints in the actual data that determine how the data should be treated by the plugin. Hints are encoded in a specially named datapoint within the asset, *OMFHint*. The hints themselves are encoded as JSON within a string.

### Number Format Hints

A number format hint tells the plugin what number format to use when inserting data into the PI Server. The following will cause all numeric data within the asset to be written using the format *float32*.

```
"OMFHint" : { "number" : "float32" }
```

The value of the *number* hint may be any numeric format that is supported by the PI Server.

### Integer Format Hints

An integer format hint tells the plugin what integer format to use when inserting data into the PI Server. The following will cause all integer data within the asset to be written using the format *integer32*.

```
"OMFHint" : { "number" : "integer32" }
```

The value of the *number* hint may be any numeric format that is supported by the PI Server.

## Type Name Hints

A type name hint specifies that a particular name should be used when defining the name of the type that will be created to store the object in the Asset Framework. This will override the *Naming Scheme* currently configured.

```
"OMFHint" : { "typeName" : "substation" }
```



### Type Hint

A type hint is similar to a type name hint, but instead of defining the name of a type to create it defines the name of an existing type to use. The structure of the asset *must* match the structure of the existing type with the PI Server, it is the responsibility of the person that adds this hint to ensure this is the case.

```
"OMFHint" : { "type" : "pump" }
```

### Tag Name Hint

Specifies that a specific tag name should be used when storing data in the PI Server.

```
"OMFHint" : { "tagName" : "AC1246" }
```

### Datapoint Specific Hint

Hints may also be targeted to specific data points within an asset by using the datapoint hint. A *datapoint* hint takes a JSON object as its value; the object defines the name of the datapoint and the hint to apply.

```
"OMFHint" : { "datapoint" : { "name" : "voltage:", "number" : "float32" } }
```

The above hint applies to the datapoint *voltage* in the asset and applies a *number format* hint to that datapoint.

### Asset Framework Location Hint

An Asset Framework location hint can be added to a reading to control the placement of the asset within the Asset Framework. An Asset Framework hint would be as follows:

```
"OMFHint" : { "AFLocation" : "/UK/London/TowerHill/Floor4" }
```

Note the following when defining an *AFLocation* hint:

- An asset in a Fledge Reading is used to create a [Container in the OSIsoft Asset Framework](#). A *Container* is an AF Element with one or more AF Attributes that are mapped to PI Points using the OSIsoft PI Point Data Reference. The name of the AF Element comes from the Fledge Reading asset name. The names of the AF Attributes come from the Fledge Reading datapoint names.
- If you edit the AF Location hint, the Container will be moved to the new location in the AF hierarchy.
- If you disable the OMF Hint filter, the Container will not move.
- If you wish to move a Container, you can do this with the PI System Explorer. Right-click on the AF Element that represents the Container. Choose Copy. Select the AF Element that will serve as the new parent of the Container. Right-click and choose *Paste*. You can then return to the original Container and delete it. *Note that PI System Explorer does not have the traditional Cut function for AF Elements.*
- If you move a Container, OMF North will not recreate it. If you then edit the AF Location hint, the Container will appear in the new location.



## Adding OMF Hints

An OMF Hint is implemented as a string data point on a reading with the data point name of *OMFHint*. It can be added at any point in the processing of the data, however a specific plugin is available for adding the hints, the .

### 8.2.2 Azure IoT Hub

The *fledge-north-azure* plugin sends data from Fledge to the Microsoft Azure IoT Core service.

The configuration of the *Azure* plugin requires a few simple configuration parameters to be set.

- **Primary Connection String:** The primary connection string to connect to your Azure IoT project. The connection string should contain
  - The hostname to connect to
  - The DeviceID of the device you are using
  - The shared access key generated on your Azure login
- **MQTT over websockets:** Enable if you wish to run MQTT over websockets.
- **Data Source:** Which Fledge data to send to Azure; Readings or Fledge Statistics.
- **Apply Filter:** This allows a simple jq format filter rule to be applied to the connection. This should not be confused with Fledge filters and exists for backward compatibility reason only.
- **Filter Rule:** A jq filter rule to apply. Since the introduction of Fledge filters in the north task this has become deprecated and should not be used.



### JSON Payload

The payload that is sent by this plugin to Azure is a simple JSON representation of a set of reading values. A JSON array is sent with one or more reading objects contained within it. Each reading object consists of a timestamp, an asset name and a set of data points within that asset. The data points are represented as name value pair JSON properties within the reading property.

The fixed part of every reading contains the following

Name	Description
times-tamp	The timestamp as an ASCII string in ISO 8601 extended format. If no time zone information is given it is assumed to indicate the use of UTC.
asset	The name of the asset this reading represents.
read-ings	A JSON object that contains the data points for this asset.

The content of the *readings* object is a set of JSON properties, each of which represents a data value. The type of these values may be integer, floating point, string, a JSON object or an array of floating point numbers.

A property

```
"voltage" : 239.4
```

would represent a numeric data value for the item *voltage* within the asset. Whereas

```
"voltageUnit" : "volts"
```

Is string data for that same asset. Other data may be presented as arrays

```
"acceleration" : [ 0.4, 0.8, 1.0 ]
```

would represent acceleration with the three components of the vector, x, y, and z. This may also be represented as an object

```
"acceleration" : { "X" : 0.4, "Y" : 0.8, "Z" : 1.0 }
```

both are valid formats within Fledge.

An example payload with a single reading would be as shown below

```
[
  {
    "timestamp" : "2020-07-08 16:16:07.263657+00:00",
    "asset"      : "motor1",
    "readings"   : {
      "voltage"  : 239.4,
      "current"  : 1003,
      "rpm"      : 120147
    }
  }
]
```



### 8.2.3 Google Cloud Platform North Plugin

The *fledge-north-gcp* plugin provide connectivity from a Fledge system to the Google Cloud Platform. The plugin connects to the IoT Core in Google Cloud using MQTT and is fully compliant with the security features of the Google Cloud Platform. See for a tutorial on setting up a Fledge system and getting it to send data to Google Cloud.

#### Prerequisites

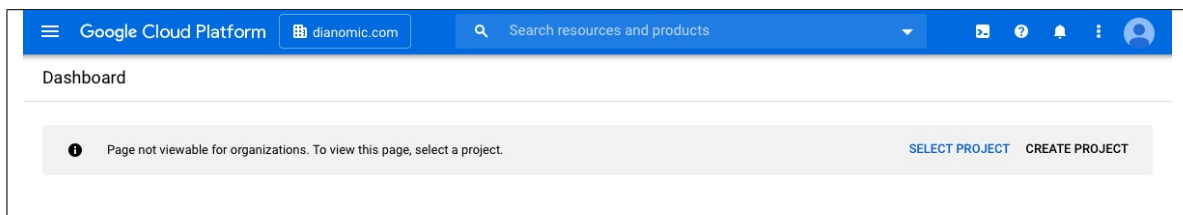
A number of things must be done in the Google Cloud before you can create your north connection to GCP. You must

- Create a GCP IoT Core project
- Download the roots.pem certificate from your GCP account
- Create a registry
- Create a device ID and configure a key pair for that device
- Upload the certificates to the Fledge certificate store

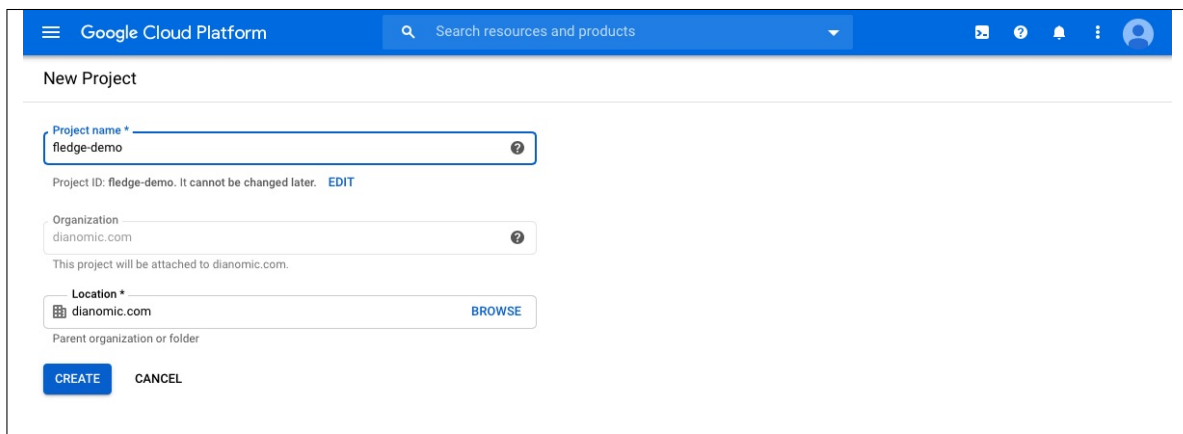
#### Create GCP IoT Core Project

To create a new project

- Goto the
- Select the Projects page and select the *Create New Project* option



- Enter your project details





### Download roots.pem

To download the roots.pem security certificate

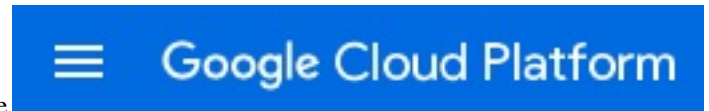
- From the command line shell of your machine run the command

```
$ wget https://pki.goog/roots.pem
```

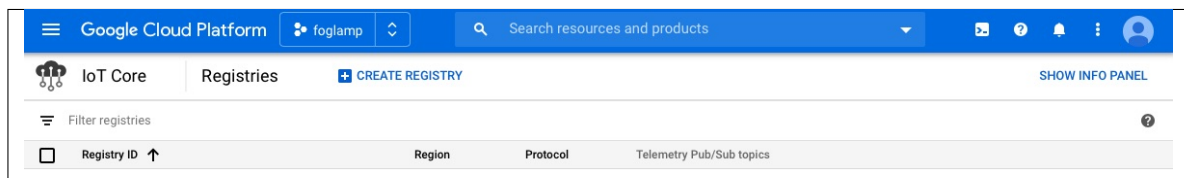
### Create a Registry

To create a registry in your project

- Goto the



- Click on the menu icon in the top left corner of the page
- Select the *Create Registry* option



- A new screen is shown that allows you to create a new registry



- Note the Registry ID and region as you will need these later
- Select an existing telemetry topic or create a new topic (for example, projects/[YOUR\_PROJECT\_ID]/topics/[REGISTRY\_ID])
- Click on *Create*

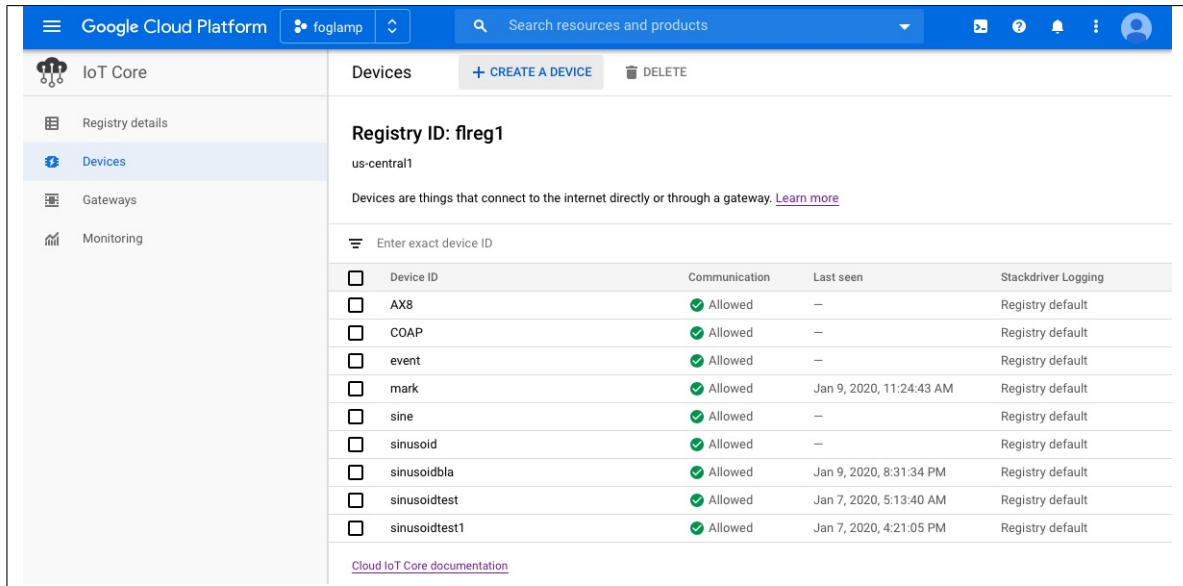
## Create a Device ID

To create a device in your Google Cloud Project

- Create an RSA public/private key pair on your local machine

```
openssl genpkey -algorithm RSA -out rsa_fledge.pem -pkeyopt rsa_keygen_bits:2048
openssl rsa -in rsa_fledge.pem -pubout -out rsa_fledge.pem
```

- Goto the
- In the left pane of the IoT Core page in the Cloud Console, click Devices



The screenshot shows the Google Cloud Platform IoT Core console. The left sidebar contains the 'IoT Core' menu with options: Registry details, Devices (selected), Gateways, and Monitoring. The main content area is titled 'Devices' and shows the Registry ID: freg1. Below this, there is a table of devices with columns: Device ID, Communication, Last seen, and Stackdriver Logging. The table lists several devices, all with 'Allowed' communication status and 'Registry default' logging.

Device ID	Communication	Last seen	Stackdriver Logging
AX8	Allowed	—	Registry default
COAP	Allowed	—	Registry default
event	Allowed	—	Registry default
mark	Allowed	Jan 9, 2020, 11:24:43 AM	Registry default
sine	Allowed	—	Registry default
sinusoid	Allowed	—	Registry default
sinusoidbla	Allowed	Jan 9, 2020, 8:31:34 PM	Registry default
sinusoidtest	Allowed	Jan 7, 2020, 5:13:40 AM	Registry default
sinusoidtest1	Allowed	Jan 7, 2020, 4:21:05 PM	Registry default

- At the top of the Devices page, click *Create a device*



- Enter a device ID, you will need to add this in the north plugin configuration later
- Click on the *ADD ATTRIBUTE COMMUNICATION, STACKDRIVER LOGGING, AUTHENTICATION* link to open the remainder of the inputs
- Make sure the public key format matches the type of key that you created in the first step of this section (for example, RS256)
- Paste the contents of your public key in the Public key value field.

## Upload Your Certificates

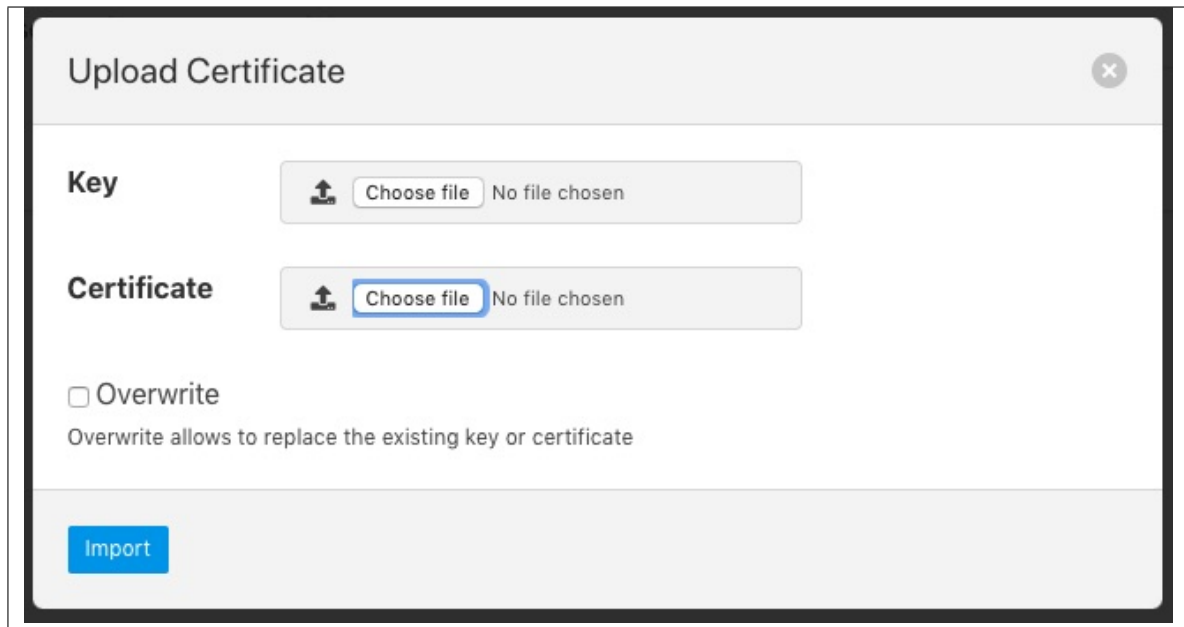
You should upload your certificates to Fledge

- From the Fledge user interface select the *Certificate Store* from the left-hand menu bar

Key	Extension		Certificate	Extension	
admin	key	<a href="#">delete</a>	admin	cert	<a href="#">delete</a>
ca	key	<a href="#">delete</a>	ca	cert	<a href="#">delete</a>
fledge	key		fledge	cert	
user	key	<a href="#">delete</a>	user	cert	<a href="#">delete</a>
			root	cert	<a href="#">delete</a>

- Click on the Import option in the top left corner





Upload Certificate

**Key**

Choose file No file chosen

**Certificate**

Choose file No file chosen

☐ Overwrite

Overwrite allows to replace the existing key or certificate

Import

- In the Certificate option select the *Choose file* option and select your roots.pem and click on open
- Repeat the above for your device key and certificate

### Create Your North Task

Having completed the pre-requisite steps it is now possible to create the north task to send data to GCP.

- Select the *North* option from the left-hand menu bar.
- Select GCP from the North Plugin list
- Name your North task and click on *Next*



The screenshot shows the 'Review Configuration' step of the Fledge plugin setup. The progress bar at the top indicates three steps: 1. Plugin & Name, 2. Review Configuration (current step), and 3. Done. The configuration form contains the following fields and values:

Field	Value
Project ID	fledge-demo
The GCP Region	us-central1
Registry ID	fledge
Device ID	demo
Key Name	fledge_private
JWT Algorithm	RS256
Data Source	readings

At the bottom of the form, there are two buttons: 'Back' and 'Next'.

- Configure your GCP plugin
  - **Project ID:** Enter the project ID you created in GCP
  - **The GCP Region:** Select the region in which you created your registry
  - **Registry ID:** The Registry ID you created should be entered here
  - **Device ID:** The Device ID you created should be entered here
  - **Key Name:** Enter the name of the device key you uploaded to the certificate store
  - **JWT Algorithm:** Select the algorithm that matches the key you created earlier
  - **Data Source:** Select the data to send to GCP, this may be readings or Fledge statistics
- Click on *Next*
- Enable your plugin and click on *Done*

## 8.2.4 HarperDB

The *fledge-north-harperdb* plugin sends data from Fledge to the database. HarperDB is a geo-distributed database with hybrid SQL & NoSQL functionality in one powerful tool, accessed via a REST API. Each asset that is read by Fledge is written to a separate table within the specified HarperDB schema. The plugin will support both local installations and cloud installations of HarperDB.

The configuration of the *HarperDB* plugin requires a few simple configuration parameters to be set.



1 Plugin & Name 2 Review Configuration 3 Done

URL

Username

Password

Schema

Source

Back Next

- **URL:** The URL of the HarperDB database that will be used to store the data sent from Fledge. This may be either an HTTP or HTTPS URL
- **Username:** The username to use when authenticating with the HarperDB database.
- **Password:** The password of the user that will be used to store the data in HarperDB.
- **Schema:** The name of the schema in the HarperDB database in which the tables will be stored.
- **Source:** The source of the Fledge data to store in the HarperDB database; Readings or Fledge Statistics.

## 8.2.5 North HTTP

The *fledge-north-http* plugin allows data to be sent from the north of one Fledge instance into the south of another Fledge instance. It allows hierarchies of Fledge instances to be built. The Fledge to which the data is sent must run the corresponding in order for data to flow between the two Fledge instances. The plugin supports both HTTP and HTTPS transport protocols and sends a JSON payload of reading data in the internal Fledge format.

The plugin may also be used to send data from Fledge to another system, the receiving system should implement a REST end point that will accept a POST request containing JSON data. The format of the JSON payload is described below. The required REST endpoint path is defined in the configuration of the plugin.

Filters may be applied to the connection in either the north task that loads this plugin or the receiving south service on the up stream Fledge.

A of this plugin exists also that performs the same function as this plugin, the pair are provided for purposes of comparison and the user may choose whichever they prefer to use.

To create a north task to send to another Fledge you should first create the that will receive the data. Then create a new north tasks by;

- Selecting *North* from the left hand menu bar.
- Click on the + icon in the top left
- Choose `http_north` from the plugin selection list
- Name your task
- Click on *Next*



- Configure the plugin

- **URL:** The URL of the receiving , the address and port should match the service in the up stream Fledge. The URL can specify either HTTP or HTTPS protocols.
- **Source:** The data to send, this may be either the reading data or the statistics data
- **Verify SSL:** When HTTPS rather the HTTP is used this toggle allows for the verification of the certificate that is used. If a self signed certificate is used then this should not be enabled.
- **Apply Filter:** This allows a simple jq format filter rule to be applied to the connection. This should not be confused with Fledge filters and exists for backward compatibility reason only.
- **Filter Rule:** A jq filter rule to apply. Since the introduction of Fledge filters in the north task this has become deprecated and should not be used.

- Click *Next*
- Enable your task and click *Done*

## JSON Payload

The payload that is sent by this plugin is a simple JSON presentation of a set of reading values. A JSON array is sent with one or more reading objects contained within it. Each reading object consists of a timestamp, an asset name and a set of data points within that asset. The data points are represented as name value pair JSON properties within the reading property.

The fixed part of every reading contains the following

Name	Description
times-tamp	The timestamp as an ASCII string in ISO 8601 extended format. If no time zone information is given it is assumed to indicate the use of UTC.
asset	The name of the asset this reading represents.
read-ings	A JSON object that contains the data points for this asset.

The content of the *readings* object is a set of JSON properties, each of which represents a data value. The type of these values may be integer, floating point, string, a JSON object or an array of floating point numbers.



A property

```
"voltage" : 239.4
```

would represent a numeric data value for the item *voltage* within the asset. Whereas

```
"voltageUnit" : "volts"
```

Is string data for that same asset. Other data may be presented as arrays

```
"acceleration" : [ 0.4, 0.8, 1.0 ]
```

would represent acceleration with the three components of the vector, x, y, and z. This may also be represented as an object

```
"acceleration" : { "X" : 0.4, "Y" : 0.8, "Z" : 1.0 }
```

both are valid formats within Fledge.

An example payload with a single reading would be as shown below

```
[
  {
    "timestamp" : "2020-07-08 16:16:07.263657+00:00",
    "asset"      : "motor1",
    "readings"   : {
      "voltage"  : 239.4,
      "current"  : 1003,
      "rpm"      : 120147
    }
  }
]
```

## 8.2.6 North HTTP-C

The *fledge-north-http-c* plugin allows data to be sent from the north of one Fledge instance into the south of another Fledge instance. It allows hierarchies of Fledge instances to be built. The Fledge to which the data is sent must run the corresponding in order for data to flow between the two Fledge instances. The plugin supports both HTTP and HTTPS transport protocols and sends a JSON payload of reading data in the internal Fledge format.

Additionally this plugin allows for two URL's to be configured, a primary URL and a secondary URL. If the connection to the primary URL fails then the plugin will switch over to using the secondary URL. It will switch back if the connection to the secondary fails or if when the north task completes and a new north task is later run.

The plugin may also be used to send data from Fledge to another system, the receiving system should implement a REST end point that will accept a POST request containing JSON data. The format of the JSON payload is described below. The required REST endpoint path is defined in the configuration of the plugin.

Filters may be applied to the connection in either the north task that loads this plugin or the receiving south service on the up stream Fledge.

A plugin exists also that performs the same function as this plugin, the pair are provided for purposes of comparison and the user may choose whichever they prefer to use.

To create a north task to send to another Fledge you should first create the that will receive the data. Then create a new north tasks by;

- Selecting *North* from the left hand menu bar.



- Click on the + icon in the top left
- Choose httpc from the plugin selection list
- Name your task
- Click on *Next*
- Configure the HTTP-C plugin

The screenshot displays the 'Review Configuration' step of the Fledge plugin configuration process. The interface is divided into a left sidebar with labels for configuration sections and a main area with input fields. The sections and their values are: URL (http://localhost:6683/sensor-reading), Secondary URL (empty), Proxy (empty), Source (readings), Headers (a table with one row containing an empty key and a value of {}), Script (empty), Sleep Time Retry (1), Maximum Retry (3), Http Timeout (in seconds) (10), and Verify SSL (unchecked). At the bottom, there are 'Back' and 'Next' buttons.

- **URL:** The URL of the receiving , the address and port should match the service in the up stream Fledge. The URL can specify either HTTP or HTTPS protocols.
- **Secondary URL:** The URL to failover to if the connection to the primary URL fails. If failover is not required then leave this field empty.
- **Source:** The data to send, this may be either the reading data or the statistics data
- **Proxy:** The host and port of the proxy server to use. Leave empty if a proxy is not in use. This should be formatted as an address followed by a colon and then the port or a hostname followed by a colon and then the port. E.g. 192.168.0.42:8080. If the default port is used then the port may be omitted.
- **Headers:** An optional set of header fields to send in every request. The headers are defined as a JSON document with the name of each item in the document as header field name and the value the value of the header field.



- **Script:** An optional Python script that can be used to convert the payload format. If given the script should contain a method called *convert* that will be passed a single reading as a JSON DICT and must return the new payload as a string.
- **Sleep Time Retry:** A tuning parameter used to control how often a connection is retried to the up stream Fledge if it is not available. On every retry the time will be doubled.
- **Maximum Retry:** The maximum number of retries to make a connection to the up stream Fledge. When this number is reached the current execution of the task is suspended until the next scheduled run.
- **Http Timeout (in seconds):** The timeout to set on the HTTP connection after which the connection will be closed. This can be used to tune the response of the system when communication links are unreliable.
- **Verify SSL:** When HTTPS rather the HTTP is used this toggle allows for the verification of the certificate that is used. If a self signed certificate is used then this should not be enabled.

- Click *Next*
- Enable your task and click *Done*

## Header Fields

Header fields can be defined if required using the *Headers* configuration item. This is a JSON document that defines a set of key/value pairs for each header field. For example if a header field *token* was required with the value of *sfe93rjfk93rj* then the *Headers* JSON document would be as follows

```
{
  "token" : "sfe93rjfk93rj"
}
```

Multiple header fields may be set by specifying multiple key/value pairs in the JSON document.

## JSON Payload

The payload that is sent by this plugin is a simple JSON presentation of a set of reading values. A JSON array is sent with one or more reading objects contained within it. Each reading object consists of a timestamp, an asset name and a set of data points within that asset. The data points are represented as name value pair JSON properties within the reading property.

The fixed part of every reading contains the following

Name	Description
ts	The timestamp as an ASCII string in ISO 8601 extended format. If no time zone information is given it is assumed to indicate the use of UTC. This timestamp is added by Fledge when it first reads the data.
user_ts	The timestamp as an ASCII string in ISO 8601 extended format. If no time zone information is given it is assumed to indicate the use of UTC. This timestamp is added by the device itself and can be used to reflect the timestamp the data refers to rather than the timestamp Fledge read the data.
asset	The name of the asset this reading represents.
readings	A JSON object that contains the data points for this asset.

The content of the *readings* object is a set of JSON properties, each of which represents a data value. The type of these values may be integer, floating point, string, a JSON object or an array of floating point numbers.

A property



```
"voltage" : 239.4
```

would represent a numeric data value for the item *voltage* within the asset. Whereas

```
"voltageUnit" : "volts"
```

Is string data for that same asset. Other data may be presented as arrays

```
"acceleration" : [ 0.4, 0.8, 1.0 ]
```

would represent acceleration with the three components of the vector, x, y, and z. This may also be represented as an object

```
"acceleration" : { "X" : 0.4, "Y" : 0.8, "Z" : 1.0 }
```

both are valid formats within Fledge.

An example payload with a single reading would be as shown below

```
[
  {
    "user_ts"   : "2020-07-08 16:16:07.263657+00:00",
    "ts"        : "2020-07-08 16:16:07.263657+00:00",
    "asset"     : "motor1",
    "readings"  : {
      "voltage" : 239.4,
      "current" : 1003,
      "rpm"     : 120147
    }
  }
]
```

### Payload Script

If a script is given then it must provide a method called *convert*, that method is passed a single reading as a Python DICT and must return a formatted string payload for that reading.

As a simple example lets assume we want a JSON payload to be sent, but we want to use different keys to those in the default reading payload. We will replace *readings* with *data*, *user\_ts* with *when* and *asset* with *device*. A simple Python script to do this would be as follows;

```
import json
def convert(reading):
    newReading = {
        "data" : reading["readings"],
        "when" : reading["user_ts"],
        "device" : reading["asset"],
    }
    return json.dumps(newReading)
```

An HTTP request would be sent with one reading per request and that reading would be formatted as a JSON payload of the format

```
{
  "data":
  {
```

(continues on next page)



(continued from previous page)

```
        "sinusoid": 0.0,  
        "sine10": 10.0  
    },  
    "when": "2022-02-16 15:12:55.196494+00:00",  
    "device": "sinusoid"  
}
```

Note that white space and newlines have been added to improve the readability of the payload.

The above example returns a JSON format payload, the return may however not be encoded as JSON, for example an XML payload

```
from dict2xml import dict2xml  
def convert(reading):  
    newReading = {  
        "data" : reading["readings"],  
        "when" : reading["user_ts"],  
        "device" : reading["asset"],  
    }  
    payload = "<reading>" + dict2xml(newReading) + "</reading>"  
    return payload
```

This return XML format data as follows

```
<reading>  
  <data>  
    <sine10>10.0</sine10>  
    <sinusoid>0.0</sinusoid>  
  </data>  
  <device>sinusoid</device>  
  <when>2022-02-16 15:12:55.196494+00:00</when>  
</reading>
```

Note that white space and formatting have been added for ease of reading the XML data. You must also make sure you have installed the Python XML support as this is not normally installed with Fledge, To do this run

```
pip3 install dict2xml
```

from the command line of the Fledge machine.



## 8.2.7 Kafka Producer

The *fledge-north-kafka* plugin sends data from Fledge to the an Apache Kafka. Fledge acts as a Kafka producer, sending reading data to Kafka. This implementation is a simplified producer that sends all data on a single Kafka topic. Each message contains an asset name, timestamp and set of readings values as a JSON document.

The configuration of the *Kafka* plugin is very simple, consisting of four parameters that must be set.

The screenshot displays the 'Review Configuration' step of the Kafka plugin setup. At the top, a progress bar indicates the sequence: 1. Plugin & Name, 2. Review Configuration (highlighted), and 3. Done. The configuration form contains the following fields:

- Bootstrap Brokers:** A text input field containing 'localhost:9092,kafka.local:9092'.
- Kafka Topic:** A text input field containing 'Fledge'.
- Send JSON:** A dropdown menu set to 'Strings'.
- Data Source:** A dropdown menu set to 'readings'.

At the bottom of the form, there are 'Back' and 'Next' buttons.

- **Bootstrap Brokers:** A comma separate list of Kafka brokers to use to establish a connection to the Kafka system.
- **Kafka Topic:** The Kafka topic to which all data is sent.
- **Send JSON:** This controls how JSON data points should be sent to Kafka. These may be sent as strings or as JSON objects.
- **Data Source:** Which Fledge data to send to Kafka; Readings or Fledge Statistics.

## 8.2.8 ThingSpeak

The *fledge-north-thingspeak* plugin provides a mechanism to , allowing an easy route to send data from an Fledge environment into MATLAB.

In order to send data to ThingSpeak you must first create a channel to receive it.

- Login to your account
- From the menu bar select the *Channels* menu and the *My Channels* option



ThingSpeak™ Channels Apps Support Commercial Use How to Buy MR

## My Channels

[New Channel](#) Search by tag

Name	Created	Updated
sinusoid	2018-08-08	2019-07-26 15:04

[Private](#) [Public](#) [Settings](#) [Sharing](#) [API Keys](#) [Data Import / Export](#)

## Help

Collect data in a ThingSpeak channel from a device, from another channel, or from the web.

Click **New Channel** to create a new ThingSpeak channel.

Click on the column headers of the table to sort by the entries in that column or click on a tag to show channels with that tag.

Learn to [create channels](#), explore and transform data.

Learn more about [ThingSpeak Channels](#).

## Examples

- [Arduino](#)
- [Arduino MKR1000](#)
- [ESP8266](#)
- [Raspberry Pi](#)
- [Netduino Plus](#)

## Upgrade

Need to send more data faster?

Need to use ThingSpeak for a commercial project?

[Upgrade](#)

- Click on *New Channel* to create a new channel



[Channels](#)
[Apps](#)
[Support](#)

[Commercial Use](#)
[How to Buy](#)
[MR](#)

## New Channel

**Name**

**Description**

**Field 1**  ☒

**Field 2**  ☐

**Field 3**  ☐

**Field 4**  ☐

**Field 5**  ☐

**Field 6**  ☐

**Field 7**  ☐

**Field 8**  ☐

**Metadata**

**Tags** 

(Tags are comma separated)

**Link to External Site**

**Link to GitHub**

**Elevation**

**Show Channel Location** ☐

**Latitude**

**Longitude**

**Show Video** ☐

☒ YouTube
 ☐ Vimeo

**Video URL**

**Show Status** ☐

## Help

Channels store all the data that a ThingSpeak application collects. Each channel includes eight fields that can hold any type of data, plus three fields for location data and one for status data. Once you collect data in a channel, you can use ThingSpeak apps to analyze and visualize it.

### Channel Settings

- **Percentage complete:** Calculated based on data entered into the various fields of a channel. Enter the name, description, location, URL, video, and tags to complete your channel.
- **Channel Name:** Enter a unique name for the ThingSpeak channel.
- **Description:** Enter a description of the ThingSpeak channel.
- **Field#:** Check the box to enable the field, and enter a field name. Each ThingSpeak channel can have up to 8 fields.
- **Metadata:** Enter information about channel data, including JSON, XML, or CSV data.
- **Tags:** Enter keywords that identify the channel. Separate tags with commas.
- **Link to External Site:** If you have a website that contains information about your ThingSpeak channel, specify the URL.
- **Show Channel Location:**
  - **Latitude:** Specify the latitude position in decimal degrees. For example, the latitude of the city of London is 51.5072.
  - **Longitude:** Specify the longitude position in decimal degrees. For example, the longitude of the city of London is -0.1275.
  - **Elevation:** Specify the elevation position meters. For example, the elevation of the city of London is 35.052.
- **Video URL:** If you have a YouTube™ or Vimeo® video that displays your channel information, specify the full path of the video URL.
- **Link to GitHub:** If you store your ThingSpeak code on GitHub®, specify the GitHub repository URL.

### Using the Channel

You can get data into a channel from a device, website, or another ThingsSpeak channel. You can then visualize data and transform it using ThingSpeak [Apps](#).

See [Get Started with ThingSpeak™](#) for an example of measuring dew point from a weather station that acquires data from an Arduino® device.

[Learn More](#)

- Enter the details for your channel, in particular name and the set of fields. These field names should match the asset names you are going to send from Fledge.
- When satisfied click on *Save Channel*
- You will need the channel ID and the API key for your channel. To get this for a channel, on the *My Channels* page click on the *API Keys* box for your channel



**sinusoid**  
Channel ID: 556345 | Test channel  
Author: markdianomic  
Access: Private

Private View Public View Channel Settings Sharing API Keys Data Import / Export

### Write API Key

Key:

[Generate New Write API Key](#)

### Read API Keys

Key:

Note:

[Save Note](#) [Delete API Key](#)

[Add New Read API Key](#)

### Help

API keys enable you to write data to a channel or read data from a private channel. API keys are auto-generated when you create a new channel.

### API Keys Settings

- Write API Key:** Use this key to write data to a channel. If you feel your key has been compromised, click **Generate New Write API Key**.
- Read API Keys:** Use this key to allow other people to view your private channel feeds and charts. Click **Generate New Read API Key** to generate an additional read key for the channel.
- Note:** Use this field to enter information about channel read keys. For example, add notes to keep track of users with access to your channel.

### API Requests

**Write a Channel Feed**

```
GET https://api.thingspeak.com/update?api_key=APIKEY&field1=0
```

**Read a Channel Feed**

```
GET https://api.thingspeak.com/channels/556345/feeds.json?api_key=APIKEY&results=2
```

**Read a Channel Field**

```
GET https://api.thingspeak.com/channels/556345/fields/1.json?api_key=APIKEY&results=2
```

**Read Channel Status Updates**

```
GET https://api.thingspeak.com/channels/556345/status.json?api_key=APIKEY
```

[Learn More](#)

Once you have created your channel on you may create your north task on Fledge to send data to this channel

- Select *North* from the left hand menu bar.
- Click on the + icon in the top left
- Choose ThingSpeak from the plugin selection list
- Name your task
- Click on *Next*
- Configure the plugin



The screenshot shows the 'Review Configuration' step of the Fledge plugin setup. The configuration details are as follows:

- URL:** `https://api.thingspeak.com/channels`
- API Key:** (Empty field)
- Source:** `readings` (Selected from a dropdown)
- Fields:** A JSON document editor showing:
 

```

1 {
2   "elements": [
3     {
4       "asset": "sinusoid",
5       "reading": "sinusoid"
6     }
7   ]
8 }
      
```
- Channel ID:** `0`

Navigation buttons: `Back` and `Next`.

- **URL:** The URL of the ThingSpeak server, this can usually be left as the default.
- **API Key:** The write API key from the ThingSpeak channel you created
- **Source:** Controls if readings data or Fledge statistics are to be send to ThingSpeak
- **Fields:** Allows you to select what fields to send to ThingSpeak. It's a JSON document that contains a single array called elements. Each item of the array is a JSON object that has two properties, asset and reading. The asset should match the asset you wish to send and the reading the data point name.
- **Channel ID:** The channel ID of your ThingSpeak Channel

- Click on *Next*
- Enable your north task and click on *Done*

## 8.3 Fledge Filter Plugins

### 8.3.1 Asset Filter

The *fledge-filter-asset* is a filter that allows for assets to be included, excluded or renamed in a stream. It may be used either in *South* services or *North* tasks and is driven by a set of rules that define for each named asset what action should be taken.

Asset filters are added in the same way as any other filters.

- Click on the Applications add icon for your service or task.



- Select the *asset* plugin from the list of available plugins.
- Name your asset filter.
- Click *Next* and you will be presented with the following configuration page

1 Plugin Name 2 Review Configuration

**Asset rules**

```

1 {
2   "rules": [
3     {
4       "asset_name"    : "temperature",
5       "action"        : "rename",
6       "new_asset_name": "abient"
7     }
8   ]
9 }

```

Enabled ☒

Previous Done

- Enter the *Asset rules*
- Enable the plugin and click *Done* to activate it

## Asset Rules

The asset rules are an array of JSON objects which define the asset name to which the rule is applied and an action. Actions can be one of

- **include:** The asset should be forwarded to the output of the filter
- **exclude:** The asset should not be forwarded to the output of the filter
- **rename:** Change the name of the asset. In this case a third property is included in the rule object, "new\_asset\_name"

In addition a *defaultAction* may be included, however this is limited to *include* and *exclude*. Any asset that does not match a specific rule will have this default action applied to them. If the default action is not given it is treated as if a default action of *include* had been set.

A typical set of rules might be

```

{
  "rules": [
    {
      "asset_name": "Random1",

```

(continues on next page)



(continued from previous page)

```

        "action": "include"
    },
    {
        "asset_name": "Random2",
        "action": "rename",
        "new_asset_name": "Random92"
    },
    {
        "asset_name": "Random3",
        "action": "exclude"
    },
    {
        "asset_name": "Random4",
        "action": "rename",
        "new_asset_name": "Random94"
    },
    {
        "asset_name": "Random5",
        "action": "exclude"
    },
    {
        "asset_name": "Random6",
        "action": "rename",
        "new_asset_name": "Random96"
    },
    {
        "asset_name": "Random7",
        "action": "include"
    }
],
"defaultAction": "include"
}

```

### 8.3.2 Change Filter

The *fledge-filter-change* filter is used to only send information about an asset onward when a particular datapoint within that asset changes by more than a configured percentage. Data is sent for a period of time before and after the change in the monitored value. The amount of data to send before and after the change is configured in milliseconds, with a value for the pre-change time and one for the post-change time.

It is possible to define a rate at which readings should be sent regardless of the monitored value changing. This provides an average of the values of the period defined, e.g. send a 1 minute average of the values every minute.

This filter only operates on a single asset, all other assets are passed through the filter unaltered.

Change filters are added in the same way as any other filters.

- Click on the Applications add icon for your service or task.
- Select the *change* plugin from the list of available plugins.
- Name your change filter.
- Click *Next* and you will be presented with the following configuration page



The screenshot shows a configuration window with a progress bar at the top. Step 1 is 'Plugin Name' and Step 2 is 'Review Configuration'. The configuration form contains the following fields:

Field	Value
Asset	fan1
Trigger	current
Required Change %	5
Pre-trigger time (mS)	500
Post-trigger time (mS)	500
Reduced collection rate	2
Rate Units	per hour
Enabled	<input checked="" type="checkbox"/>

At the bottom, there are two buttons: 'Previous' and 'Done'.

- Enter the configuration for your change filter
  - **Asset:** The asset to monitor and control with this filter. This asset is both the asset that is used to look for changes and also the only asset whose data is affected by the triggered or non-triggered state of this filter.
  - **Trigger:** The datapoint within the asset that is used to trigger the sending of data at full rate. This datapoint may be either a numeric value or a string. If it is a string then a change of value of the defined change percentage or greater will trigger the sending of data. If the value is a string then any change in value will trigger the sending of the data.
  - **Required Change %:** The percentage change required for a numeric value change to trigger the sending of data. If this value is set to 0 then any change in the trigger value will be enough to trigger the sending of data.
  - **Pre-trigger time:** The number of milliseconds worth of data before the change that triggers the sending of data will be sent.
  - **Post-trigger time:** The number of milliseconds after a change that triggered the sending of data will be sent. If there is a subsequent change while the data is being sent then this period will be reset and the sending of data will recommence.
  - **Reduced collection rate:** The rate at which to send averages if a change does not trigger full rate data. This is defined as a number of averages for a period defined in the rateUnit, e.g. 4 per hour.
  - **Rate Units:** The unit associated with the average rate above. This may be one of “per second”, “per minute”, “per hour” or “per day”.
- Enable the change filter and click on *Done* to activate your plugin



### 8.3.3 Delta Filter

The *fledge-filter-delta* is a filter that only forwards data that changes by more than a configurable percentage. It is used to remove duplicate data values from an asset stream. The definition of duplicate however allows for some noise in the reading value by requiring a delta percentage.

By defining a minimum rate it is possible to force readings to be sent at that defined rate when there is no change in the value of the reading. Rates may be defined as per second, per minute, per hour or per day.

Delta filters are added in the same way as any other filters.

- Click on the Applications add icon for your service or task.
- Select the *delta* plugin from the list of available plugins.
- Name your delta filter.
- Click *Next* and you will be presented with the following configuration page

The screenshot shows the configuration interface for the Delta Filter plugin. At the top, a progress bar indicates two steps: '1 Plugin Name' and '2 Review Configuration'. The main configuration area is a light gray box with a help icon (?) in the top right corner. It contains the following fields:

- Tolerance %**: A text input field with the value '0'.
- Minimum Rate**: A text input field with the value '0'.
- Minimum Rate Units**: A dropdown menu currently set to 'per second'.
- Individual Tolerances**: A list box containing one entry with the value '1' and a green double curly brace icon.
- Enabled**: A checkbox that is currently unchecked.

At the bottom of the configuration box, there are two buttons: 'Previous' (disabled) and 'Done' (active).

- Configure the parameters of the delta filter
  - **Tolerance %**: The percentage tolerance when comparing reading data. Only values that differ by more than this percentage will be considered as different from each other.
  - **Minimum Rate**: The minimum rate at which readings should be sent. This is the rate at which readings will appear if there is no change in value.
  - **Minimum Rate Units**: The units in which minimum rate is define (per second, minute, hour or day)



- **Individual Tolerances:** A JSON document that can be used to define specific tolerance values for an asset. This is defines as a set of name/value pairs for those assets that should use a tolerance percentage other than the global tolerances specified above. The following example would set the tolerance for the temperature asset to 15% and for the pressure asset to 5%. All other assets would use the tolerance specified in *Tolerance %*.

```
{
  "temperature" : 15,
  "pressure" : 5
}
```

- Enable the filter and click *Done* to complete the process of adding the new filter.

### 8.3.4 Expression Filter

The *fledge-filter-expression* allows an arbitrary mathematical expression to be applied to data values. The expression filter allows user to augment the data at the edge to include values calculate from one or more asset to be added and acted upon both within the Fledge system itself, but also forwarded on to the up stream systems. Calculations can range from very simply manipulates of a single value to convert ranges, e.g. a linear scale to a logarithmic scale, or can combine multiple values to create composite value. E.g. create a power reading from voltage and current or work out a value that is normalized for speed.

Expression filters are added in the same way as any other filters.

- Click on the Applications add icon for your service or task.
- Select the *expression* plugin from the list of available plugins.
- Name your expression filter.
- Click *Next* and you will be presented with the following configuration page

The screenshot shows the 'Review Configuration' step of the Fledge Expression Filter setup. A progress bar at the top indicates the current step is 2 of 2. The configuration form contains the following fields:

- Datapoint Name:** A text input field containing the value 'calculated'.
- Expression to apply:** A text input field containing the value 'log(x)'.
- Enabled:** A checkbox that is checked, indicating the filter is active.

At the bottom of the form, there are two buttons: 'Previous' (disabled) and 'Done' (active).

- Configure the expression filter
  - **Datapoint Name:** The name of the new data point into which the new value will be stored.
  - **Expression to apply:** This is the expression that will be evaluated for each asset reading. The expression will use the data points within the reading as symbols within the asset. See *Expressions* below.
- Enable the plugin and click *Done* to activate your filter



### Expressions

The *fledge-filter-expression* plugin makes use of the library to do run time expression evaluation. This library provides a rich mathematical operator set, the most useful of these in the context of this plugin are;

- Logical operators (and, nand, nor, not, or, xor, xnor, mand, mor)
- Mathematical operators (+, -, \*, /, %, ^)
- Functions (min, max, avg, sum, abs, ceil, floor, round, roundn, exp, log, log10, logn, pow, root, sqrt, clamp, inrange, swap)
- Trigonometry (sin, cos, tan, acos, asin, atan, atan2, cosh, cot, csc, sec, sinh, tanh, d2r, r2d, d2g, g2d, hyp)

Within the expression the data points of the asset become symbols that may be used; therefore if an asset contains values “voltage” and “current” the expression will contain those as symbols and an expression of the form

```
voltage * current
```

can be used to determine the power in Watts.

When the filter is used in an environment in which more than one asset is passing through the filter then symbols are created of the form <asset name>.<data point>. As an example if you have one asset called “electrical” that has data points of “voltage” and “current” and another asset called “speed” that has a data point called “rpm” then you can write an expression to obtain the power per 1000 RPM’s of the motor as follows;

```
(electrical.voltage * electrical.current) / (speed.rpm / 1000)
```

### 8.3.5 Fast Fourier Transform Filter

The *fledge-filter-fft* filter is designed to accept some periodic data such as a sample electrical waveform, audio data or vibration data and perform a Fast Fourier Transform on that data to supply frequency data about that waveform.

Data is added as a new asset which is named as the sampled asset with “ FFT” append. This FFT asset contains a set of data points that each represent the a band of frequencies, or as a frequency spectrum in a single array data point. The band information that is returned by the filter can be chosen by the user. The options available to represent each band are;

- the average in the band,
- the peak
- the RMS
- or the sum of the band.

The bands are created by dividing the frequency space into a number of equal ranges after first applying a low and high frequency filter to discard a percentage of the low and high frequency results. The bands are not created if the user instead opts to return the frequency spectrum.

If the low Pass filter is set to 15% and the high Pass filter is set to 10%, with the number of bands set to 5, the lower 15% of results are discarded and the upper 10% are discarded. The remaining 75% of readings are then divided into 5 equal bands, each of which representing 15% of the original result space. The results within each of the 15% bands are then averaged to produce a result for the frequency band.

FFT filters are added in the same way as any other filters.

- Click on the Applications add icon for your service or task.
- Select the *fft* plugin from the list of available plugins.
- Name your FFT filter.



- Click *Next* and you will be presented with the following configuration page

Waveform South Service

1 Plugin Name 2 Review Configuration

Asset to analysis: wave

Result Data: average

Frequency Bands: 10

Band Prefix: Band

No. of samples per FFT: 8194

Low Frequency Reject %: 0

High Frequency Reject %: 0

Enabled: ☒

Previous Done

- Configure your FFT filter
  - Asset to analysis:** The name of the asset that will be used as the input to the FFT algorithm.

Result Data

Frequency Bands

Band Prefix

average (selected)

peak

sum

rms

spectrum

- Result Data:** The data that should be returned for each band. This may be one of average, sum, peak, rms or spectrum. Selecting average will return the average amplitude within the band, sum returns the sum of all amplitudes within the frequency band, peak the greatest amplitude and rms the root mean square of the amplitudes within the band. Setting the output type to be spectrum will result in the full FFT spectrum data being written. Spectrum data however can not be sent to all north destinations as it is not supported natively on all the systems Fledge can send data to.
- Frequency Bands:** The number of frequency bands to divide the resultant FFT output into
- Band Prefix:** The prefix to add to the data point names for each band in the output



- **No. of Samples per FFT**: The number of input samples to use. This must be a power of 2.
- **Low Frequency Reject %**: A percentage of low frequencies to discard, effectively reducing the range of frequencies to examine
- **High Frequency Reject %**: A percentage of high frequencies to discard, effectively reducing the range of frequencies to examine

### 8.3.6 Flir Validity Filter

The *fledge-filter-Flir-Validity* plugin is a simple filter that filters out unused boxes and spot temperatures in the Flir temperature data stream. The filter also allows the naming of the boxes such that the data points added to the asset will use these names rather than the default box1, box2 etc.

Adding the filter to a south plugin you will receive a configuration screen as below

AX8 South Service

1 Plugin Name 2 Review Configuration

**Area Labels**

```
{
  "areas": [
    "1",
    "2",
    "3",
    "4",
    "5",
    "6",
    "7",
    "8",
    "9",
    "10",
    "11",
    "12"
  ]
}
```

**Enabled** ☒

Previous Done

The JSON document *Area Labels* can be used to set the labels to use for each of the boxes and replace the min1, min2 etc. The value of this configuration option is a JSON document that has a single element called *areas* which is a JSON



array. Each element in that area is the name to assign to the particular box. The default values would set the name of box1 to simply be 1, box2 to 2 etc.

If we assume we are monitoring a lathe with the camera and taking the temperature of the motor, the bearing and cutting bit using the boxes 1, 2, and 3 in the camera. We wish to rename the first box to be called *Motor*, the second box to be called *Bearing* and the third to be called *Tool*, setting an *areas* array as follows would achieve this.

```
{
  "areas" : [
    "Motor",
    "Bearing",
    "Tool",
    "4",
    "5",
    "6",
    "7",
    "8",
    "9",
    "10"
  ]
}
```

Note that we do not change the boxes 4 to 10 as these are not in use and have not been defined within the area interface. Using the above configuration setting for areas will result in asset names of *minMotor*, *maxMotor* and *averageMotor* being generated for the motor temperature. Similarly the bearing temperatures would be *minBearing*, *maxBearing* and *averageBearing*. The tool would have asset names of *minTool*, *maxTool* and *averageTool*.

### 8.3.7 Log Filter

The *fledge-filter-log* plugin is a simple filter that converts data to a logarithmic scale.

When adding a scale filter to either the south service or north task, via the *Add Application* option of the user interface, a configuration page for the filter will be shown as below;

The screenshot displays the configuration page for the *fledge-filter-log* plugin. At the top, a progress bar indicates the current step is '2 Review Configuration', with '1 Plugin Name' being the previous step. The main configuration area contains two fields: 'Asset filter' with an empty text input box, and 'Enabled' with a checked checkbox. At the bottom, there are two buttons: 'Previous' and 'Done'.

The *Asset Filter* entry is a regular expression that can be used to limit the assets that the filter will effect. To change all assets leave this entry blank.



### 8.3.8 Metadata Filter

The *fledge-filter-metadata* filter allows data to be added to assets within Fledge. Metadata takes the form of fixed data points that are added to an asset used to add context to the data. Examples of metadata might be unit of measurement information, location information or identifiers for the piece of equipment to which the measurement relates.

A metadata filter may be added to either a south service or a north task. In a south service it will be adding data for just those assets that originate in that service, in which case it probably relates to a single machine that is being monitored and would add metadata related to that machine. In a north task it causes metadata to be added to all assets that the Fledge is sending to the up stream system, in which case the metadata would probably related to that particular Fledge instance. Adding metadata in the north is particularly useful when a hierarchy of Fledge systems is used and an audit trail is required with the data or the individual Fledge systems related to some physical location information such as building, floor and/or site.

To add a metadata filter

- Click on the Applications add icon for your service or task.
- Select the *metadata* plugin from the list of available plugins.
- Name your metadata filter.
- Click *Next* and you will be presented with the following configuration page

1 Plugin Name 2 Review Configuration

Metadata to add

```
1 {  
2   "floor": "Third",  
3   "location": "AirIntake",  
4   "unit": "Celsius",  
5   "serialNo": "A73953-42492-3229"  
6 }
```

Enabled ☒

- Enter your metadata in the JSON array shown. You may add multiple items in a single filter by separating them with commas. Each item takes the format of a JSON key/value pair and will be added as data points within the asset.
- Enable the filter and click on *Done* to activate it



## Example Metadata

Assume we are reading the temperature of air entering a paint booth. We might want to add the location of the paint booth, the booth number, the location of the sensor in the booth and the unit of measurement. We would add the following configuration value

```
{
  "value": {
    "floor": "Third",
    "booth": 1,
    "units": "C",
    "location": "AirIntake"
  }
}
```

In above example the filter would add “floor”, “booth”, “units” and “location” data points to all the readings processed by it. Given an input to the filter of

```
{ "temperature" : 23.4 }
```

The resultant reading that would be passed onward would become

```
{ "temperature" : 23.5, "booth" : 1, "units" : "C", "floor" : "Third", "location" :
  ↪ "AirIntake" }
```

This is an example of how metadata might be added in a south service. Turning to the north now, assume we have a configuration whereby we have several sites in an organization and each site has several building. We want to monitor data about the buildings and install a Fledge instance in each building to collect building data. We also install a Fledge instance in each site to collect the data from each individual Fledge instance per building, this allows us to then send the site data to the head office without having to allow each building Fledge to have access to the corporate network. Only the site Fledge needs that access. We want to label the data to say which building it came from and also which site. We can do this by adding metadata at each stage.

To the north task of a building Fledge, for example the “Pearson” building, we add the following metadata

```
{
  "value" : {
    "building": "Pearson"
  }
}
```

Likewise to the “Lawrence” building Fledge instance we add the following to the north task

```
{
  "value" : {
    "building": "Lawrence"
  }
}
```

These buildings are both in the “London” site and will send their data to the site Fledge instance. In this instance we have a north task that sends the data to the corporate headquarters, in this north task we add

```
{
  "value" : {
    "site": "London"
  }
}
```



If we assume we measure the power flow into each building in terms of current, and for the Pearson building we have a value of 117A at 11:02:15 and for the Lawrence building we have a value of 71.4A at 11:02:23, when the data is received at the corporate system we would see readings of

```
{ "current" : 117, "site" : "London", "building" : "Pearson" }  
{ "current" : 71.4, "site" : "London", "building" : "Lawrence" }
```

By adding the data like this it gives us more flexibility, if for example we want to change the way site names are reported, or we acquire a second site in London, we only have to change the metadata in one place.

### 8.3.9 OMF Hint Filter

The *fledge-filter-omfhint* filter allows hints to be added to assets within Fledge that will be used by the plugin. These hints allow for individual configuration of specific assets within the OMF plugin.

A OMF hint filter may be added to either a south service or a north task. In a south service it will be adding data for just those assets that originate in that service. In a north task it causes OMF hints to be added to all assets that the Fledge is sending to the upstream system, it would normally only be used in a north that was using the OMF plugin, however it could be used in a north that is sending data to another Fledge that then forwards to OMF.

To add an OMF hints filter:

- Click on the Applications add icon for your service or task.
- Select the *omfhint* plugin from the list of available plugins.
- Name your OMF hint filter.
- Click *Next* and you will be presented with the following configuration page



1 Plugin Name 2 Review Configuration

**OMF Hint**

```

1 {
2   "asset": {
3     "number": "float64"
4   }
5 }

```

Enabled ☐

Previous Done

- Enter your OMF Hints in the JSON editor shown. You may add multiple hints for multiple assets in a single filter instance. See *OMF Hint data*.
- Enable the filter and click on *Done* to activate it.

### OMF Hint data

OMF Hints comprise of an asset name which the hint applies and a JSON document that is the hint. A hint is a name/value pair: the name is the hint type and the value is the value of that hint.

The asset name may be expressed as a regular expression, in which case the hint is applied to all assets that match that regular expression.

The following hint types are currently supported by :

- *integer*: The format to use for integers, the value is a string and may be any of the PI Server supported formats; int64, int32, int16, uint64, uint32 or uint16
- *number*: The format to use for numbers, the value is a string and may be any of the PI Server supported formats; float64, float32 or float16
- *typeName*: Specify a particular type name that should be used by the plugin when it generates a type for the asset. The value of the hint is the name of the type to create.
- *tagName*: Specify a particular tag name that should be used by the plugin when it generates a tag for the asset. The value of the hint is the name of the tag to create.



- *type*: Specify a pre-existing type that should be used for the asset. In this case the value of the hint is the type to use. The type must already exist within your PI Server and must be compatible with the values within the asset.
- *datapoint*: Specifies that this hint applies to a single datapoint within the asset. The value is a JSON object that contains the name of the datapoint and one or more hints.
- *AFLocation*: Specifies a location in the OSIsoft Asset Framework for the asset. This hint is fully documented in the plugin page.

The following example shows a simple hint to set the number format to use for all numeric data within the asset names *supply*.

```
{
  "supply": {
    "number": "float32"
  }
}
```

To apply a hint to all assets, the single hint definition can be used with a regular expression.

```
{
  ".*": {
    "number": "float32"
  }
}
```

Regular expressions may also be used to select subsets of assets, in the following case only assets with the prefix OPCUA will have the hint applied.

```
{
  "OPCUA.*": {
    "number": "float32"
  }
}
```

To apply a hint to a particular data point the hint would be as follows

```
{
  "supply": {
    "datapoint": {
      "name": "frequency"
      "integer": "uint16"
    }
  }
}
```

This example sets the datapoint *frequency* within the *supply* asset to be stored in the PI server as a uint16.

Datapoint hints can be combined with asset hints

```
{
  "supply": {
    "number": "float32",
    "datapoint": {
      "name": "frequency"
      "integer": "uint16"
    }
  }
}
```

(continues on next page)



(continued from previous page)

```
}
}
```

In this case all numeric data except for *frequency* will be stored as float32 and *frequency* will be stored as uint16.

### 8.3.10 Python 2.7 Filter

The *fledge-filter-python27* filter allows snippets of Python to be easily written that can be used as filters in Fledge. A similar filter exists that uses Python 3.5 syntax, the *filter*. A Python code snippet will be called with sets of asset readings as they are read or processed in a filter pipeline. The data appears in the Python code as a JSON document passed as a Python Dict type.

The user should provide a Python function whose name matches the name given to the plugin when added to the filter pipeline of the south service or north task, e.g. if you name your filter *myPython* then you should have a function named *myPython* in the code you enter. This function is send a set of readings to process and should return a set of processed readings. The returned set of readings may be empty if the filter removes all data.

A general code syntax for the function that should be provided is;

```
def myPython(readings):
    for elem in list(readings):
        ...
    return readings
```

Each element that is processed has a number of attributes that may be accessed

Attribute	Description
asset_code	The name of the asset the reading data relates to.
timestamp	The data and time Fledge first read this data
user_timestamp	The data and time the data for the data itself, this may differ from the timestamp above
readings	The set of readings for the asset, this is itself an object that contains a number of key/value pairs that are the data points for this reading.

In order to access an data point within the readings, for example one named *temperature*, it is a simple case of extracting the value of with *temperature* as its key.

```
def myPython(readings):
    for elem in list(readings):
        reading = elem['readings']
        temp = reading['temperature']
        ...
    return readings
```

It is possible to write your Python code such that it does not know the data point names in advance, in which case you are able to iterate over the names as follows;

```
def myPython(readings):
    for elem in list(readings):
        reading = elem['readings']
        for attribute in reading:
            value = reading[attribute]
            ...
    return readings
```



A second function may be provided by the Python plugin code to accept configuration from the plugin that can be used to modify the behavior of the Python code without the need to change the code. The configuration is a JSON document which is again passed as a Python Dict to the `set_filter_config` function in the user provided Python code. This function should be of the form

```
def set_filter_config(configuration):
    config = json.loads(configuration['config'])
    value = config['key']
    ...
    return True
```

Python27 filters are added in the same way as any other filters.

- Click on the Applications add icon for your service or task.
- Select the *python27* plugin from the list of available plugins.
- Name your python27 filter, this should be the same name as the Python function you will provide.
- Click *Next* and you will be presented with the following configuration page

The screenshot shows the Fledge configuration interface for a python27 filter. At the top, there are two steps: 1. Plugin Name and 2. Review Configuration. The main area is divided into two sections: Python script and Configuration. The Python script section contains a code editor with the following code:

```
1 # generate exponential moving average
2
3 import json
4
5 # exponential moving average rate default value: include 7%
  of current value
6 rate = 0.07
7 # latest ema value
8 latest = None
9
10 # get configuration if provided.
11 # set this JSON string in configuration:
12 # {"rate":0.07}
```

Below the code editor is a file upload button labeled 'Choose Files' and a text box showing 'No file chosen'. The Configuration section contains a code editor with the following JSON configuration:

```
1 {"rate" : 0.75}
```

At the bottom left, there is a checkbox labeled 'Enabled'. At the bottom right, there are two buttons: 'Previous' and 'Done'.

- Enter the configuration for your python27 filter



- **Python script:** This is the script that will be executed. Initially you are unable to type in this area and must load your initial script from a file using the *Choose Files* button below the text area. Once a file has been chosen and loaded you are able to update the Python code in this page.

---

**Note:** Any changes made to the script in this screen will **not** be written back to the original file it was loaded from.

---

- **Configuration:** You may enter a JSON document here that will be passed to the *set\_filter\_config* function of your Python code.
- Enable the python27 filter and click on *Done* to activate your plugin

## Example

The following example uses Python to create an exponential moving average plugin. It adds a data point called *ema* to every asset. It assumes a single data point exists within the asset, but it does not assume the name of that data point. A rate can be set for the EMA using the configuration of the plugin.

```
# generate exponential moving average

import json

# exponential moving average rate default value: include 7% of current value
rate = 0.07
# latest ema value
latest = None

# get configuration if provided.
# set this JSON string in configuration:
# {"rate":0.07}
def set_filter_config(configuration):
    global rate
    config = json.loads(configuration['config'])
    if ('rate' in config):
        rate = config['rate']
    return True

# Process a reading
def doit(reading):
    global rate, latest

    for attribute in list(reading):
        if not latest:
            latest = reading[attribute]
        else:
            latest = reading[attribute] * rate + latest * (1 - rate)
            reading[b'ema'] = latest

# process one or more readings
def ema(readings):
    for elem in list(readings):
        doit(elem['reading'])
    return readings
```

Examining the content of the Python, a few things to note are;



- The filter is given the name `ema`. This name defines the default method which will be executed, namely `ema()`.
- The function `ema` is passed 1 or more readings to process. It splits these into individual readings, and calls the function `doit` to perform the actual work.
- The function `doit` walks through each attribute in that reading, updates a global variable `latest` with the latest value of the `ema`. It then adds an `ema` attribute to the reading.
- The function `ema` returns the modified readings list which then is passed to the next filter in the pipeline.
- `set_filter_config()` is called whenever the user changes the JSON configuration in the plugin. This function will alter the global variable `rate` that is used within the function `doit`.

### 8.3.11 Python 3.5 Filter

The *fledge-filter-python35* filter allows snippets of Python to be easily written that can be used as filters in Fledge. A similar filter exists that uses Python 2.7 syntax, the filter, however it is recommended that the *python35* filter is used by preference as it provides a more compatible interface to the rest of the Fledge components.

The purpose of the filter is to allow the user the flexibility to add their own processing into a pipeline that is not supported by any of the existing filters offered by Fledge. The philosophy of Fledge is to provide processing by adding a set of filters that act as a pipeline between a data source and and data sink. The data source may be the south plugin in the south service or the buffered readings data from the storage service in the case of a north service or task. The data sink is either the buffer within the storage service in the case of a south service or the north plugin that sends the data to the upstream system in the case of a north service or task. Each of the filters provides a small, focused operation on the data as it traverses the pipeline, data is passed from one filter in the pipeline to another.

The functionality provided by this filters gives the user the opportunity to write Python code that can manipulate the reading data as it flows, however that modification should follow the same guidelines and principles as followed in the filters that come supplied as part of the Fledge distribution or are contributed by other Fledge users. The overriding principles however are

- Do not duplicate existing functionality provided by existing filters.
- Keep the operations small and focused. It is better to have multiple filters each with a specific purpose than to create large, complex Python scripts.
- Do not buffer large quantities of data, this will effect the footprint of the service and also slow the data pipeline.

The Python code snippet that the user provides within this filter will be called with sets of asset readings as they or read or processed in a filter pipeline. The data appears in the Python code as a JSON document passed as a Python Dict type.

The user should provide a Python function whose name matches the name given to the plugin when added to the filter pipeline of the south service or north task, e.g. if you name your filter `myPython` then you should have a function named `myPython` in the code you enter. This function is passed a set of readings to process and should return a set of processed readings. The returned set of readings may be empty if the filter removes all data.

A general code syntax for the function that should be provided is as follows;

```
def myPython(readings):
    for elem in list(readings):
        ...
    return readings
```

The script iterates over the set of readings it is passed from the previous filter or the south plugin and returns the result of processing that data. Multiple readings are passed for two reasons, one is to improve the efficiency of processing and the second is because a south service or filter may ingest or process multiple readings in a single operation.

Each element that is processed has a number of attributes that may be accessed



Attribute	Description
asset_code	The name of the asset the reading data relates to.
timestamp	The data and time Fledge first read this data
user_timestamp	The data and time the data for the data itself, this may differ from the timestamp above
readings	The set of readings for the asset, this is itself an object that contains a number of key/value pairs that are the data points for this reading.

In order to access an data point within the readings, for example one named *temperature*, it is a simple case of extracting the value of with *temperature* as its key.

```
def myPython(readings):
    for elem in list(readings):
        reading = elem['readings']
        temp = reading['temperature']
        ...
    return readings
```

It is possible to write your Python code such that it does not know the data point names in advance, in which case you are able to iterate over the names as follows;

```
def myPython(readings):
    for elem in list(readings):
        reading = elem['readings']
        for attribute in reading:
            value = reading[attribute]
            ...
    return readings
```

The Python script is not limited to returning the same number of readings it receives, additional readings may be added into the pipeline or readings may be removed. If the filter removes all the readings it was sent it must still return either an empty list or it may return the *None* object.

A second function may be provided by the Python plugin code to accept configuration from the plugin that can be used to modify the behavior of the Python code without the need to change the code. The configuration is a JSON document which is again passed as a Python Dict to the `set_filter_config` function in the user provided Python code. This function should be of the form

```
def set_filter_config(configuration):
    config = json.loads(configuration['config'])
    value = config['key']
    ...
    return True
```

This function is called each time the configuration of the filter is changed. The function is responsible for taking whatever actions are required to change the behavior of the Python script. The most common approach taken with the configuration function is to record the configuration information in global variables for reference by the Python script. This however is contrary to the recommendations for writing Python scripts that are embedded within plugins.

There is little choice but to use globals in this case, however precautions should be taken than minimize the risk of sharing common global variables between instances.

- Do not use common names or names that are not descriptive. E.g. avoid simply calling the variable *config*.
- Do not use multiple variables, there are other options that can be used.
  - Use a single Python DICT as reference individuals items within the DICT



- Create a Python class and use a global instance of the class

### Adding Python35 Filters

Python35 filters are added in the same way as any other filters.

- Click on the Applications add icon for your service or task.
- Select the *python35* plugin from the list of available plugins.
- Name your python35 filter, this should be the same name as the Python function you will provide.
- Click *Next* and you will be presented with the following configuration page

1 Plugin Name 2 Review Configuration

**Python script**

```

1  # generate exponential moving average
2
3  import json
4
5  # exponential moving average rate default value: include 7%
6  # of current value
7  rate = 0.07
8  # latest ema value
9  latest = None
10
11 # get configuration if provided.
12 # set this JSON string in configuration:
13 # {"rate":0.07}

```

ema.py

Choose Files No file chosen

**Configuration**

```

1  {"rate" : 0.75}

```

Enabled ☐

Previous Done

- Enter the configuration for your python35 filter
  - **Python script:** This is the script that will be executed. Initially you are unable to type in this area and must load your initial script from a file using the *Choose Files* button below the text area. Once a file has been chosen and loaded you are able to update the Python code in this page.

**Note:** Any changes made to the script in this screen will **not** be written back to the original file it was



loaded from.

- **Configuration:** You may enter a JSON document here that will be passed to the `set_filter_config` function of your Python code.

- Enable the python35 filter and click on *Done* to activate your plugin

## Example

The following example uses Python to create an exponential moving average plugin. It adds a data point called *ema* to every asset. It assumes a single data point exists within the asset, but it does not assume the name of that data point. A rate can be set for the EMA using the configuration of the plugin.

```
# generate exponential moving average

import json

# exponential moving average rate default value: include 7% of current value
rate = 0.07
# latest ema value
latest = None

# get configuration if provided.
# set this JSON string in configuration:
# {"rate":0.07}
def set_filter_config(configuration):
    global rate
    config = json.loads(configuration['config'])
    if ('rate' in config):
        rate = config['rate']
    return True

# Process a reading
def doit(reading):
    global rate, latest

    for attribute in list(reading):
        if not latest:
            latest = reading[attribute]
        else:
            latest = reading[attribute] * rate + latest * (1 - rate)
            reading[b'ema'] = latest

# process one or more readings
def ema(readings):
    for elem in list(readings):
        doit(elem['reading'])
    return readings
```

Examining the content of the Python, a few things to note are;

- The filter is given the name *ema*. This name defines the default method which will be executed, namely *ema()*.
- The function *ema* is passed 1 or more readings to process. It splits these into individual readings, and calls the function *doit* to perform the actual work.
- The function *doit* walks through each attribute in that reading, updates a global variable *latest* with the latest value of the *ema*. It then adds an *ema* attribute to the reading.



- The function `ema` returns the modified readings list which then is passed to the next filter in the pipeline.
- `set_filter_config()` is called whenever the user changes the JSON configuration in the plugin. This function will alter the global variable `rate` that is used within the function `doit`.

### Scripting Guidelines

The user has the full range of Python functionality available to them within the script code they provide to this filter, however caution should be exercised as it is possible to adversely impact the functionality and performance of the Fledge system by misusing Python features to the detriment of Fledge's own features.

The overriding guidance given above should always be observed

- Do not duplicate existing functionality provided by existing filters.
- Keep the operations small and focused. It is better to have multiple filters each with a specific purpose than to create large, complex Python scripts.
- Do not buffer large quantities of data, this will effect the footprint of the service and also slow the data pipeline.

### Importing Python Packages

The user is free to import whatever packages they wish in a Python script, this includes the likes of the numpy packages and other that are limited to a single instance within a Python interpreter.

Do not import packages that you do not use or are not required. This adds an extra overhead to the filter and can impact the performance of Fledge. Only import packages you actually need.

Python does not provide a mechanism to remove a package that has previously been imported, therefore if you import a package in your script and then update your script to no longer import the package, the package will still be in memory from the previous import. This is because we reload updated scripts without closing down as restarting the Python interpreter. This is part of the sharing of the interpreter that is needed in order to allow packages such as numpy and scipy to be used. This can lead to misleading behavior as when the service gets restarted the package will not be loaded and the script may break because it makes use of the package still.

If you remove a package import from your script and you want to be completely satisfied that the script will still run without it, then you must restart the service in which you are using the plugin. This can be done by disabling and then re-enabling the service.

### Use of Global Variables

You may use global variables within your script and these globals will retain their value between invocations of the processing function. You may use global variables as a method to keep information between executions and perform such operations as trend analysis based on data seen in previous calls to the filter function.

All Python code within a single service shares the same Python interpreter and hence they also share the same set of global variables. This means you must be careful as to how you name global variables and also if you need to have multiple instances of the same filter in a single pipeline you must be aware that the global variables will be shared between them. If your filter uses global variables it is normally not recommended to have multiple instances of them in the same pipeline.

It is tempting to use this sharing of global variables as a method to share information between filters, this is not recommended as should not be used. There are several reasons for this

- It provides data coupling between filters, each filter should be independent of each other filter.
- One of the filters sharing global variables may be disabled by the user with unexpected consequences.



- Filter order may be changed, resulting in data that is expected by a later filter in the chain not being available.
- Intervening filters may add or remove readings resulting in the data in the global variables not referring to the same reading, or set of readings that it was intended to reference.

If you no wish one filter to pass data onto a later filter in the pipeline this is best done by adding data to the reading, as an extra data point. This data point can then be removed by the later filter. An example of this is the way Fledge adds OMF hints to readings that are processed and removed by the OMF north plugin.

For example let us assume we have calculated some value *delta* that we wish to pass to a later filter, we can add this as a data point to our reading which we will call *\_hintDelta*.

```
def myPython(readings):
    for elem in list(readings):
        reading = elem['readings']
        ...
        reading['_hintDelta'] = delta
        ...
    return readings
```

This is far better than using a global as it is attached to the reading to which it refers and will remain attached to that reading until it is removed. It also means that it is independent of the number of readings that are processed per call, and resilient to readings being added or removed from the stream.

The name chosen for this data point in the example above has no significance, however it is good practice to choose a name that is unlikely to occur in the data normally and portrays the usage or meaning of the data.

## File IO Operations

It is possible to make use of file operations within a Python35 filter function, however it is not recommended for production use for the following reasons;

- Pipelines may be moved to other hosts where files may not be accessible.
- Permissions may change dependent upon how Fledge systems are deployed in the various different scenarios.
- Edge devices may also not have large, high performance storage available, resulting in performance issues for Fledge or failure due to lack of space.
- Fledge is designed to be managed solely via the Fledge API and applications that use the API. There is no facility within that API to manage arbitrary files within the filesystem.

It is common to make use of files during development of a script to write information to in order to aid development and debugging, however this should be removed, along with associated imports of packages required to perform the file IO, when a filter is put into production.

## Threads within Python

It is tempting to use threads within Python to perform background activity or to allow processing of data sets in parallel, however there is an issue with threading in Python, the Python Global Interpreter Lock or GIL. The GIL prevents two Python statements from being executed within the same interpreter by two threads simultaneously. Because we use a single interpreter for all Python code running in each service within Fledge, if a Python thread is created that performs CPU intensive work within it, we block all other Python code from running within that Fledge service.

We therefore avoid using Python threads within Fledge as a means to run CPU intensive tasks, only using Python threads to perform IO intensive tasks, using the asyncio mechanism of Python 3.5.3 or later. In older versions of Fledge we used multiple interpreters, one per filter, in order to workaround this issue, however that had the side effect that a number of popular Python packages, such as *numpy*, *pandas* and *scipy*, could not be used as they can not support



multiple interpreters within the same address space. It was decided that the need to use these packages was greater than the need to support multiple interpreters and hence we have a single interpreter per service in order to allow the use of these packages.

### Interaction with External Systems

Interaction with external systems, using network connections or any form of blocking communication should be avoided in a filter. Any blocking operation will cause data to be blocked in the pipeline and risks either large queues of data accumulating in the case of asynchronous south plugins or data being missed in the case of polled plugins.

### Scripting Errors

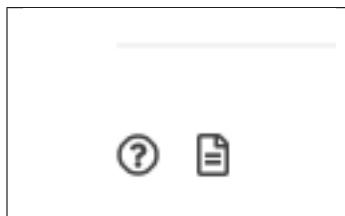
If an error occurs in the plugin or Python script, including script coding errors and Python exception, details will be logged to the error log and data will not flow through the pipeline to the next filter or into the storage service.

Warnings raised will also be logged to the error log but will not cause data to cease flowing through the pipeline.

To view the error log you may examine the file directly on your host machine, for example `/var/log/syslog` on a Ubuntu host, however it is also possible to view the error logs specific to Fledge from the Fledge user interface. Select the *System* option under *Logs* in the left hand menu pane. You may then filter the logs for a specific service to see only those logs that refer to the service which uses the filter you are interested in.

The screenshot shows the Fledge user interface. On the left, a sidebar menu has 'System' selected under the 'Logs' section. The main area is titled 'System Logs' and features a table of log entries. The table has columns for 'Service' (set to 'Simple'), 'Severity' (set to 'Info and above'), and a search bar. The log entries are dated 'Jul 7' and show various error messages, such as 'The supplied Python script does not define a valid "convert" function' and 'Failed to register service Simple'.

Alternatively if you open the dialog for the service in the *South* or *North* menu items you will see two icons displayed in the bottom left corner of the dialog that lets you alter the configuration of the service.





The left most icon, with the ? in a circle, allows you to view the documentation for the plugin, the right most icon, which looks like a page of text with a corner folded over, will open the log view page filtered to view the service.

## Error Messages & Warnings

The following are some errors you may see within the log with some description of the cause and remedy for the error.

**Unable to obtain a reference to the asset tracker. Changes will not be tracked** The service is unable to obtain the required reference to the asset tracker within Fledge. Data will continue to flow through the pipeline, but there will not be any trace of the assets that have been modified by this plugin within the pipeline.

**The return type of the python35 filter function should be a list of readings.** The python script has returned an incorrect data type. The return value of the script should be a list of readings

**Badly formed reading in list returned by the Python script** One or more of the readings in the list returned by the Python script is an incorrectly formed reading object.

**Each element returned by the script must be a Python DICT** The list returned by the Python script contains an element that is not a DICT and therefore can not be a valid reading.

**Badly formed reading in list returned by the Python script: Reading has no asset code element** One or more of the readings that is returned in the list from the script is missing the *asset\_code* key. This item is the name of the asset to which the reading refers.

**Badly formed reading in list returned by the Python script: Reading is missing the reading element which should contain the data** One or more of the readings that is returned in the list from the script is missing the *reading* DICT that contains the actual data.

**Badly formed reading in list returned by the Python script: The reading element in the python Reading is of an incorrect type, i** One or more of the readings that is returned in the list from the script has an item with a key of *reading* which is not a Python Dict. This item should always be a DICT and contains the data values as key/value pairs.

**Badly formed reading in list returned by the Python script: Unable to parse the asset code value. Asset codes should be a string** One or more of the readings that is returned in the list from the script has an item with a key of *asset\_code* whose value is not a string.

### 8.3.12 Rate Filter

The *fledge-filter-rate* plugin that can be used to reduce the rate a reading is stored until an interesting event occurs. The filter will read data at full rate from the input side and buffer data internally, sending out averages for each value over a time frame determined by the filter configuration.

The user can provide either one or two simple expressions that will be evaluated to form a trigger for the filter. One expressions will set the trigger and the other will clear it. When the trigger is set then the filter will no longer average the data over the configured time period, but will instead send the full bandwidth data out of the filter. If the second expression, the one that clears the full rate sending of data is omitted then the full rate is cleared as soon as the trigger expression returns false. Alternatively the filter can be configured to clear the sending of full rate data after a fixed time.

The filter also allows a pre-trigger time to be configured. In this case it will buffer this much data internally and when the trigger is initially set this pre-buffered data will be sent. The pre-buffered data is discarded if the trigger is not set and the data gets to the defined age for holding pre-trigger information.

Rate filters are added in the same way as any other filters.

- Click on the Applications add icon for your service or task.
- Select the *rate* plugin from the list of available plugins.



- Name your rate filter.
- Click *Next* and you will be presented with the following configuration page

1 Plugin Name

2 Review Configuration

Trigger expression:

Terminate on:

End Expression:

Full rate time (mS):

Pre-trigger time (mS):

Reduced collection rate:

Rate Units:

Exclusions: 

```
1 {
2   "exclusions": [ "speed" ]
3 }
```

Enabled: ☒

- Configure your rate filter
  - **Trigger Expression:** An expression to set the trigger for full rate data
  - **Terminate ON:** The mechanism to stop full rate forwarding, this may be another expression or a time window

Terminate on

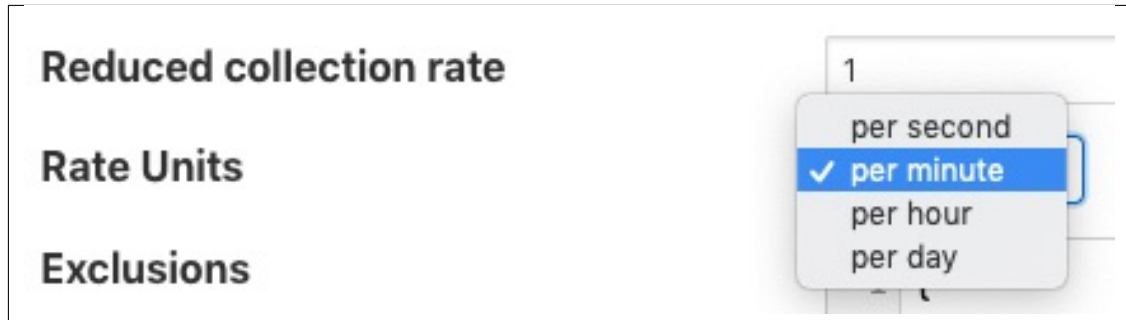
Expression

Time

- **End Expression:** An expression to clear the trigger for full rate data, if left blank this will simply be the trigger filter evaluating to false
- **Full rate time (ms):** The time window, in milliseconds to forward data at the full rate



- **Pre-trigger time (ms):** An optional pre-trigger time expressed in milliseconds
- **Reduced collection rate:** The nominal data rate to send data out. This defines the period over which is outgoing data item is averaged.
- **Rate Units:** The units that the reduced collection rate is expressed in; per second, minute, hour or day



- **Exclusions:** A set of asset names that are excluded from the rate limit processing and always sent at full rate

- Enable your filter and click *Done*

For example if the filter is working with a SensorTag and it reads the tag data at 10ms intervals but we only wish to send 1 second averages under normal circumstances. However if the X axis acceleration exceed 1.5g then we want to send full bandwidth data until the X axis acceleration drops to less than 0.2g, and we also want to see the data for the 1 second before the acceleration hit this peak the configuration might be:

- **Nominal Data Rate:** 1, data rate unit “per second”
- **Trigger set expression:**  $X > 1.5$
- **Trigger clear expression:**  $X < 0.2$
- **Pre-trigger time (mS):** 1000

The trigger expression uses the same expression mechanism, as the , and plugins

Expression may contain any of the following. . .

- Mathematical operators (+, -, \*, /, %, ^)
- Functions (min, max, avg, sum, abs, ceil, floor, round, roundn, exp, log, log10, logn, pow, root, sqrt, clamp, inrange, swap)
- Trigonometry (sin, cos, tan, acos, asin, atan, atan2, cosh, cot, csc, sec, sinh, tanh, d2r, r2d, d2g, g2d, hyp)
- Equalities & Inequalities (=, ==, <>, !=, <, <=, >, >=)
- Logical operators (and, nand, nor, not, or, xor, xnor, mand, mor)

**Note:** This plugin is designed to work with streams with a single asset in the stream, there is no mechanism in the expression syntax to support multiple asset names.



### 8.3.13 Rename Filter

The *fledge-filter-rename* filter that can be used to modify the name of an asset, datapoint or both. It may be used either in *South* services or *North* services or *North* tasks.

To add a Rename filter

- Click on the Applications add icon for your service or task.
- Select the *rename* plugin from the list of available plugins.
- Name your Rename filter.
- Click *Next* and you will be presented with the following configuration page
- Configure the plugin

- **Operation:** Search and replace operation be performed on asset name, datapoint name or both
- **Find:** A regular expression to match for the given operation
- **Replace With:** A substitution string to replace the matched text with
- Enable the filter and click on *Done* to activate it

#### Example

The simplest following example perform on given below reading object

```
{
  "readings": {
    "sinusoid": -0.978147601,
    "a": {
      "sinusoid": "2.0"
    }
  },
  "asset": "sinusoid",
  "id": "a1bedea3-8d80-47e8-b256-63370ccf5b",
  "ts": "2021-06-28 14:03:22.106562+00:00",
}
```

(continues on next page)



(continued from previous page)

```
{
  "user_ts": "2021-06-28 14:03:22.106435+00:00"
}
```

1. To replace an asset apply a configuration would be as follows

- Operation : asset
- Find : sinusoid
- Replace With : sin

#### Output

```
{
  "readings": {
    "sinusoid": -0.978147601,
    "a": {
      "sinusoid": 2.0
    }
  },
  "asset": "sin",
  "id": "albedea3-8d80-47e8-b256-63370ccfce5b",
  "ts": "2021-06-28 14:03:22.106562+00:00",
  "user_ts": "2021-06-28 14:03:22.106435+00:00"
}
```

2. To replace a datapoint apply a configuration would be as follows

- Operation : datapoint
- Find : sinusoid
- Replace With : sin

#### Output

```
{
  "readings": {
    "sin": -0.978147601,
    "a": {
      "sin": 2.0
    }
  },
  "asset": "sinusoid",
  "id": "albedea3-8d80-47e8-b256-63370ccfce5b",
  "ts": "2021-06-28 14:03:22.106562+00:00",
  "user_ts": "2021-06-28 14:03:22.106435+00:00"
}
```

3. To replace both asset and datapoint apply a configuration would be as follows

- Operation : both
- Find : sinusoid
- Replace With : sin

#### Output



```
{
  "readings": {
    "sin": -0.978147601,
    "a": {
      "sin": 2.0
    }
  },
  "asset": "sin",
  "id": "albedea3-8d80-47e8-b256-63370ccfce5b",
  "ts": "2021-06-28 14:03:22.106562+00:00",
  "user_ts": "2021-06-28 14:03:22.106435+00:00"
}
```

### 8.3.14 Replace Filter

The *fledge-filter-replace* is a filter that allows an be used to replace all occurrence of a set of characters with a single replacement character. This can be used to change reserved characters in the names of assets and datapoints.

The screenshot displays the configuration interface for the 'Replace' filter. At the top, a progress bar indicates two steps: '1 Plugin Name' and '2 Review Configuration'. The 'Review Configuration' step is active, showing a configuration box with the following fields:

- Replace:** A text input field containing the regular expression '\A?'. A help icon (?) is visible to the right of the field.
- With:** A text input field containing the character '-'.
- Enabled:** A checkbox that is currently unchecked.

At the bottom of the configuration box, there are two buttons: 'Previous' (disabled) and 'Done' (active).

- **Replace:** The set of reserved characters to be replaced.
- **With:** The character to replace each occurrence of the above characters with

### 8.3.15 Root Mean Squared (RMS) Filter

The *fledge-filter-rms* filter is designed to accept some periodic data such as a sample electrical waveform, audio data or vibration data and perform a Root Mean Squared, *RMS* operation on that data to supply power of the waveform. The filter can also return the *peak to peak* amplitude of the waveform over the sampled period and the *crest* factor of the waveform.

**Note:** peak values may be less than individual values of the input if the asset value does not fall to or below zero. Where a data value swings between negative and positive values then the peak value will be greater than the maximum value in the data stream. For example if the minimum value of a data point in the sample set is 0.3 and the maximum



is 3.4 then the peak value will be 3.1. If the maximum value is 2.4 and the minimum is zero then the peak will be 2.4. If the maximum value is 1.7 and the minimum is -0.5 then the peak value will be 2.2.

RMS, also known as the quadratic mean, is defined as the square root of the mean square (the arithmetic mean of the squares of a set of numbers).

Peak to peak, is the difference between the smallest value in the sampled data and the highest, this give the maximum amplitude variation during the period sampled.

Crest factor is a parameter of a waveform, showing the ratio of peak values to the effective value. In other words, crest factor indicates how extreme the peaks are in a waveform. Crest factor 1 indicates no peaks, such as direct current or a square wave. Higher crest factors indicate peaks, for example sound waves tend to have high crest factors.

The user may also choose to include or not the raw data that is used to calculate the RMS values via a switch in the configuration.

Where a data stream has multiple assets within it the RMS filter may be limited to work only on those assets whose name matches a regular expression given in the configuration of the filter. The default for this expression is `.*`, i.e. all assets are processed.

RMS filters are added in the same way as any other filters.

- Click on the Applications add icon for your service or task.
- Select the *rms* plugin from the list of available plugins.
- Name your RMS filter.
- Click *Next* and you will be presented with the following configuration page

Waveform South Service

1 Plugin Name 2 Review Configuration

Sample size: 10

RMS Asset name: %a RMS

Include Peak Values: ☐

Include Crest Values: ☐

Include Raw Data: ☐

Asset filter: .\*

Enabled: ☐

Previous Done

- Configure your RMS filter
  - **Sample size:** The number of data samples to perform a calculation over.



- **RMS Asset name:** The asset name to use to output the RMS values. “%a” will be replaced with the original asset name.
- **Include Peak Values:** A switch to include peak to peak measurements for the same data set as the RMS measurement.
- **Include Crest Values:** A switch to include crest measurements for the same data set as the RMS measurement.
- **Include Raw Data:** A switch to include the raw input data in the output.
- **Asset Filter:** A regular expression to limit the asset names on which this filter operations. Useful when multiple assets appear in the input data stream as it allows data which is not part of the periodic function that is being examined to be excluded.

### 8.3.16 Scale Filter

The *fledge-filter-scale* plugin is a simple filter that allows a scale factor and an offset to be applied to numerical data. It's primary uses are for adjusting values to match different measurement scales, for example converting temperatures from Centigrade to Fahrenheit or when a sensor reports a value in non-base units, e.g. 1/10th of a degree.

When adding a scale filter to either the south service or north task, via the *Add Application* option of the user interface, a configuration page for the filter will be shown as below;

The screenshot shows a configuration window titled "Sine South Service". At the top, there is a progress bar with two steps: "1 Plugin Name" and "2 Review Configuration". The "Review Configuration" step is currently active. Below the progress bar, there is a form with the following fields:

- Scale Factor:** A text input field containing the value "100.0".
- Constant Offset:** A text input field containing the value "0.0".
- Asset filter:** A text input field that is currently empty.
- Enabled:** A checkbox that is checked, indicated by a blue checkmark icon.

At the bottom of the window, there are two buttons: "Previous" (disabled) and "Done" (active).

The configuration options supported by the scale filter are detailed in the table below



Setting	Description
Scale Factor	The scale factor to multiply the numeric values by
Constant Offset	A constant to add to all numeric values after applying the scale
Asset filter	This is useful when applying the filter in the north, it allows the filter to be applied only to those assets that match the regular expression given. If left blank then the filter is applied to all assets/

### 8.3.17 Scale Set Filter

The *fledge-filter-scale-set* plugin is a filter that allows a scale factor and an offset to be applied to numerical data where an asset has multiple data points. It is very similar to the filter, which allows a single scale and offset to be applied to all assets and data points. It's primary uses are for adjusting values to match different measurement scales, for example converting temperatures from Centigrade to Fahrenheit or when a sensor reports a value in non-base units, e.g. 1/10th of a degree.

Scale set filters are added in the same way as any other filters.

- Click on the Applications add icon for your service or task.
- Select the *scale-set* plugin from the list of available plugins.
- Name your scale-set filter.
- Click *Next* and you will be presented with the following configuration page

1 Plugin Name 2 Review Configuration

Scale factors

```

1 {
2   "factors": [
3     {
4       "asset": ".*",
5       "datapoint": ".*",
6       "scale": 1,
7       "offset": 0
8     }
9   ]
10 }

```

Enabled ☐

Previous Done

- Enter the configuration for your change filter



- **Scale factors:** A JSON document that defines a set of factors to apply. It is an array of JSON objects that define the scale factor and offset, a regular expression that is matched against the asset name and another that matches the data point name within the asset.

Name	Description
asset	A regular expression to match against the asset name. The scale factor is only applied to assets whose name matches this regular expression.
data-point	A regular expression to match against the data point name within a matching asset. The scale factor is only applied to assets whose name matches this regular expression.
scale	The scale factor to apply to the numeric data.
off-set	The offset to add to the matching numeric data.

- Enable the scale-set filter and click on *Done* to activate your plugin

### Example

In the following example we have an asset whose name is *environment* which contains two data points; *temperature* and *humidity*. We wish to allow two different scale factors and offsets to these two data points whilst not affecting assets of any other name in the data stream. We can accomplish this by using the following JSON document in the plugin configuration;

```
{
  "factors" : [
    {
      "asset"      : "environment",
      "datapoint"  : "temperature",
      "scale"      : 1.8,
      "offset"     : 32
    },
    {
      "asset"      : "environment",
      "datapoint"  : "humidity",
      "scale"      : 0.1,
      "offset"     : 0
    }
  ]
}
```

If instead we had multiple assets that contain *temperature* and *humidity* we can accomplish the same transformation on all these assets, whilst not affecting any other assets, by changing the *asset* regular expression to something that matches more asset names;

```
{
  "factors" : [
    {
      "asset"      : ".*",
      "datapoint"  : "temperature",
      "scale"      : 1.8,
      "offset"     : 32
    },
    {
      "asset"      : ".*",
      "datapoint"  : "humidity",
      "scale"      : 0.1,

```

(continues on next page)



(continued from previous page)

```
        "offset"      : 0
    }
  ]
}
```

### 8.3.18 Threshold Filter

The *fledge-filter-threshold* plugin is a filter that is used to control the forwarding of data within Fledge. Its use is to only allow data to be stored or forwarded if a condition about that data is true. This can save storage or network bandwidth by eliminating data that is of no interest.

The filter uses an expression, that is entered by the user, to evaluate if data should be forwarded, if that expression evaluates to true then the data is forwarded, in the case of a south service this would be to the Fledge storage. In the case of a north task this would be to the upstream system.

---

**Note:** If the threshold filter is part of a chain of filters and the data is not forwarded by the threshold filter, i.e. the expression evaluates to false, then the following filters will not receive the data.

---

If an asset in the case of a south service, or data stream in the case of a north task, has other data points or assets that are not part of the expression, then they too are subject to the threshold. If the expression evaluates to false then no assets will be forwarded on that stream. This allows a single value to control the forwarding of data.

Another example use might be to have two north streams, one that uses a high cost, link to send data when some condition that requires close monitoring occurs and the other that is used to send data by a lower cost mechanism when normal operating conditions apply.

E.g. We have a temperature critical process, when the temperature is above 80 degrees it must be closely monitored. We use a high cost link to send data north wards in this case. We would have a north task setup that has the threshold filter with the condition:

```
temperature >= 80
```

We then have a second, lower cost link with a north task using the threshold filter with the condition:

```
temperature < 80
```

This way all data is sent once, but data is sent in an expedited fashion if the temperature is above the 80 degree threshold.

Threshold filters are added in the same way as any other filters.

- Click on the Applications add icon for your service or task.
- Select the *threshold* plugin from the list of available plugins.
- Name your threshold filter.
- Click *Next* and you will be presented with the following configuration page



The screenshot shows a configuration window with a progress bar at the top. Step 1 is 'Plugin Name' and Step 2 is 'Review Configuration'. In the 'Review Configuration' step, there is a form with two fields: 'Expression' containing the text 'speed > 10' and 'Enabled' with a checked checkbox. At the bottom of the form are two buttons: 'Previous' and 'Done'.

- Enter the expression to control forwarding in the box labeled *Expression*
- Enable the filter and click on *Done* to activate it

## Expressions

The *fledge-filter-threshold* plugin makes use of the library to do run time expression evaluation. This library provides a rich mathematical operator set, the most useful of these in the context of this plugin are;

- Comparison operators (=, ==, <>, !=, <, <=, >, >=)
- Logical operators (and, nand, nor, not, or, xor, xnor, mand, mor)
- Mathematical operators (+, -, \*, /, %, ^)
- Functions (min, max, avg, sum, abs, ceil, floor, round, roundn, exp, log, log10, logn, pow, root, sqrt, clamp, inrange, swap)
- Trigonometry (sin, cos, tan, acos, asin, atan, atan2, cosh, cot, csc, sec, sinh, tanh, d2r, r2d, d2g, g2d, hyp)

Within the expression the data points of the asset become symbols that may be used; therefore if an asset contains values “voltage” and “current” the expression will contain those as symbols and an expression of the form

```
voltage * current > 1000
```

can be used to determine if power (voltage \* current) is greater than 1kW.

## 8.4 Fledge Notification Rule Plugins

### 8.4.1 Threshold Rule

The threshold rule is used to detect the value of a data point within an asset going above or below a set threshold.

The configuration of the rule allows the threshold value to be set, the operation and the datapoint used to trigger the rule.



- **Asset name:** The name of the asset that is tested by the rule.
- **Datapoint Name:** The name of the datapoint in the asset used for the test.
- **Condition:** The condition that is being tested, this may be one of  $>$ ,  $>=$ ,  $<=$  or  $<$ .
- **Trigger value:** The value used for the test.
- **Evaluation data:** Select if the data evaluate is a single value or a window of values.
- **Window evaluation:** Only valid if evaluation data is set to Window. This determines if the value used in the rule evaluation is the average, minimum or maximum over the duration of the window.
- **Time window:** Only valid if evaluation data is set to Window. This determines the time span of the window.

### 8.4.2 Moving Average Rule

The *fledge-rule-average* plugin is a notification rule that is used to detect when a value moves outside of the determined average by more than a specified percentage. The plugin only monitors a single asset, but will monitor all data points within that asset. It will trigger if any of the data points within the asset differ by more than the configured percentage, an average is maintained for each data point separately.

During the configuration of a notification use the screen presented to choose the average plugin as the rule.



The screenshot shows the 'Rule' configuration step in a four-part wizard. The progress bar at the top indicates the current step is 2, with steps 1 (Notification Instance), 3 (Delivery Channel), and 4 (Done) also visible. The main content area is titled 'Rule Plugin' and features a dropdown menu with four options: 'Average' (selected), 'OutOfBound', 'SimpleExpression', and 'Threshold'. To the right of the dropdown, a description reads: 'Trigger if the current value deviates from the moving average by more than a defined percentage'. Below the dropdown is a link labeled 'available plugins'. At the bottom of the screen are 'Previous' and 'Next' buttons.

The next screen you are presented with provides the configuration options for the rule.

This screenshot shows the configuration options for the 'Average' rule plugin. The progress bar at the top remains the same. The main content area contains five labeled input fields: 'Asset' with the value 'temperature', 'Deviation %' with the value '10', 'Direction' with a dropdown menu set to 'Both', 'Average' with a dropdown menu set to 'Simple Moving Average', and 'EMA Factor' with the value '10'. At the bottom are 'Previous' and 'Next' buttons.

The *Asset* entry field is used to define the single asset that the plugin should monitor.

The *Deviation %* defines how far away from the observed average the current value should be in order to be considered as triggering the rule.



Deviation %	<div>Above Average Below Average ✓ Both</div>
Direction	

The *Direction* entry is used to define if the rule should trigger when the current value is above average, below average or in both cases.

Average	<div>✓ Simple Moving Average Exponential Moving Average 10</div>
EMA Factor	

The *Average* entry is used to determine what type of average is used for the calculation. The average calculated may be either a simple moving average or an exponential moving average. If an exponential moving average is chosen then a second configuration parameter, *EMA Factor*, allows the setting of the factor used to calculate that average.

Exponential moving averages give more weight to the recent values compared to historical values. The smaller the EMA factor the more weight recent values carry. A value of 1 for *EMA Factor* will only consider the most recent value.

---

**Note:** The *Average* rule is not applicable to all data, only simple numeric values are considered and those values should not deviate with an average of 0 or close to 0 if good results are required. Data points that deviate wildly are also not suitable for this plugin.

---

### 8.4.3 Delta Rule

The *fledge-rule-delta* plugin is a notification rule that triggers when a data point value changes. When a datapoint that is monitored changes the plugin will trigger. An alias value is given for the triggered datapoint and is included in the reason message when the plugin triggers.

During the configuration of a notification use the screen presented to choose the delta plugin as the rule.



1 Notification Instance 2 Rule 3 Delivery Channel 4 Done

**Rule Plugin**

- Average
- Delta
- OutOfBound
- Simple-Expression

[available\\_plugins](#)

Previous Next

The next screen you are presented with provides the configuration options for the rule.

1 Notification Instance 2 Rule 3 Delivery Channel 4 Done

**Asset**

**JSON Configuration**

```

1 {
2   "datapoint_name": "alias_name",
3   "sinusoid": "cosinus"
4 }
  
```

Previous Next

- **Asset:** define the single asset that the plugin should monitor.
- **Datapoints:** the datapoints monitor and the alias for the datapoint that is used in the trigger reason.



### 8.4.4 Expression Rule

The *fledge-rule-simple-expression* is a notification rule plugin that evaluates a user defined function to determine if a notification has triggered or not. The rule will work with a single asset, but does allow access to all the data points within the asset.

During the configuration of a notification use the screen presented to choose the average plugin as the rule.

The next screen you are presented with provides the configuration options for the rule.

The *Asset* entry field is used to define the single asset that the plugin should monitor.

The *Expression to apply* defines the expression that will be evaluated each time the rule is checked. This should be a boolean expression that returns true when the rule is considered to have triggered. Each data point within the asset will become a symbol in the expression, therefore if your asset contains a data point called voltage, the symbol voltage can be used in the expression to obtain the current voltage reading. As an example to create an under voltage notification if the voltage falls below 48 volts, the expression to use would be;

```
voltage < 48
```

The trigger expression uses the same expression mechanism, as the , and plugins

Expression may contain any of the following...



- Mathematical operators (+, -, \*, /, %, ^)
- Functions (min, max, avg, sum, abs, ceil, floor, round, roundn, exp, log, log10, logn, pow, root, sqrt, clamp, inrange, swap)
- Trigonometry (sin, cos, tan, acos, asin, atan, atan2, cosh, cot, csc, sec, sinh, tanh, d2r, r2d, d2g, g2d, hyp)
- Equalities & Inequalities (=, ==, <>, !=, <, <=, >, >=)
- Logical operators (and, nand, nor, not, or, xor, xnor, mand, mor)

### 8.4.5 Watchdog Rule

The *fledge-rule-watchdog* is a notification rule plugin that is to detect the absence of data. The plugin is simply configured with the name of an asset to track and a period to test over. If no new readings are observed within that specified period of time then the rule will trigger.

During configuration of the notification choose the WatchDog rule from the set of available rules.

The screenshot shows the configuration wizard for a notification rule. At the top, a progress bar indicates four steps: 1. Notification Instance, 2. Rule (current step), 3. Delivery Channel, and 4. Done. Below the progress bar, the 'Rule Plugin' dropdown menu is open, displaying a list of available plugins: OutOfBound, Simple-Expression, Threshold, and WatchDog. The 'WatchDog' plugin is highlighted in blue. To the right of the dropdown, a description states: 'Generate a notification based on the last time of received data'. Below the dropdown, there is a link labeled 'available plugins'. At the bottom of the configuration panel, there are two buttons: 'Previous' and 'Next'.

The next step in the process is to enter the configuration options for the watchdog rule.

The screenshot shows the configuration options for the Watchdog rule. The 'Asset name' field is filled with the text 'sinusoid'. The 'Evaluation interval in ms' field is filled with the text '5000'. The progress bar at the top remains the same, with step 2 'Rule' being the current step. At the bottom of the configuration panel, there are two buttons: 'Previous' and 'Next'.



- **Asset name:** The name of the asset to track.
- **Evaluation interval in ms:** The period of time to monitor for new readings of the asset. This is expressed in milliseconds.

As soon as the configured time period expires, if no readings for the asset have been observed then the rule will trigger.

It is important to consider the tuning of the south service buffering when setting up watchdog plugins. The watchdog is based on data entering the Fledge buffer, hence if the south service buffers data for a period that is the same or close to the watchdog period then false triggering of the watchdog may occur. Ensure the south service maximum latency is less than the watchdog interval for reliable behavior.

## 8.5 Fledge Notification Delivery Plugins

### 8.5.1 Amazon Alexa Notification

The *fledge-notify-alexa* notification delivery plugin sends notifications via Amazon Alexa devices using the Alexa *NotifyMe* skill.

When you receive a notification Alexa will make a noise to say you have a new notification and the green light on your Alexa device will light to say you have waiting notifications. To hear your notifications simply say “Alexa, read my notifications”

To enable notifications on an Alexa device

- You must enable the NotifyMe skill on your Amazon Alexa device.
- Link this skill to your Amazon account
- NotifyMe will send you an access code that is required to configure this plugin.

Once you have created your notification rule and move on to the delivery mechanism

- Select the alexa plugin from the list of plugins
- Click *Next*

The screenshot shows the configuration screen for the Alexa notification plugin. At the top, a progress bar indicates the current step is 'Delivery Channel' (step 3 of 4). The main form contains the following fields:

- Access Code:** An empty text input field.
- Title:** A text input field containing the text "The level in tank 15 is below 10%".
- Enabled:** A checkbox that is checked.

At the bottom of the form, there are two buttons: "Previous" and "Next".

- Configure the plugin
  - **Access Code:** Paste the access code you received from the *NotifyMe* application here
  - **Title:** This is the title that the Alexa device will read to you
- Enable the plugin and click *Next*
- Complete your notification setup



When you notification triggers the Alexa device will read the title text to you followed by either “Notification has triggered” or “Notification has cleared”.

## 8.5.2 Asset Notification

The *fledge-notify-asset* notification delivery plugin is unusual in that it does not notify an external system, instead it creates a new asset which is then processed like any other asset within Fledge. This plugin is useful to inform up stream systems that a event has occurred and allow them to take action or merely as a way to have a record of a condition occurring which may not require any further actions.

Once you have created your notification rule and move on to the delivery mechanism

- Select the asset plugin from the list of plugins
- Click *Next*

The screenshot shows a configuration interface for a notification rule. At the top, a progress bar with four steps is visible: 1. Notification Instance, 2. Rule (current step), 3. Delivery Channel, and 4. Done. Below the progress bar, there is a form with three fields: 'Asset' with the value 'event', 'Description' with the value 'Notification alert', and 'Enabled' with an unchecked checkbox. At the bottom of the form, there are two buttons: 'Previous' and 'Next'.

- Now configure the asset delivery plugin
  - **Asset:** The name of the asset to create.
  - **Description:** A textual description to add to the asset
- Enable the plugin and click *Next*
- Complete your notification setup

The asset that will be created when the notification triggers will contain

- The timestamp of the trigger event
- Three data points
  - **rule:** The name of the notification that triggered this asset creation
  - **description:** The textual description entered in the configuration of the delivery plugin
  - **event:** This will be one of *triggered* or *cleared*. If the notification type was not set to be *toggled* then the *cleared* event will not appear. If *toggled* was set as the notification type then there will be a *triggered* value in the asset created when the rule triggered and a *cleared* value in the asset generated when the rule moved from the triggered to untriggered state.



### 8.5.3 Control Dispatcher Notification

The *fledge-notify-control* notification delivery plugin is a mechanism by which a notification can be used to send set point control writes and operations via the control dispatcher service.

Once you have created your notification rule and move on to the delivery mechanism

- Select the control plugin from the list of plugins
- Click *Next*

The screenshot shows the 'Rule' configuration step (step 2 of 4) for the 'Control Dispatcher Notification' plugin. The interface includes a progress bar at the top with steps: 1 Notification Instance, 2 Rule, 3 Delivery Channel, and 4 Done. The main configuration area has two sections: 'Trigger Value' and 'Cleared Value', each with a code editor. Both editors contain the following JSON payload:

```
1 {
2   "write": {
3     "name": "value"
4   }
5 }
```

At the bottom left, there is an 'Enabled' checkbox which is currently unchecked. At the bottom right, there are 'Previous' and 'Next' buttons.

- Configure the plugin
  - **Trigger Value:** The control payload to send to the dispatcher service. These are set when the notification rule triggers.
  - **Cleared Value:** The control payload to send to the dispatcher service. These are set when the notification rule clears.
- Enable the plugin and click *Next*
- Complete your notification setup



## Trigger Values

The *Trigger Value* and *Cleared Value* are JSON documents that are sent to the dispatcher. The format of these is a JSON document that describes the control operation to perform. The document contains a definition of the recipient of the control input, this may be a south service, an asset, an automation script or all south services. If the recipient is a service then the document contains a *service* key, the value of which is the name of the south service that should receive the control operation. To send the control request to the south service responsible for the ingest of a particular asset, then an *asset* key would be given. If sending to an automation script then a *script* key would be given. If known of the *service*, *asset* or *script* keys are given then the request will be sent to all south services that support control.

The document also contains the control request that should be made, either a *write* or an *operation* and the values to write or the name and parameters of the operation to perform.

The example below shows a JSON document that would cause the two values *temperature* and *rate* to be written to the south service called *oven001*.

```
{
  "service" : "oven001",
  "write": {
    "temperature" : "110",
    "rate"        : "245"
  }
}
```

In this example the values are constants defined in the plugin configuration. It is possible however to use values that are in the data that triggered the notification.

As an example of this assume we are controlling the speed of a fan based on the temperature of an item of equipment. We have a south service that is reading the temperature of the equipment, let's assume this is in an asset called *equipment* which has a data point called *temperature*. We add a filter using the *fledge-filter-expression* filter to calculate a desired fan speed. The expression we will use in this example is *desiredSpeed = temperature \* 100*. This will cause the asset to have a second data point called *desiredSpeed*.

We create a notification that is triggered if the *desiredSpeed* is greater than 0. The delivery mechanism will be this plugin, *fledge-notify-setpoint*. We want to set two values in the south plugin *speed* to set the speed of the fan and *run* which controls if the fan is on or off. We set the *Trigger Value* to the following

```
{
  "service" : "fan001",
  "write": {
    "speed" : "$equipment.desiredSpeed$",
    "run"   : "1"
  }
}
```

In this case the *speed* value will be substituted by the value of the *desiredSpeed* data point of the *equipment* asset that triggered the notification to be sent.

If then fan is controlled by the same south service that is ingesting the data into the asset *equipment*, then we could use the *asset* destination key rather than name the south service explicitly.

```
{
  "asset" : "equipment",
  "write": {
    "speed" : "$equipment.desiredSpeed$",
    "run"   : "1"
  }
}
```



Another option for controlling the destination of a control request is to broadcast it to all south services. In this example we will assume we want to trigger a shutdown operation across all the devices we monitor.

```
{
  "operation" : {
    "shutdown" : { }
  }
}
```

Here we are not giving *asset*, *script* or *service* keys, therefore the control request will be broadcast. Also we have used an *operation* rather than a *write* request. The operation name is *shutdown* and we have assumed it takes no arguments.

## 8.5.4 Custom Asset Notification

The *fledge-notify-customasset* notification delivery plugin is a plugin that creates an event asset in the Fledge readings. This event asset can be customised via the configuration and may include data from the asset that triggered the notification.

The asset created will contain a number of data points

- *description*: A fixed description that is set in the plugin configuration.
- *event*: The event that caused the asset to be created. This may be one of *triggered* or *cleared*.
- *rule*: The name of the notification rule that triggered the notification.
- *store*: The data that triggered the notification.

Once you have created your notification rule and move on to the delivery mechanism

- Select the *customasset* plugin from the list of plugins
- Click *Next*



- Configure the plugin
  - **Custom Asset:** The name of the asset to create
  - **Description:** The content to add in the *description* data point within the asset
  - **JSON Configuration:** This is a description of how to map the asset that triggered the notification to the data in the *store* data point of the event asset.
  - **Enable authentication:** Enable the authentication of the plugin to the Fledge API
  - **Username:** The user name to use when connecting to the Fledge API
  - **Password:** The password to use when connecting to the Fledge API
- Enable the plugin and click *Next*
- Complete your notification setup

## Store Configuration

The content of the *store* data point would normally contain data from the reading that triggered the notification. It will be written as a JSON data point type and will always contain the timestamp of the reading that triggered this notification.

The configuration consists of one or more asset names as a key, the value is an array of objects that defines the data point names to extract from the triggering asset and an alias to use in the store. The example below would include the *sinusoid* asset and the data point within *sinusoid*, also called *sinusoid*. However it would write this value using an alias of *cosinusoid*.



```
{
  "sinusoid": [
    {
      "sinusoid": "cosinusoid"
    }
  ]
}
```

This would result in an asset with a *store* data point that would be as follows

```
{"sinusoid":{"cosinusoid":0.994521895,"timestamp":"2022-09-08 11:31:29.323666 +0000"}}
```

### 8.5.5 Email Notifications

The *fledge-notify-email* delivery notification plugin allows notifications to be delivered as email messages. The plugin uses an SMTP server to send email and requires access to this to be configured as part of configuring the notification delivery method.

During the creation of your notification select the email notification plugin from the list of available notification mechanisms. You will be prompted with a configuration dialog in which to enter details of your SMTP server and of the email you wish to send.

The screenshot displays the 'Delivery Channel' configuration step for an email notification. The progress bar at the top indicates the current step is 3 of 4. The configuration form contains the following fields and values:

- To address:** alert.subscriber@dianomic.com
- To:** Notification alert subscriber
- Subject:** Fledge alert notification
- From address:** dianomic.alerts@gmail.com
- From name:** Notification alert
- SMTP Server:** smtp.gmail.com
- SMTP Port:** 587
- SSL/TLS:** ☒
- Username:** dianomic.alerts@gmail.com
- Password:** pass
- Enabled:** ☐

At the bottom, there are 'Previous' and 'Next' buttons.

- **To address:** The email address to which the notification will be sent
- **To:** A textual name for the recipient of the email
- **Subject:** A Subject to put in the email message
- **From address:** A from address to use for the email message
- **From name:** A from name to include in the email
- **SMTP Server:** The address of the SMTP server to which to send messages



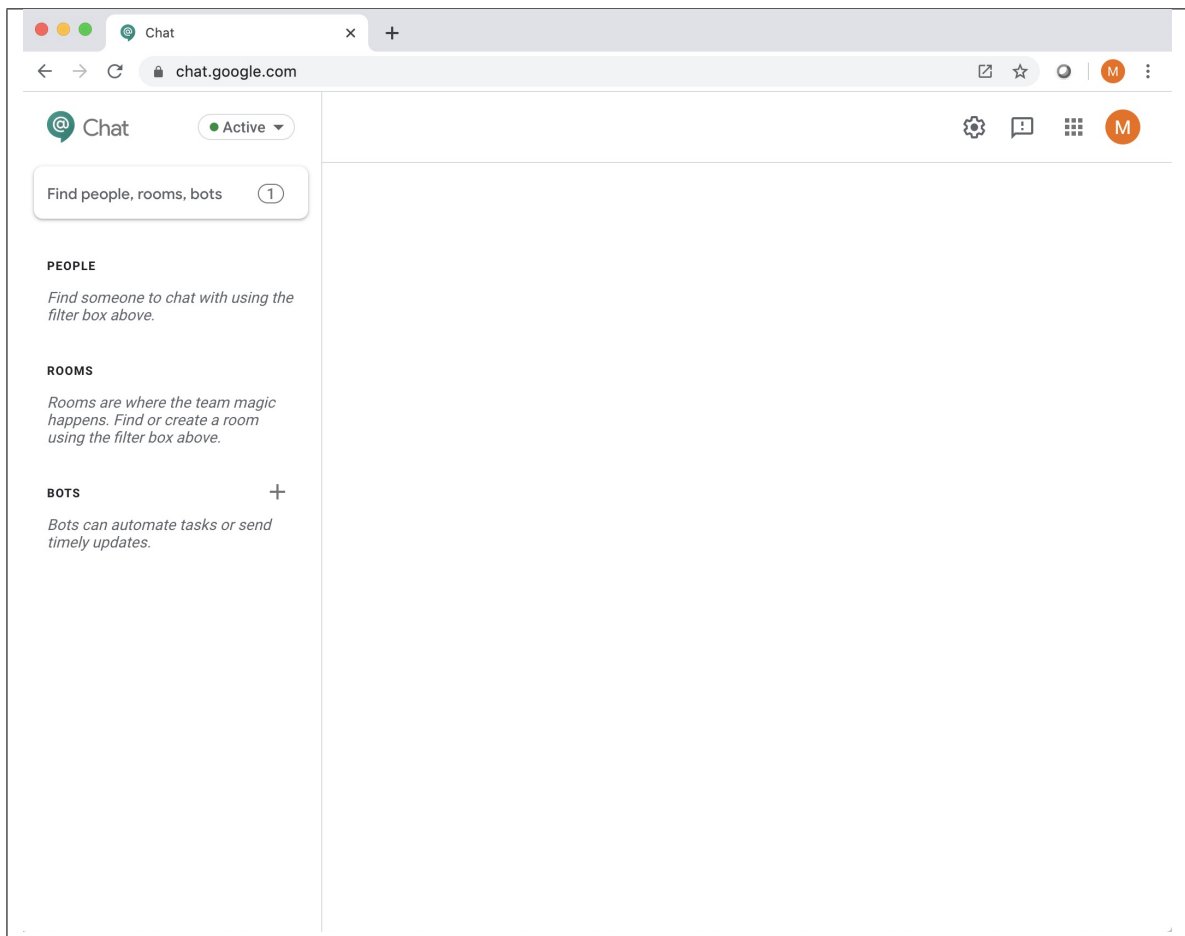
- **SMTP Port:** The port of your SMTP server
- **SSL/TLS:** A toggle to control if SSL/TLS encryption should be used when communicating with the SMTP server
- **Username:** A username to use to authenticate with the SMTP server
- **Password:** A password to use to authenticate with the SMTP server.

### 8.5.6 Google Chat

The *fledge-notify-google-hangouts* plugin allows notifications to be delivered to the Google chat platform. The notifications are delivered into a specific chat room within the application, in order to allow access to the chat room you must create a webhook for sending data to that chatroom.

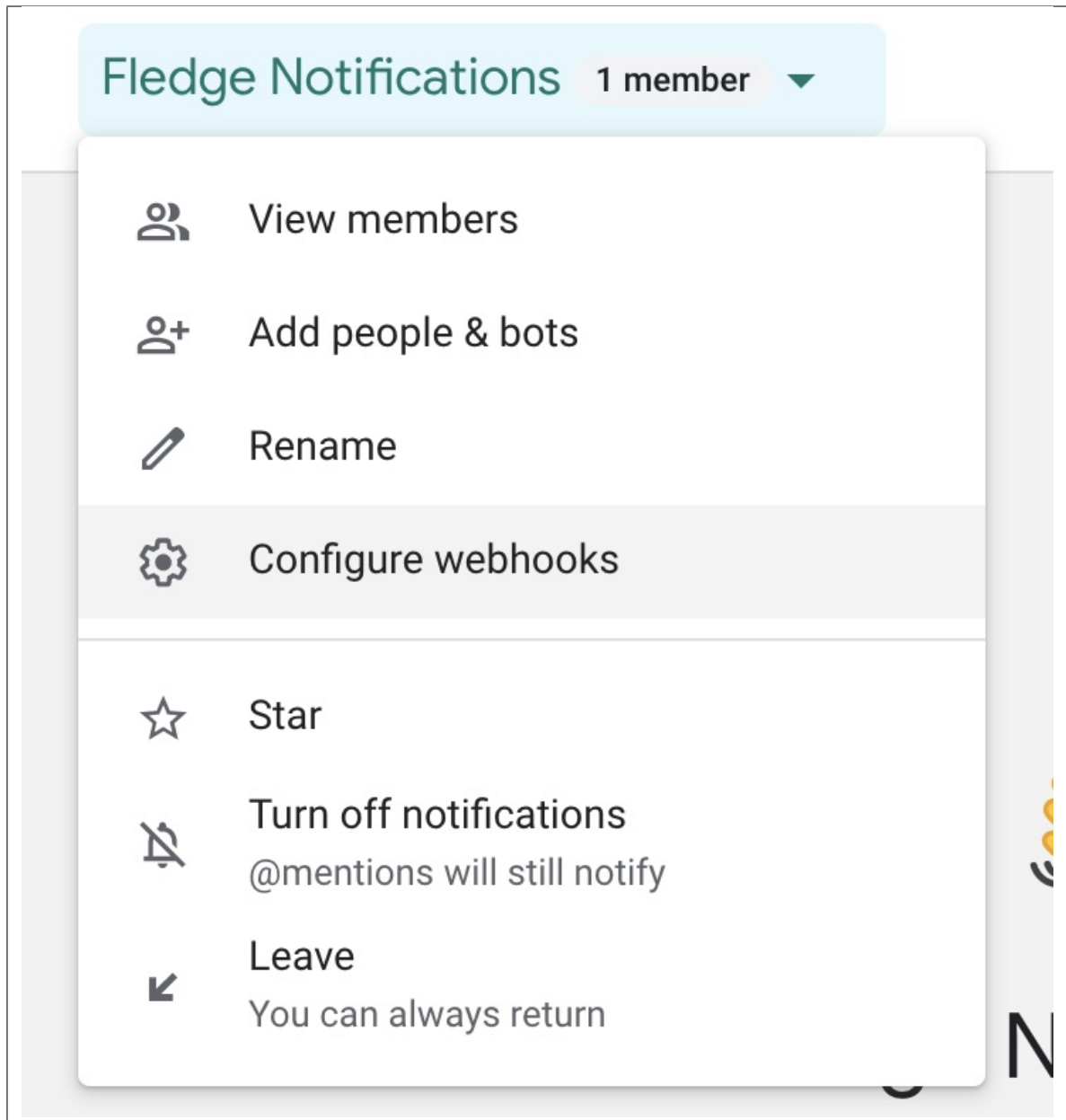
To create a webhook

- Go to the page in your browser



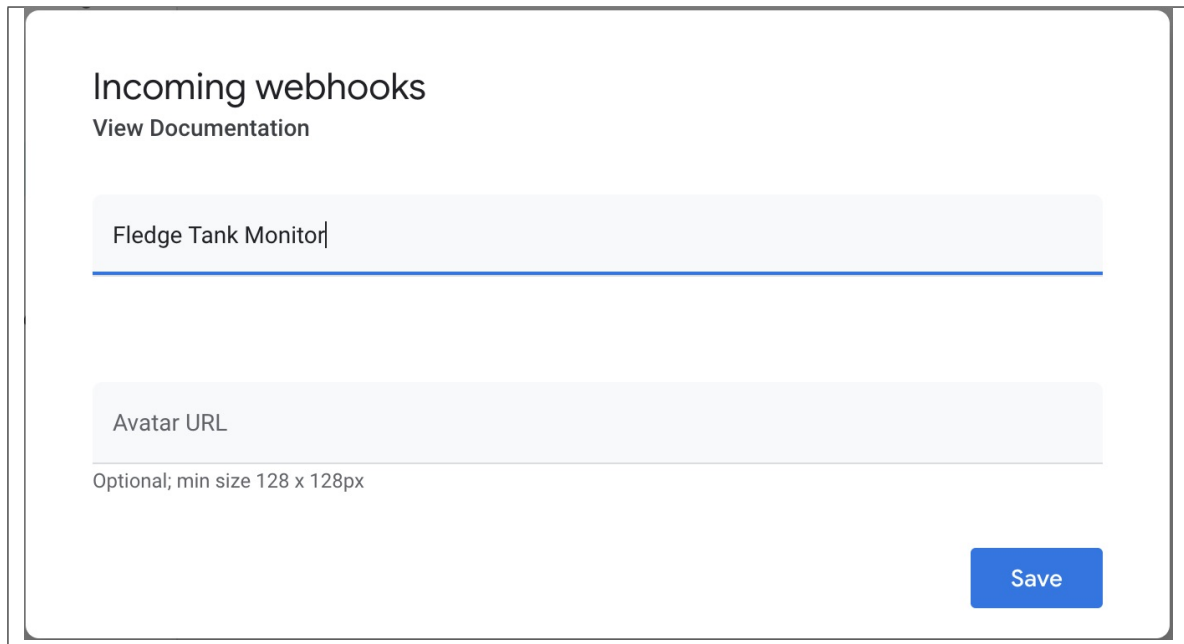
- Select the chat room you wish to use or create a new chat room
- In the menu at the top of the screen select *Configure webhooks*





- Enter a name for your webhook and optional avatar and click *Save*





Incoming webhooks

[View Documentation](#)

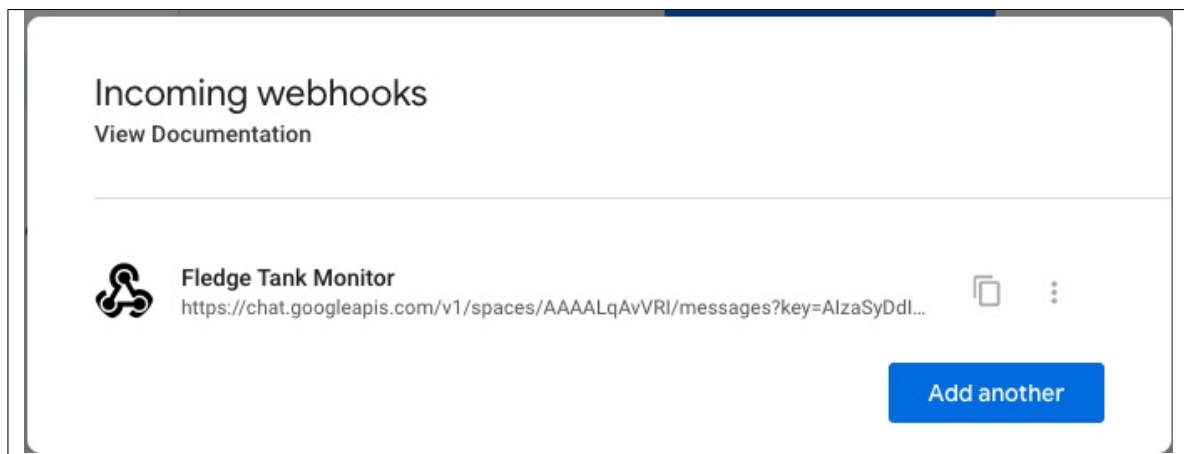
Fledge Tank Monitor

Avatar URL

Optional; min size 128 x 128px




Save

- Copy the URL that appears under your webhook name, you can use the copy icon next to the URL to place it in the clipboard



Incoming webhooks

[View Documentation](#)

 **Fledge Tank Monitor**  
<https://chat.googleapis.com/v1/spaces/AAAAALqAvVRI/messages?key=AlzaSyDdl...>  

Add another

- Close the webhooks window by clicking outside the window

Once you have created your notification rule and move on to the delivery mechanism

- Select the Hangouts plugin from the list of plugins
- Click *Next*



1 Notification Instance      2 Rule      3 Delivery Channel      4 Done

**Google Hangout Webhook URL**

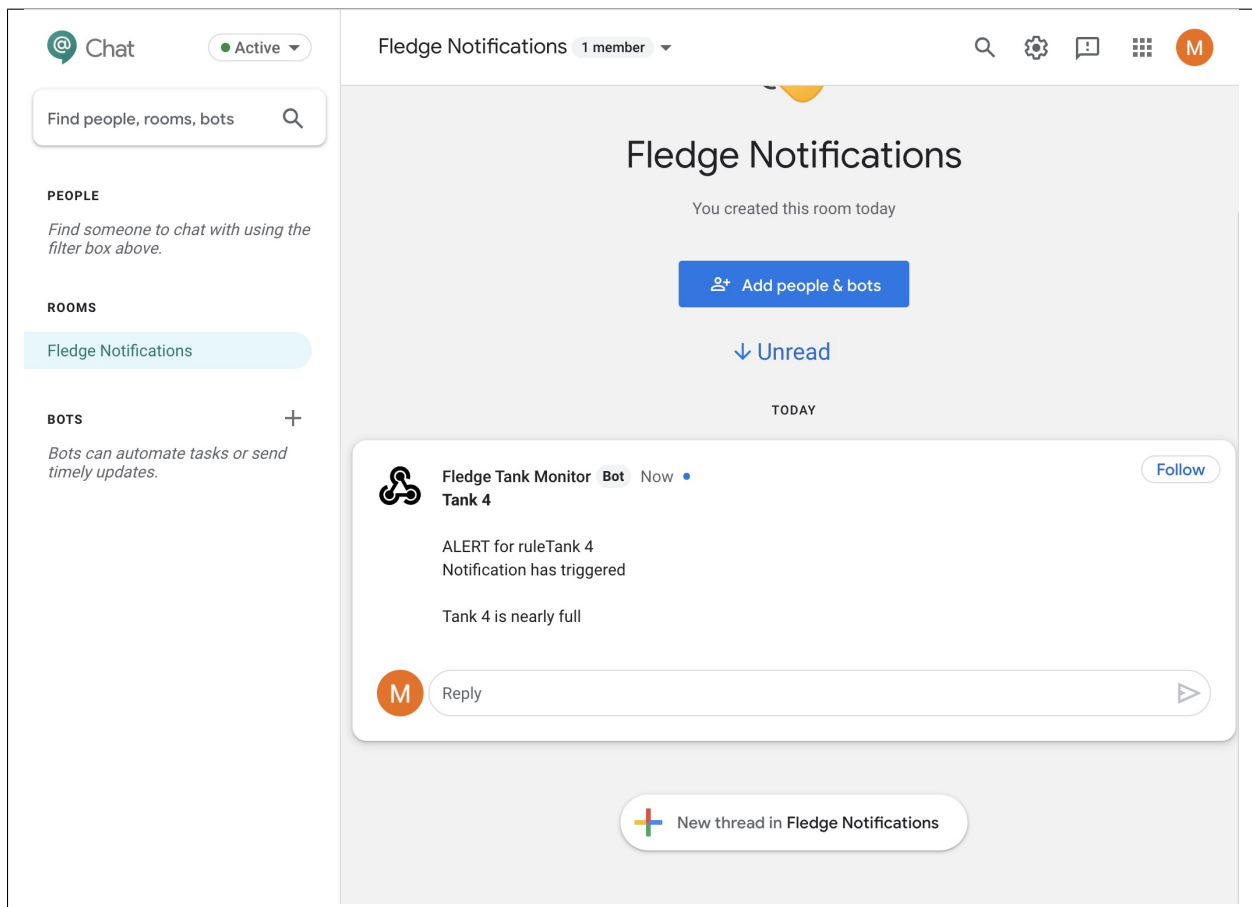
**Message Text**

**Enabled** ☒

[Previous](#) [Next](#)

- Now configure the asset delivery plugin
  - **Google Hangout Webhook URL:** Paste the URL obtain above here
  - **Message Text:** Enter the message text you wish to send
- Enable the plugin and click *Next*
- Complete your notification setup

A message will be sent to this chat room whenever a notification is triggered.



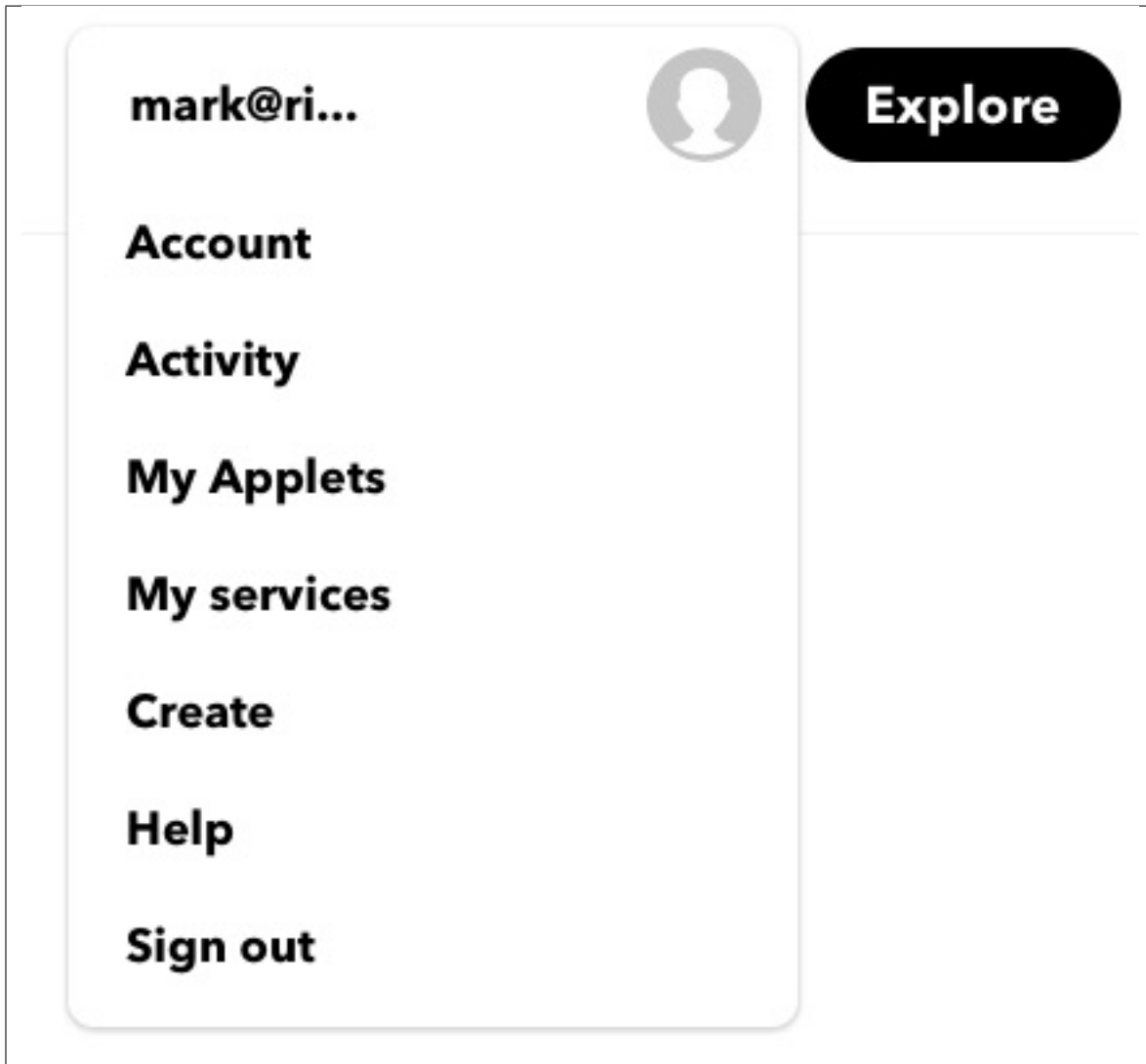


### 8.5.7 IFTTT Delivery Plugin

The *fledge-notify-ifttt* is a notification delivery plugin designed to trigger an action on the *If This Than That* IoT platform. IFTTT allows the user to setup a webhook that can be used to trigger processing on the platform. The webhook could be sending an IFTTT notification to a destination not support by any Fledge plugin to controlling a device that is controllable via IFTTT.

In order to use the IFTTT webhook you must obtain a key from IFTTT by visiting your IFTTT account

- Select the “My Applets” page from your account pull down menu



- Select “New Applet”
- Click on the blue “+ this” logo
- Choose the service Webhooks
- Click on the blue box “Receive a web request”
- Enter an “Event Name”, this may be of your choosing and will be put in the configuration entry ‘Trigger’ for the Fledge plugin



- Click on the “+ that” logo
- Select the action you wish to invoke

Once you have setup your webhook on IFTTT you can now proceed to setup the Fledge delivery notification plugin. Create you notification, choose and configure your notification rule. Select the IFTTT delivery plugin and click on *Next*. You will be presented with the IFTTT plugin configuration page.

The screenshot displays the IFTTT plugin configuration interface. At the top, a progress bar indicates the current step is 3, 'Delivery Channel'. The configuration area contains three fields: 'IFTTT Trigger' set to 'button\_press', 'IFTTT Key' set to a long alphanumeric string, and 'Enabled' checked. Navigation buttons 'Previous' and 'Next' are at the bottom.

There are two important items to be configured

- **IFTTT Trigger:** This is the *Maker Event* that you used in IFTTT when defining the action that the webhook should trigger.
- **IFTTT Key:** This is the webhook key you obtain from the IFTTT platform.

Enable the delivery and click on *Next* to move to the final stage of completing your notification.

### 8.5.8 MQTT Notification

The *fledge-notify-mqtt* notification delivery plugin sends notifications via an MQTT broker. The MQTT topic and the payloads to send when the notification triggers or is cleared are configurable.

Once you have created your notification rule and move on to the delivery mechanism

- Select the mqtt plugin from the list of plugins
- Click *Next*



1 Notification Instance      2 Rule      3 Delivery Channel      4 Done

**MQTT Broker**

**MQTT Topic**

**Trigger Payload**

**Clear Payload**

**Enabled** ☒

- Configure the plugin
  - **MQTT Broker:** The URL of your MQTT broker.
  - **Topic:** The MQTT topic on which to publish the messages.
  - **Trigger Payload:** The payload to send when the notification triggers
  - **Clear Payload:** The payload to send when the notification clears
- Enable the plugin and click *Next*
- Complete your notification setup

### 8.5.9 Operation Notification

The *fledge-notify-operation* notification delivery plugin is a mechanism by which a notification can be used to send a request to a south services to perform an operation.

Once you have created your notification rule and move on to the delivery mechanism

- Select the operation plugin from the list of plugins
- Click *Next*



- Configure the plugin
  - **Service:** The name of the south service you wish to control
  - **Trigger Value:** The operation payload to send to the south service when the rule triggers. This is the name of the operation to perform and a set of name, value pairs which are the optional parameters to pass that operations.
  - **Cleared Value:** The operation payload to send to the south service when the rule clears. This is the name of the operation to perform and a set of name, value pairs which are the optional parameters to pass that operations.
- Enable the plugin and click *Next*
- Complete your notification setup

### 8.5.10 Python 3 Script

The *fledge-notify-python35* notification delivery plugin allows a user supplied Python script to be executed when a notification is triggered or cleared. The script should be written in Python 3 syntax.

A Python script should be provided in the form of a function, the name of that function should match the name of the file the code is loaded from. E.g if you have a script to run which you have saved in a file called `alert_light.py` it should contain a function `alert_light`. ~that function is called with a message which is defined in notification itself as a simple string.

A second function may be provided by the Python plugin code to accept configuration from the plugin that can be used to modify the behavior of the Python code without the need to change the code. The configuration is a JSON document which is again passed as a Python Dict to the `set_filter_config` function in the user provided Python code. This function should be of the form



```
def set_filter_config(configuration):
    config = json.loads(configuration['config'])
    value = config['key']
    ...
    return True
```

Once you have created your notification rule and move on to the delivery mechanism

- Select the python35 plugin from the list of plugins
- Click *Next*

Notification Instance Rule Delivery Channel Done

Python script

```
1 from time import sleep
2 from envirophat import leds
3
4 def flash_leds(message):
5     for count in range(4):
6         leds.on()
7         sleep(0.5)
8         leds.off()
9         sleep(0.5)
10
```

flash\_leds.py

Choose Files flash\_leds.py

Configuration

```
1 {}
```

Enabled ☐

Previous Next

- Configure the plugin
  - **Python Script:** This is the script that will be executed. Initially you are unable to type in this area and must load your initial script from a file using the *Choose Files* button below the text area. Once a file has been chosen and loaded you are able to update the Python code in this page.

**Note:** Any changes made to the script in this screen will be written back to the original file it was loaded from.

- **Configuration:** You may enter a JSON document here that will be passed to the *set\_filter\_config* function of your Python code.
- Enable the plugin and click *Next*
- Complete your notification setup



## Example Script

The following is an example script that flashes the LEDs on the Enviro pHAT board on a Raspberry Pi

```
from time import sleep
from envirophat import leds
def flash_leds(message):
    for count in range(4):
        leds.on()
        sleep(0.5)
        leds.off()
        sleep(0.5)
```

This code imports some Python libraries and then in a loop will turn the leds on and then off 4 times.

**Note:** This example will take 4 seconds to execute, unless multiple threads have been turned on for notification delivery this will block any other notifications from being delivered during that time.

### 8.5.11 Set Point Control Notification

The *fledge-notify-setpoint* notification delivery plugin is a mechanism by which a notification can be used to send set point control writes into south services which support set point control

Once you have created your notification rule and move on to the delivery mechanism

- Select the setpoint plugin from the list of plugins
- Click *Next*

1 Notification Instance 2 Rule 3 Delivery Channel 4 Done

Service

Trigger Value

```
1 {
2   "values": {
3     "name": "value"
4   }
5 }
```

Cleared Value

```
1 {
2   "values": {
3     "name": "value"
4   }
5 }
```

Enabled ☐



- Configure the plugin
  - **Service:** The name of the south service you wish to control
  - **Trigger Value:** The set point control payload to send to the south service. This is a list of name, value pairs to be set within the service. These are set when the notification rule triggers.
  - **Cleared Value:** The set point control payload to send to the south service. This is a list of name, value pairs to be set within the service. These are set when the notification rule clears.
- Enable the plugin and click *Next*
- Complete your notification setup

### Trigger Values

The *Trigger Value* and *Cleared Value* are JSON documents that are sent to the set point entry point of the south service. The format of these is a set of name and value pairs that represent the data to write via the south service. A simple example would be as below

```
{
  "values": {
    "temperature" : "11",
    "rate"        : "245"
  }
}
```

In this example we are setting two variables in the south service, one named *temperature* and the other named *rate*. In this example the values are constants defined in the plugin configuration. It is possible however to use values that are in the data that triggered the notification.

As an example of this assume we are controlling the speed of a fan based on the temperature of an item of equipment. We have a south service that is reading the temperature of the equipment, let's assume this is in an asset called *equipment* which has a data point called *temperature*. We add a filter using the *fledge-filter-expression* filter to calculate a desired fan speed. The expression we will use in this example is  $desiredSpeed = temperature * 100$ . This will cause the asset to have a second data point called *desiredSpeed*.

We create a notification that is triggered if the *desiredSpeed* is greater than 0. The delivery mechanism will be this plugin, *fledge-notify-setpoint*. We want to set two values in the south plugin *speed* to set the speed of the fan and *run* which controls if the fan is on or off. We set the *Trigger Value* to the following

```
{
  "values" : {
    "speed" : "$equipment.desiredSpeed$",
    "run"   : "1"
  }
}
```

In this case the *speed* value will be substituted by the value of the *desiredSpeed* data point of the *equipment* asset that triggered the notification to be sent.



### 8.5.12 Slack Messages

The *fledge-notify-slack* delivery notification plugin allows notifications to be delivered as instant messages on the Slack messaging platform. The plugin uses a Slack webhook to post the message.

To obtain a webhook URL from Slack

- Visit the page
- Select *Create New App*
- Enter a name for your application, this must be unique for each Fledge slack application you create
- Select your Slack workspace in which to deliver your notification. If not already logged in you may need to login to your workspace
- Click on Create
- Select *Incoming Webhooks*
- Activate your webhook
- Add your webhook to the workspace
- Select the channel or individual to send the notification to
- Authorize your webhook
- Copy the Webhook URL which you will need when configuring the plugin

Once you have created your notification rule and move on to the delivery mechanism

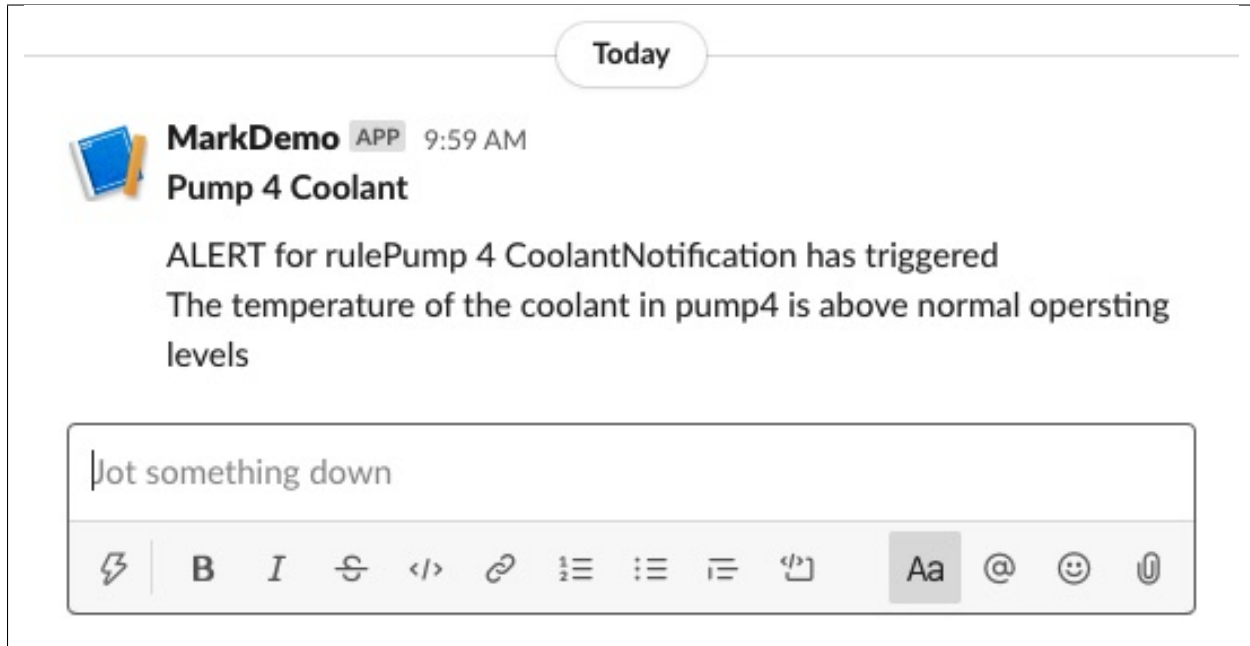
- Select the slack plugin from the list of plugins
- Click *Next*

The screenshot shows the configuration interface for the Slack plugin. At the top, a progress bar indicates four steps: 1. Notification Instance, 2. Rule (current step), 3. Delivery Channel, and 4. Done. Below the progress bar, the configuration form is displayed. It includes a 'Slack Webhook URL' field with a text input containing a long URL, a 'Message Text' field with a text input containing 'The temperature of the coolant in pump4 is above normal operating levels', and an 'Enabled' checkbox which is checked. At the bottom of the form, there are 'Previous' and 'Next' buttons.

- Configure the delivery plugin
  - **Slack Webhook URL:** Paste the URL you obtain above from the page
  - **Message Text:** Static text that will appear in the slack message you receive when the rule triggers
- Enable the plugin and click *Next*
- Complete your notification setup

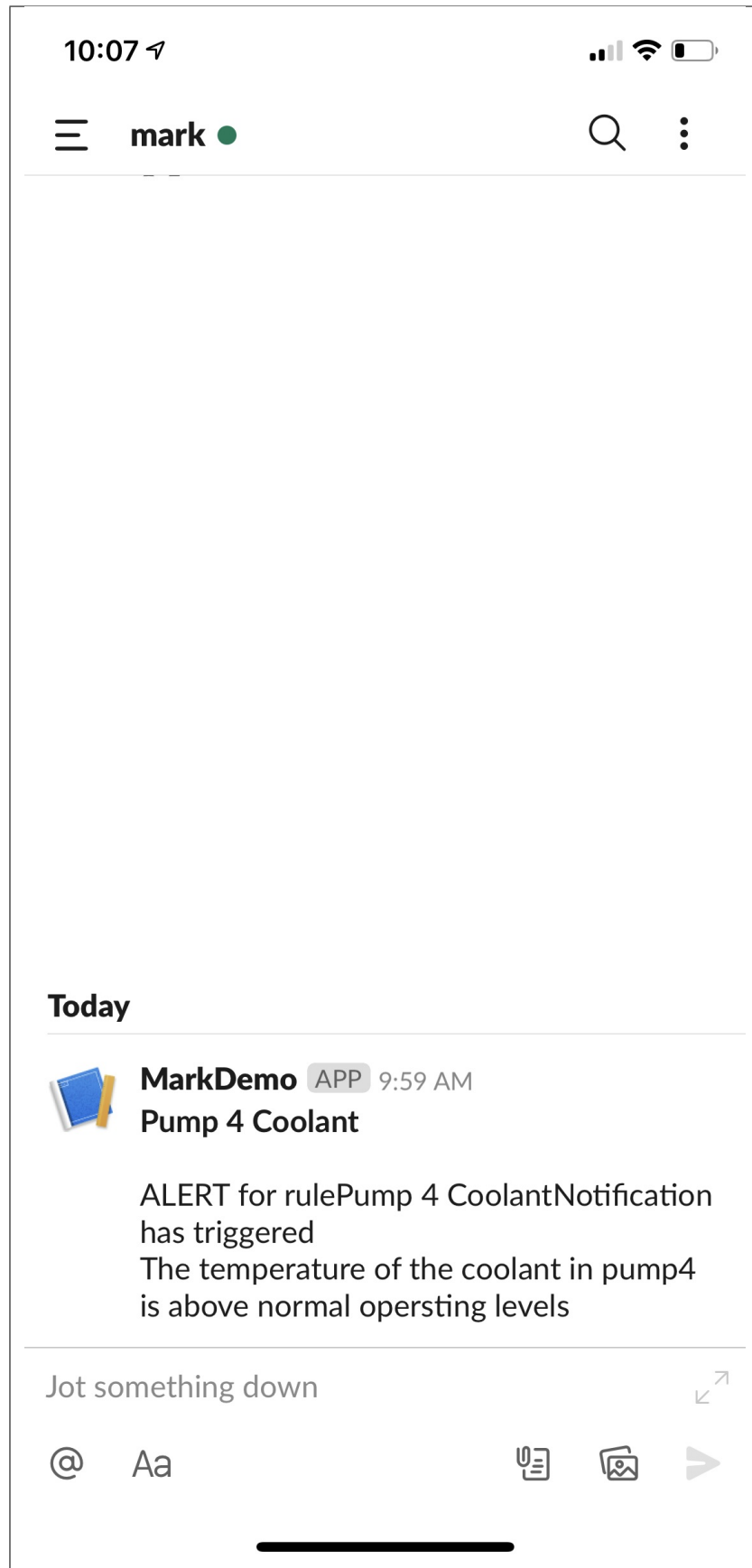
When the notification rule triggers you will receive messages in you Slack client on your desk top





and/or your mobile devices



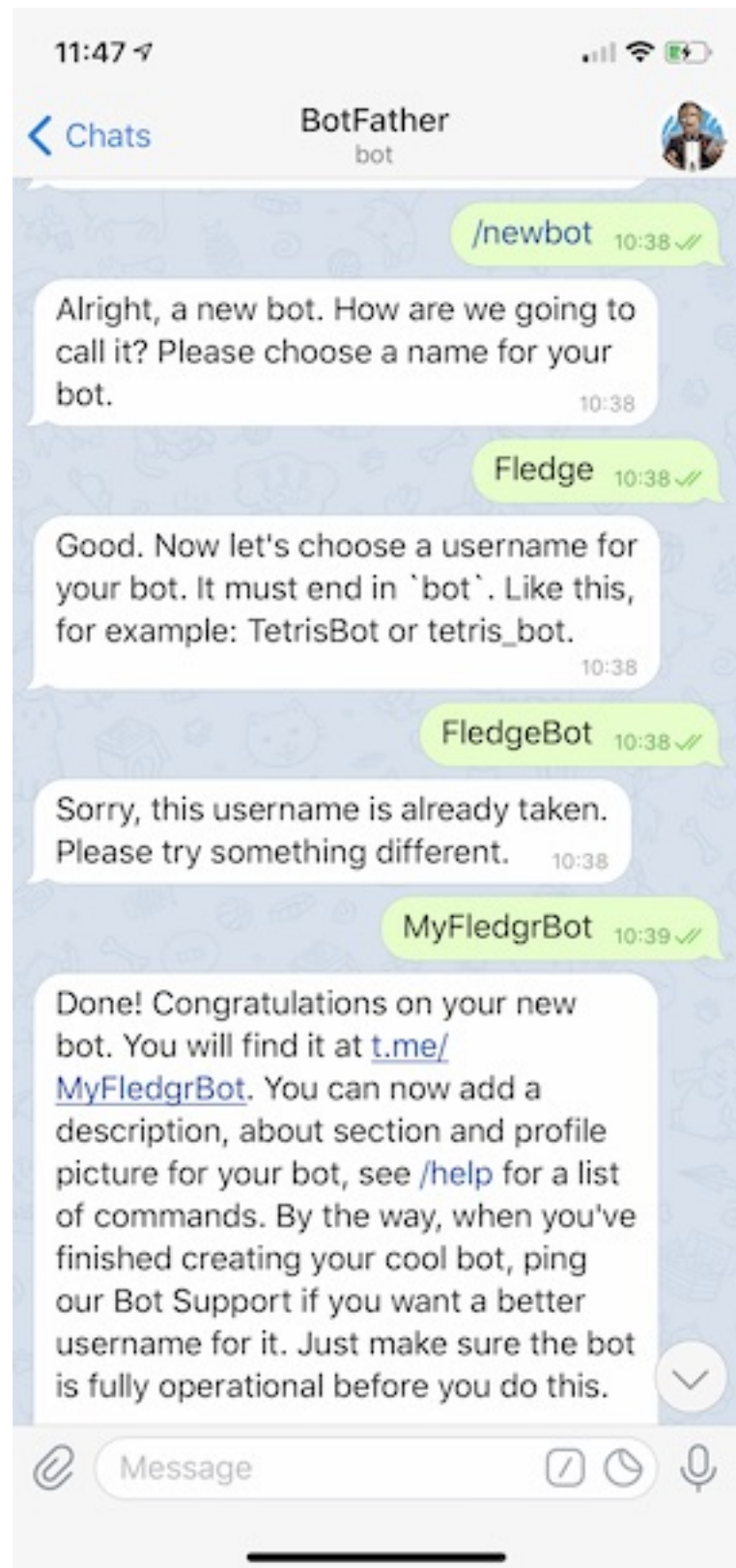




### 8.5.13 Telegram Messages

The *fledge-notify-telegram* delivery notification plugin allows notifications to be delivered as instant messages on the Telegram messaging platform. The plugin uses Telegram BOT API, to use this you must create a BOT and obtain a token.





To obtain a Telegram BOT token

- Use the Telegram application to send a message to *botfather*.



- In your message send the text /start
- Then send the message /newbot
- Follow the instructions to name your BOT

- Copy your BOT token.

You now need to get a chat id

- In the Telegram application send a message to you chat BOT
- Run the following command at the your shell command line or use a web browser to go to the URL <https://api.telegram.org/bot<YourBOTToken>/getUpdates>

```
wget https://api.telegram.org/bot<YourBOTToken>/getUpdates
```

Examine the contents of the getUpdates file or the output from the web browser

- Extract the id from the “chat” JSON object

```
{ "ok":true, "result": [ { "update_id":562812724, "message": { "message_id":1, "from": { "id":1166366214, "is_bot":false, "first_name":"Mark", "last_name":"Riddoch" }, "chat": { "id":1166366214, "first_name":"Mark", "last_name":"Riddoch", "type":"private" }, "date":1588328344, "text":"/start", "entities": [ { "offset":0, "length":6, "type":"bot_command" } ] } } ], }
```

Once you have created your notification rule and move on to the delivery mechanism

- Select the Telegram plugin from the list of plugins
- Click *Next*

1 Notification Instance      2 Rule      3 Delivery Channel      4 Done

Telegram BOT API token

Telegram user chat\_id

Telegram BOT API url prefix

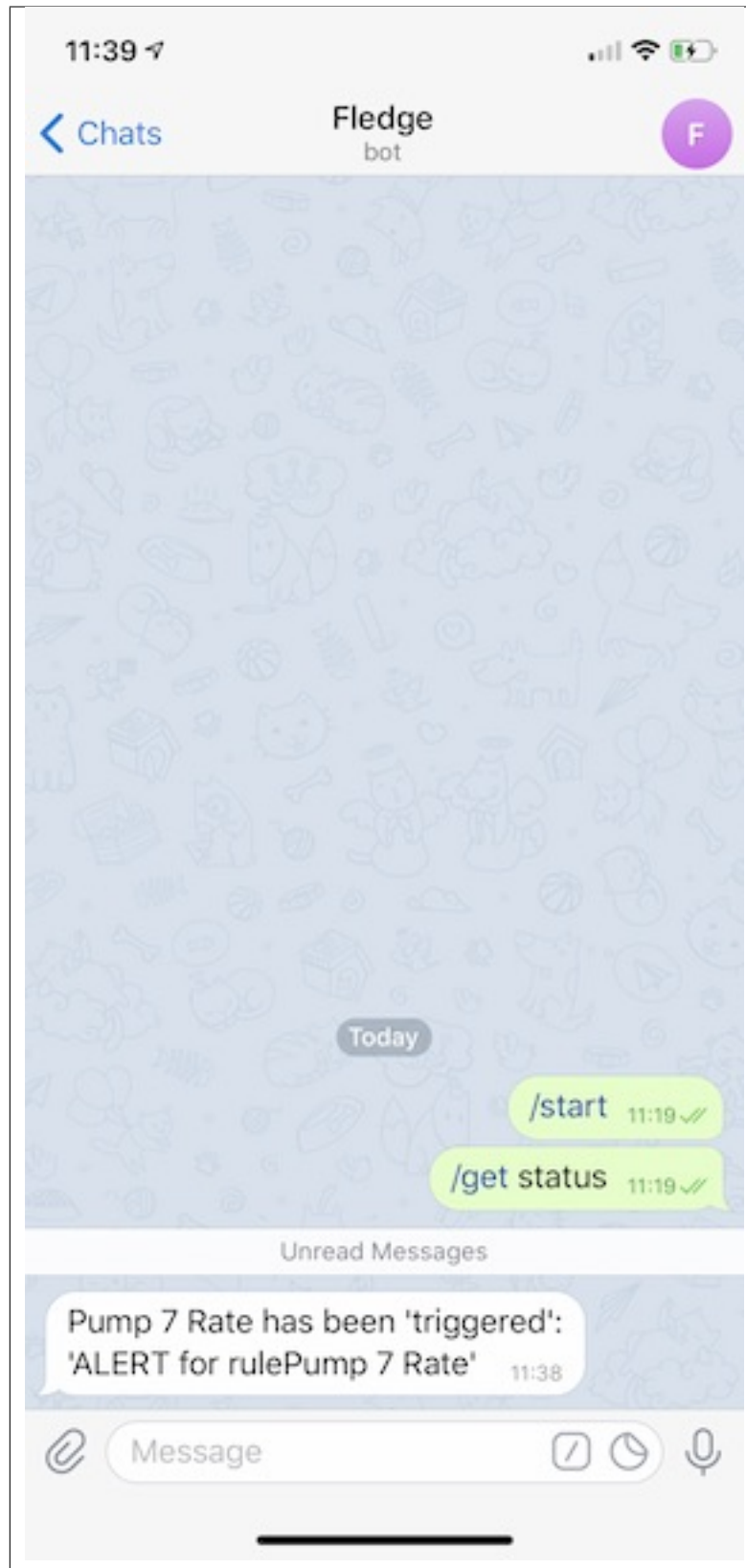
Enabled ☐

Previous Next

- Configure the delivery plugin
  - **Telegram BOT API token:** Paste the API token you received from botfather
  - **Telegram user chat\_id:** Paste the id field form the chat
  - **Telegram BOT API url Prefix:** This is the fixed part of the URL used to send messages and should not be modified under normal circumstances.
- Enable the plugin and click *Next*
- Complete your notification setup

When the notification rule triggers you will receive messages Telegram application







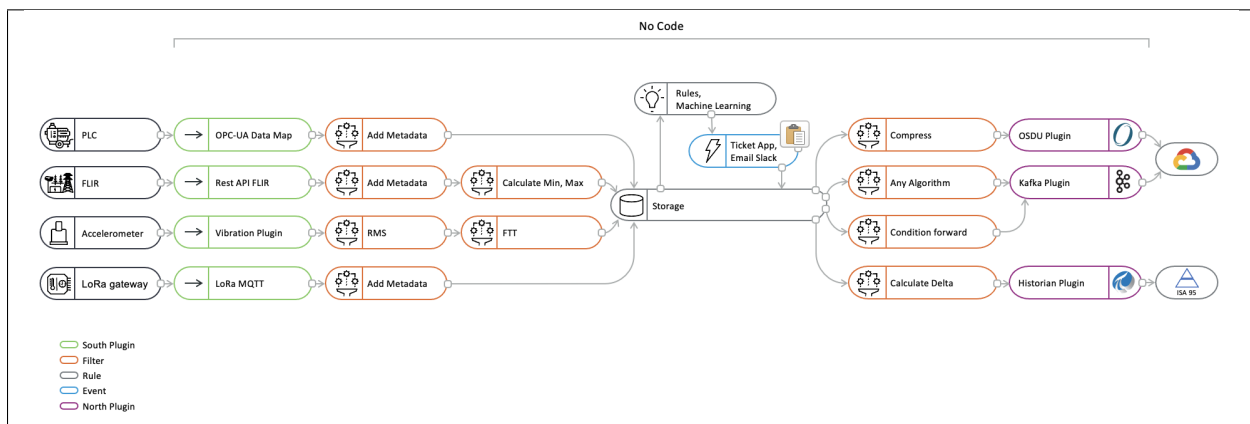




## DEVELOPING DATA PIPELINES

Fledge provides a system of data pipelines that allows data to flow from its point of ingest into the Fledge instance, the south plugin, to the storage layer in which it is buffered. The stages along this pipeline are Fledge processing filters, output of one filter becomes the input of the next. Fledge also supports pipelines on the egress as data flows from the storage layer to the north plugins and onward to the systems integrated upstream of the Fledge instance.

Operations in the south service are performed on the data from a single source, whilst operations in the north are performed on data going to a single destination. The filter pipeline in the north will have the data from sources flowing through the pipeline, this data will form a mixed stream that will contain all the data in date/time order.



### 9.1 Best Practices

It is possible with Fledge to support multiple data pipelines within a single Fledge instance, however if you have a well established Fledge instance with critical pipelines running on that instance it is perhaps not always the best practice to then develop a new, experimental pipeline on that same Fledge instance.

Looking first at south plugins; one reason for this is that data that enters the Fledge instance via your new pipeline will be sent to the storage system and then onward to the north destinations mixed with the data from other pipelines on your system. If your new pipeline is incorrect or generating poor data you are then left in a situation whereby that data has been sent to your existing upstream systems.

If it is unavoidable to use the same instance there are techniques that can be used to reduce the risk; namely to use an to block data from your new south pipeline entering your existing north pipelines and then being sent to your upstream systems. To do this you merely insert a filter at the start of each of your existing north pipelines and set it to exclude the named assets that will be ingested by your new, experimental pipeline. This will allow the data from the existing south service to still flow to the upstream systems, but prevent your new data from streaming out to these systems.



There are still risks associated with this approach, namely that the new service may produce assets of a different name to those you expect or may produce more assets than you expect. Data is still also sent to the notification service from your new pipeline, which may impact that service, although it is less likely than sending incorrect or unwanted data north. There is also the limitation that your new data will be discarded from the buffer and can not then be sent to the existing north pipelines if you subsequently decide the data is good. Data with your new asset names, from your new pipeline, will only be sent once you remove the from those pipelines in the north that send data to your upstream systems.

Developing new north pipelines is less risky, as the data that comes from the storage service and is destined for your new pipeline to upstream systems is effectively duplicated as it leaves the storage system. The main risk is that this new service will count as if the data has been sent up stream as far as the storage system is concerned and may make your data eligible for operation by the purge system sooner than would otherwise be the case. If you wish to prevent this you can update the purge configuration to insist the data is sent on all north channels before being considered sent for the purposes of the purge system. In most circumstances this is a precaution that can be ignored, however if you have configured your Fledge system for aggressive purging of data you may wish to consider this.

### 9.1.1 Incremental Development

The Fledge pipeline mechanism is designed for and lends itself to a modular development of the data processing requirement of your application. The pipeline is built from a collection of small, targeted filters that each perform a small, incremental process on the data. When building your pipelines, especially when using the filters that allow the application of scripts to the data, you should consider this approach and not build existing functionality that can be imported by applying an existing filter to the pipeline. Rather use that existing filter and add more steps to your pipeline, the Fledge environment is designed to provide minimal overhead when combining filters into a pipeline. Also the pipeline builder can make use of well used and tested filters, thus reducing the overheads to develop and test new functionality that is not needed.

This piecemeal approach can also be adopted in the process of building the pipeline, especially if you use the to block data from progressing further through the Fledge system once it has been buffered in the storage layer. Simply add your south service, bring the service up and observe the data that is buffered from the service. You can now add another filter to the pipeline and observe how this alters the data that is being buffered. Since you have a block on the data flowing further within your system, this data will disappear as part of the normal purging process and will not end up in upstream systems to the north of Fledge.

If you are developing on a standalone Fledge instance, with no existing north services, and you still set your experimental data to disappear, this can be achieved by use of the purge process. Simply configure the purge process to frequently purge data and set the process to purge unsent data. This will mean that the data will remain in the buffer for you to examine for a short time before it is purged from that buffer. Simply adjust the purge interval to allow you enough time to view the data in the buffer. Provided all the experimental data has been purged before you make your system go live, you will not be troubled with your experimental data being sent upstream.

Remember of course to reconfigure the purge process to be more inline with the duration you wish to keep the data for and to turn off the purging of unsent data unless you are willing to loose data that can not be sent for a period of time greater than the purge interval.

Configuring a more aggressive purge system, with the purging of unsent data, is probably not something you would wish to do on an existing system with live data pipelines and should not be used as a technique for developing new pipelines on such a system.

An alternative approach for removing data from the system is to enable the *Developer Features* in the Fledge User Interface. This can be done by selecting the *Settings* page in the left hand menu and clicking the option on the bottom of that screen.



Amongst the extra features introduced by selecting *Developer Features* will be the ability to manually purge data from the Fledge data store. This on-demand purging can be either applied to a single asset or to all assets within the data store. The manual purge operations are accessed via the *Assets & Readings* item in the Fledge menu. A number of new icons will appear when the *Developer Features* are turned on, one per asset and one that impacts all assets.

Asset	Readings
PREsine2.sinusoid	82,210
sinusoid2	6,308



These icons resemble erasers and are located in each row of the assets and also in the top right corner next to the help icon. Clicking on the eraser icon in each of the rows will purge the data for just that asset, leaving other assets untouched. Clicking on the icon in the top right corner will purge all the assets currently in the data store.



In both cases a confirmation dialog will be displayed to ensure against accidental use. If you choose to proceed the selected data within the Fledge buffer, either all or a specific asset, will be erased. There is no way to undo this operation or to retrieve the data once it has been purged.

Another consequence that may occur when developing new pipelines is that assets are created during the development process which are not required in the finished pipeline. The asset however remains associated with the service and the asset name and count of number of ingested readings will be displayed in the *South Services* page on the user interface.

<u>sine2</u>	enabled	sinusoid	1.9.2	sine2	734
				sine250	177



It is possible to deprecate the relationship between the service and the asset name using the developer features of the user interface. To do this you must first enable *Developer Features* in the user interface settings page. Now when you view the *South Services* page you will see an eraser icon next to each asset listed for a service.

<u>sine2</u>	enabled	sinusoid	1.9.2	 sine2	734
				 sine250	196

If you click on this icon you will be prompted to deprecate the relationship between the asset and the service. If you select *Yes* the relationship will be severed and the asset will no longer appear next to the service.

Deprecating the relationship will not remove the statistics for the asset, it will merely remove the relationship with the service and hence it will not be displayed against the service.

If an asset relationship is deprecated for an asset that is still in use, it will automatically be reinstated the next time a reading is ingested for that asset. Since the statistics were not deleted when the relationship was deprecated the previous readings will still be included in the statistics when the relationship is restored.

These *Developer Features* are designed to be of use when developing pipelines within Fledge, the functionality is not something that should be used in normal operation and the developer features should be turned off when pipelines are not being developed.

## Sacrificial North System

Developing north pipelines in a piecemeal fashion can be more of an issue as you are unlikely to want to put poorly formatted data into your upstream systems. One approach to this is to have a sacrificial north system of some type that you can use to develop the pipeline and determine if you are performing the process you need to on that pipeline. This way it is unimportant if that system becomes polluted with data that is not in the form you require it. Ideally you would use a system of the same type to do your development and then switch to the production system when you are satisfied your pipeline is correct.

If this is not possible for some reason a second choice solution would be to use another Fledge instance as your test north system. Rather than configure the north plugin you ultimately wish to use you would install the north HTTP plugin and connect this to a second Fledge instance running an HTTP plugin. Your data would then be sent to your new Fledge instance where you can then examine the data to see what was sent by the first Fledge instance. You then build up your north pipeline on that first Fledge instance in the same way you did with your south pipeline. Once satisfied you will need to carefully recreate your north pipeline against the correct north plugin and then you may remove your experimental north pipeline and destroy your sacrificial Fledge instance that you used to buffer and view the data.

### 9.1.2 OMF Specific Considerations

Certain north plugins present specific problems to the incremental development approach as changing the format of data that is sent to them can cause them internal issues. The plugin that is used to send data to the Aveva PI Server is one such plugin.

The problem with the PI Server is that it is designed to store data in fixed formats, therefore having data that is not of a consistent type, i.e. made up of the set of attributes, can cause issues. In a PI server each new data type becomes a new tag, this is not a problem if you are happy to use tag naming that is flexible. However if you require that you used fixed name tags within the PI Server, using the filter, this can be an issue for incremental development of your pipeline. Changing the properties of the tag will result in a new name being required for the tag.

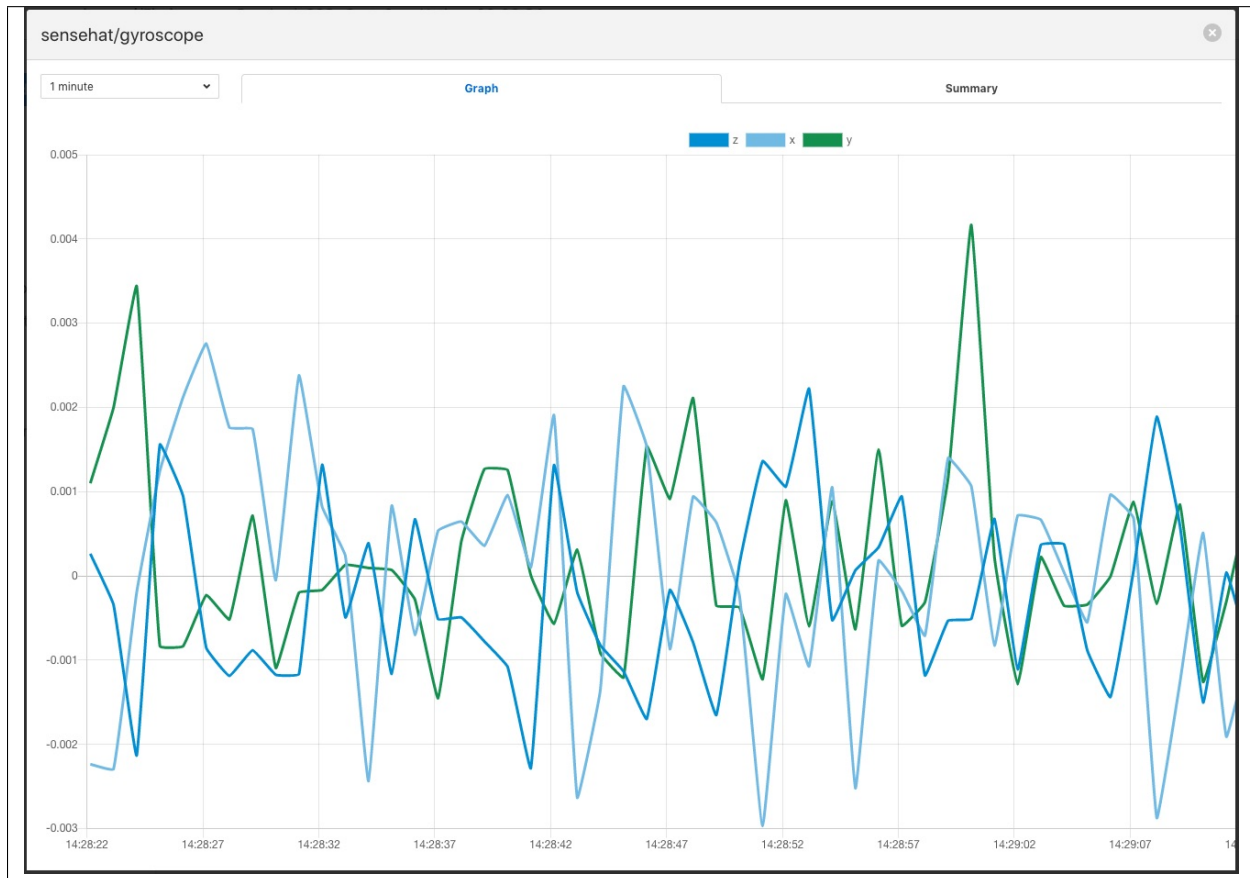
The simplest approach is to do all the initial development without the fixed name and then do the name mapping as the final step in developing the pipeline. Although not ideal it gives a relatively simple approach to resolving the problem.



Should you subsequently need to reuse the tag names with different types it becomes necessary to clear the type definitions from the PI Server by removing the element templates, the elements themselves and the cache. The PI Web API will then need to be restarted and the Fledge north plugin removed and recreated.

### 9.1.3 Examining Data

The easiest way to examine your data you have ingested via your new south pipeline is by use of the Fledge GUI to examine the data that currently resides within the buffer. You can view the data either via the graph feature of the Assets & Readings page, which will show the time series data.



If you have data that is not timeseries by nature, such as string, you may use the tabular displayed to show you non timeseries data, images if there are any or the download of the data to a spreadsheet view. This later view will not contain any image data in the readings.



sensehat_gyroscope-readings			
timestamp	z	x	y
2020-05-04 14:30:49.145006	0.000792725	0.0010765493	0.0022465843
2020-05-04 14:30:48.145022	0.0010982286	-0.0004502609	0.000719551
2020-05-04 14:30:47.145006	0.0007928684	0.0032151192	-0.0011130939
2020-05-04 14:30:46.145008	-0.0013448559	0.0047423765	0.0001088944
2020-05-04 14:30:45.145000	-0.0004286431	0.0007723272	-0.0020291833
2020-05-04 14:30:44.144999	-0.0001233947	0.0013834909	0.0007194807
2020-05-04 14:30:43.145001	-0.000734292	-0.0001437888	0.0004143068

### 9.1.4 Examining Logs

It is important to view the logs for your service when building a pipeline, this is due to the Fledge goal that Fledge instances should run as unattended services and hence any errors or warnings generated are written to logs rather than to an interactive user session. The Fledge user interface does however provide a number of mechanisms for viewing the log data and filtering it to particular sources. You may view the log from the “System” item in the Log menu and then filter the source to your particular south or north service.

The screenshot shows the Fledge System Logs interface. On the left, a sidebar contains a navigation menu with categories like Dashboard, Assets & Readings, South, North, Notifications, Control Dispatcher, Configuration, Schedules, Certificate Store, Backup & Restore, and Logs. Under the Logs category, 'System' is selected. The main panel, titled 'System Logs', has a search bar and two dropdown menus for 'Service' (set to 'Simple') and 'Severity' (set to 'Info and above'). Below these, a list of log entries is displayed, each starting with a timestamp and followed by a message. Most entries are 'ERROR' messages stating: 'The supplied Python script does not define a valid "convert" function'. One entry is an 'ERROR' message stating: 'HTTP error during service registration: 400: A Service with the same name already exists'.

Alternatively if you display the north or south configuration page for your service you will find an icon in the bottom left of the screen that looks like a page of text with the corner folded over. Simply click on this icon and the log screen will be displayed and automatically filtered to view just the logs from the service whose configuration you were previously editing.





Log are displayed with the most recent entry first, with older entries shown as you move down the page. You may move to the next page to view older log entries. It is also possible to view different log severity; fatal, error, warning, info and debug. By default a service will not write info and debug messages to the log, it is possible to turn these levels on via the advanced configuration options of the service. This will then cause the log entries to be written, but before you can view them you must set the appropriate level of severity filtering and the user interface will filter out information and debug message by default.

It is important to turn the logging level back down to warning and above messages once you have finished your debugging session and failure to do this will cause excessive log entries to be written to the system log file.

Also note that the logs are written to the logging subsystem of the underlying Linux version, either syslog or the messages mechanism depending upon your Linux distribution. This means that these log files will be automatically rotated by the operating system mechanisms. This means the system will not, under normal circumstances, fill the storage subsystem. Older log files will be kept for a short time, but will be removed automatically after a few days. This should be borne in mind if you have information in the log that you wish to keep. Also the user interface will only allow you to view data in the most recent log file.

It is also possible to configure the syslog mechanism to write log files to non-standard files or remote machines. The Fledge mechanisms for viewing the system logs does require that the standard names for log files are used.

### 9.1.5 Enabling and Disabling Filters

It should be noted that each filter has an individual enable control, this has the advantage that it is easy to temporarily remove a filter from a pipeline during the development stage. However this does have the downside that it is easy to forget to enable a filter in the pipeline or accidentally add a filter in a disabled state.

### 9.1.6 Scripting Plugins

Where there is not an existing plugin that does what is required, either in a filter or in south plugins where the data payload of a protocol is highly variable, such as generic REST or MQTT plugins, Fledge offers the option of using a scripting language in order to extend the off the shelf plugin set.

This scripting is done via the Python scripting language, both Python 3 and Python 2 are supported by Fledge, however it is recommended that the Python 3 variant, be used by preference. The Python support allows external libraries to be used to extend the basic functionality of Python, however it should be noted currently that the Python libraries have to be manually installed on the Fledge host machine.



### Scripting Guidelines

The user has the full range of Python functionality available to them within the script code they provides to this filter, however caution should be exercised as it is possible to adversely impact the functionality and performance of the Fledge system by misusing Python features to the detriment of Fledge's own features.

The general principles behind all Fledge filters apply to the scripts included in these filters;

- Do not duplicate existing functionality provided by existing filters.
- Keep the operations small and focused. It is better to have multiple filters each with a specific purpose than to create large, complex Python scripts.
- Do not buffer large quantities of data, this will effect the footprint of the service and also slow the data pipeline.

### Importing Python Packages

The user is free to import whatever packages they wish in a Python script, this includes the likes of the numpy packages and other that are limited to a single instance within a Python interpreter.

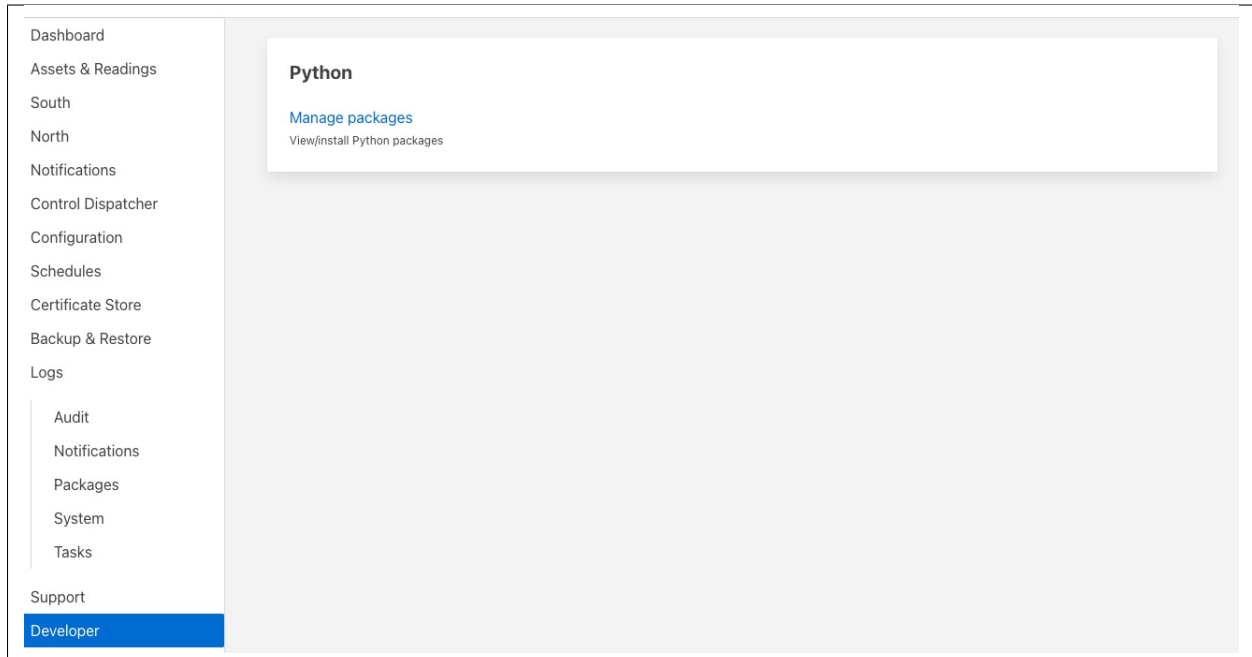
Do not import packages that you do not use or are not required. This adds an extra overhead to the filter and can impact the performance of Fledge. Only import packages you actually need.

Python does not provide a mechanism to remove a package that has previously been imported, therefore if you import a package in your script and then update your script to no longer import the package, the package will still be in memory from the previous import. This is because we reload updated scripts without closing down as restarting the Python interpreter. This is part of the sharing of the interpreter that is needed in order to allow packages such as numpy and scipy to be used. This can lead to misleading behavior as when the service gets restarted the package will not be loaded and the script may break because it makes use of the package still.

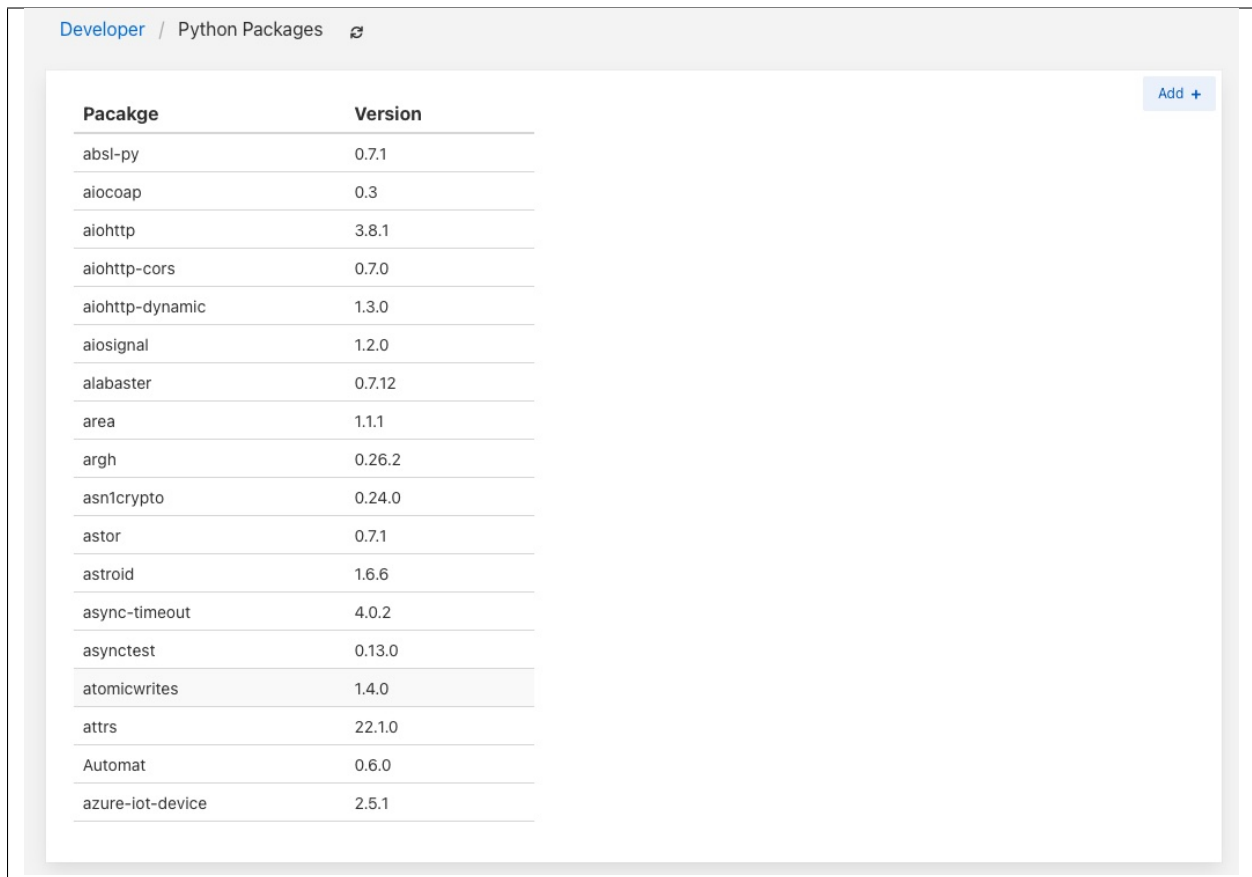
If you remove a package import from your script and you want to be completely satisfied that the script will still run without it, then you must restart the service in which you are using the plugin. This can be done by disabling and then re-enabling the service.

One of the *Developer Features* of the Fledge user interface allows the management of the installed Python Packages from within the user interface. This features is turned on via the *Developer features* toggle in the *Settings* page and will add a new menu item called *Developer*. Navigating to this page will give the the option of managing packages





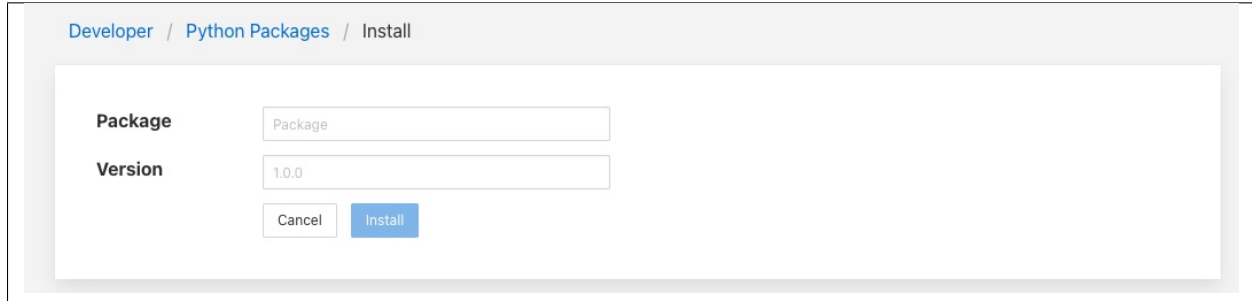
Clicking on *Manage packages* link will display the current set of Python packages that are installed on the machine.



To add a new package click on the *Add +* link in the top right corner. This will display a screen that allows you to



enter details of a Python package to install.



Enter package name and an optional package version and then click on the *Install* button to install a new package via *pip3*.

### Use of Global Variables

You may use global variables within your script and these globals will retain their value between invocations of the of processing function. You may use global variables as a method to keep information between executions and perform such operations as trend analysis based on data seen in previous calls to the filter function.

All Python code within a single service shares the same Python interpreter and hence they also share the same set of global variables. This means you must be careful as to how you name global variables and also if you need to have multiple instances of the same filter in a single pipeline you must be aware that the global variables will be shared between them. If your filter uses global variables it is normally not recommended to have multiple instances of them in the same pipeline.

It is tempting to use this sharing of global variables as a method to share information between filters, this is not recommended as should not be used. There are several reasons for this

- It provides data coupling between filters, each filter should be independent of each other filter.
- One of the filters sharing global variables may be disabled by the user with unexpected consequences.
- Filter order may be changed, resulting in data that is expected by a later filter in the chain not being available.
- Intervening filters may add or remove readings resulting in the data in the global variables not referring to the same reading, or set of readings that it was intended to reference.

If you wish one filter to pass data onto a later filter in the pipeline this is best done by adding data to the reading, as an extra data point. This data point can then be removed by the later filter. An example of this is the way Fledge adds to readings that are processed and removed by the north plugin.

For example let us assume we have calculated some value delta that we wish to pass to a later filter, we can add this as a data point to our reading which we will call *\_hintDelta*.

```
def myPython(readings):  
    for elem in list(readings):  
        reading = elem['readings']  
        ...  
        reading['_hintDelta'] = delta  
        ...  
    return readings
```

This is far better than using a global as it is attached to the reading to which it refers and will remain attached to that reading until it is removed. It also means that it is independent of the number of readings that are processed per call, and resilient to readings being added or removed from the stream.



The name chosen for this data point in the example above has no significance, however it is good practice to choose a name that is unlikely to occur in the data normally and portrays the usage or meaning of the data.

## File IO Operations

It is possible to make use of file operations within a Python filter function, however it is not recommended for production use for the following reasons;

- Pipelines may be moved to other hosts where files may not be accessible.
- Permissions may change dependent upon how Fledge systems are deployed in the various different scenarios.
- Edge devices may also not have large, high performance storage available, resulting in performance issues for Fledge or failure due to lack of space.
- Fledge is designed to be managed solely via the Fledge API and applications that use the API. There is no facility within that API to manage arbitrary files within the filesystem.

It is common to make use of files during development of a script to write information to in order to aid development and debugging, however this should be removed, along with associated imports of packages required to perform the file IO, when a filter is put into production.

## Threads within Python

It is tempting to use threads within Python to perform background activity or to allow processing of data sets in parallel, however there is an issue with threading in Python, the Python Global Interpreter Lock or GIL. The GIL prevents two Python statements from being executed within the same interpreter by two threads simultaneously. Because we use a single interpreter for all Python code running in each service within Fledge, if a Python thread is created that performs CPU intensive work within it, we block all other Python code from running within that Fledge service.

We therefore avoid using Python threads within Fledge as a means to run CPU intensive tasks, only using Python threads to perform IO intensive tasks, using the asyncio mechanism of Python 3.5.3 or later. In older versions of Fledge we used multiple interpreters, one per filter, in order to workaround this issue, however that had the side effect that a number of popular Python packages, such as numpy, pandas and scipy, could not be used as they can not support multiple interpreters within the same address space. It was decided that the need to use these packages was greater than the need to support multiple interpreters and hence we have a single interpreter per service in order to allow the use of these packages.

## Interaction with External Systems

Interaction with external systems, using network connections or any form of blocking communication should be avoided in a filter. Any blocking operation will cause data to be blocked in the pipeline and risks either large queues of data accumulating in the case of asynchronous south plugins or data being missed in the case of polled plugins.

## Scripting Errors

If an error occurs in the plugin or Python script, including script coding errors and Python exception, details will be logged to the error log and data will not flow through the pipeline to the next filter or into the storage service.

Warnings raised will also be logged to the error log but will not cause data to cease flowing through the pipeline.

See [Examining Logs](#): for details have how to access the system logs.







## **SECURING FLEDGE**

The default installation of a Fledge service comes with security features turned off, there are several things that can be done to add security to Fledge. The REST API by default support unencrypted HTTP requests, it can be switched to require HTTPS to be used. The REST API and the GUI can be protected by requiring authentication to prevent users being able to change the configuration of the Fledge system. Authentication can be via username and password or by means of an authentication certificate.

### **10.1 Enabling HTTPS Encryption**

Fledge can support both HTTP and HTTPS as the transport for the REST API used for management, to switch between there two transport protocols select the *Configuration* option from the left-hand menu and the select *Admin API* from the configuration tree that appears,



Fledge Admin and User REST API

**Enable HTTP** ☒

**HTTP Port** 8081

**HTTPS Port** 1995

**Certificate Name** fledge

**Authentication** optional

**Authentication method** any

**Auth Certificate** ca

**Allow Ping** ☒

**Password Expiry Days** 0

**Auth Providers**

```

1 {
2   "providers": [
3     "username",
4     "ldap"
5   ]
6 }

```

The first option you will see is a tick box labeled *Enable HTTP*, to select HTTPS as the protocol to use this tick box should be deselected.

Fledge Admin and User REST API

**Enable HTTP** ☐

**HTTP Port** 8081

**HTTPS Port** 1995

**Certificate Name** fledge

When this is unticked two options become active on the page, *HTTPS Port* and *Certificate Name*. The HTTPS Port is



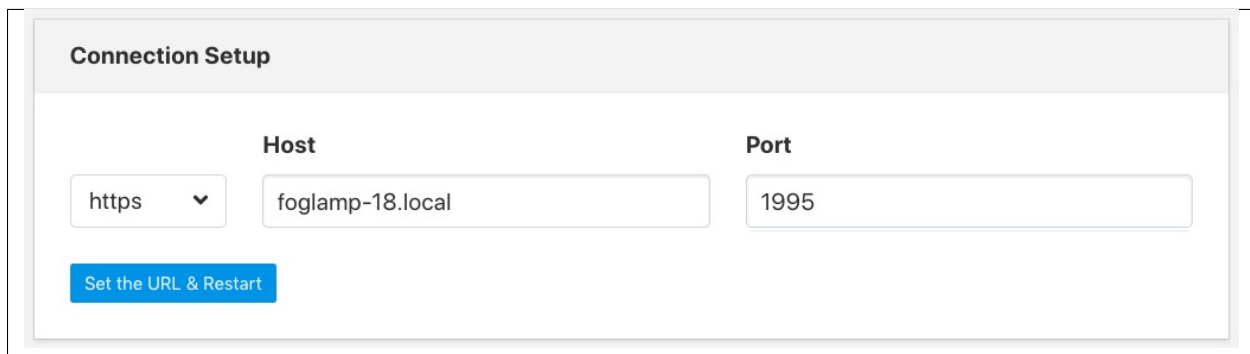
the port that Fledge will listen on for HTTPS requests, the default for this is port 1995.

The *Certificate Name* is the name of the certificate that will be used for encryption. The default is to use a self signed certificate called *fledge* that is created as part of the installation process. This certificate is unique per fledge installation but is not signed by a certificate authority. If you require the extra security of using a signed certificate you may use the Fledge *Certificate Store* functionality to upload a certificate that has been created and signed by a certificate authority.

After enabling HTTPS and selecting save you must restart Fledge in order for the change to take effect. You must also update the connection setting in the GUI to use the HTTPS transport and the correct port.

*Note:* if using the default self-signed certificate you might need to authorise the browser to connect to IP:PORT. Just open a new browser tab and type the URL `https://YOUR_FLEDGE_IP:1995`

Then follow browser instruction in order to allow the connection and close the tab. In the Fledge GUI you should see the green icon (Fledge is running).



The screenshot shows a 'Connection Setup' dialog box. It has a title bar 'Connection Setup'. Below the title bar, there are two columns: 'Host' and 'Port'. Under 'Host', there is a dropdown menu showing 'https' and a text input field containing 'foglamp-18.local'. Under 'Port', there is a text input field containing '1995'. At the bottom left of the dialog, there is a blue button labeled 'Set the URL & Restart'.

## 10.2 Requiring User Login

In order to set the REST API and GUI to force users to login before accessing Fledge select the *Configuration* option from the left-hand menu and then select *Admin API* from the configuration tree that appears.



**Fledge** foglamp-18/Fledge Demo

Dashboard  
Assets & Readings  
South  
North  
Notifications  
**Configuration**  
Schedules  
Certificate Store  
Backup & Restore  
Logs  
Audit  
Notifications  
Packages  
System  
Tasks  
Support  
Settings

General

► Installation  
► Admin API  
► Fledge Service

Fledge Admin and User REST API

**Enable HTTP** ☒

**HTTP Port** 8081

**HTTPS Port** 1995

**Certificate Name** fledge

**Authentication** optional

**Authentication method** any

**Auth Certificate** ca

**Allow Ping** ☒

**Password Expiry Days** 0

**Auth Providers**

```

1 {
2   "providers": [
3     "username",
4     "ldap"
5   ]
6 }

```

Two particular items are of interest in this configuration category that is then displayed; *Authentication* and *Authentication method*

**Certificate Name** fledge

**Authentication**

**Authentication method**

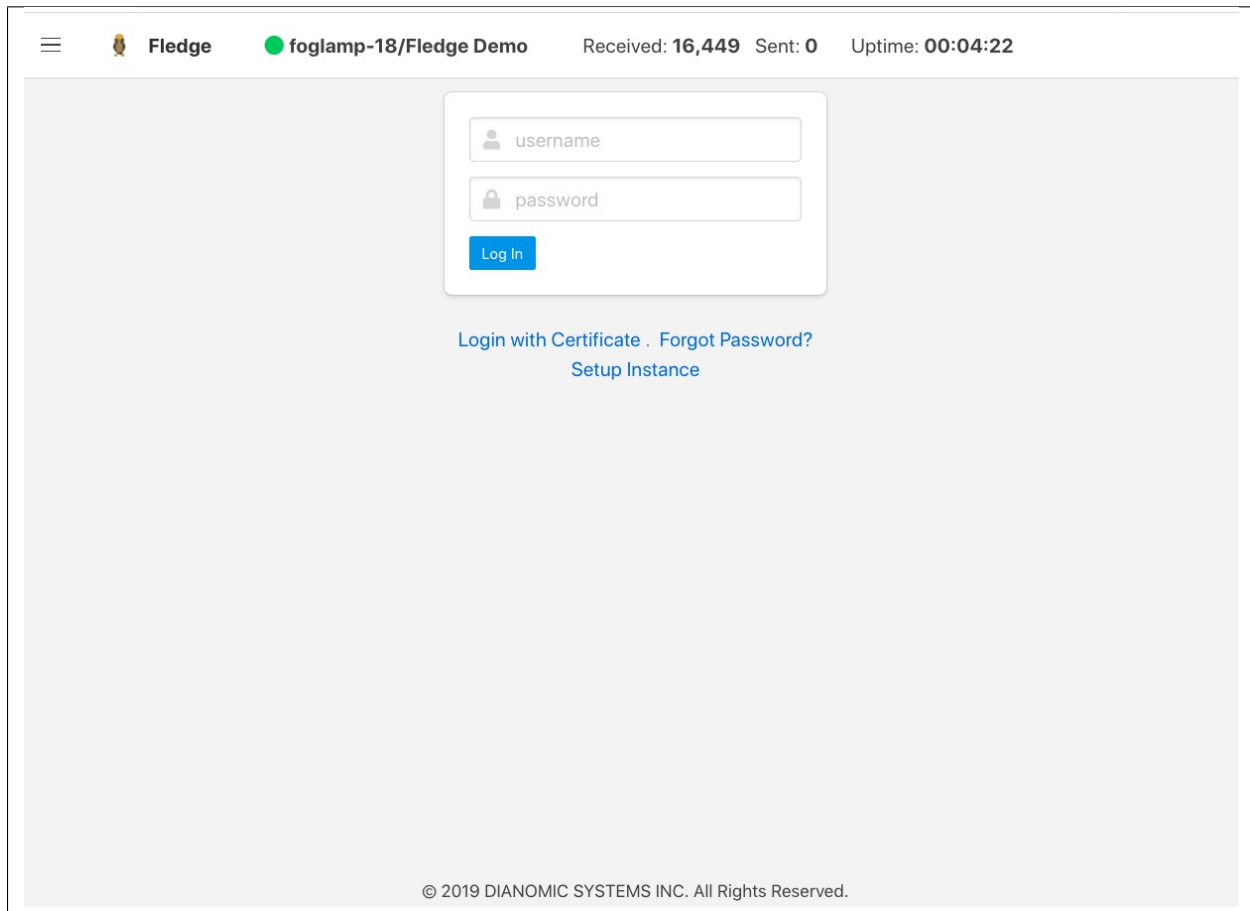
✓ mandatory  
optional  
password

Select the *Authentication* field to be mandatory and the *Authentication method* to be password. Click on *Save* at the bottom of the dialog.

In order for the changes to take effect Fledge must be restarted, this can be done in the GUI by selecting the restart item in the top status bar of Fledge. Confirm the restart of Fledge and wait for it to be restarted.



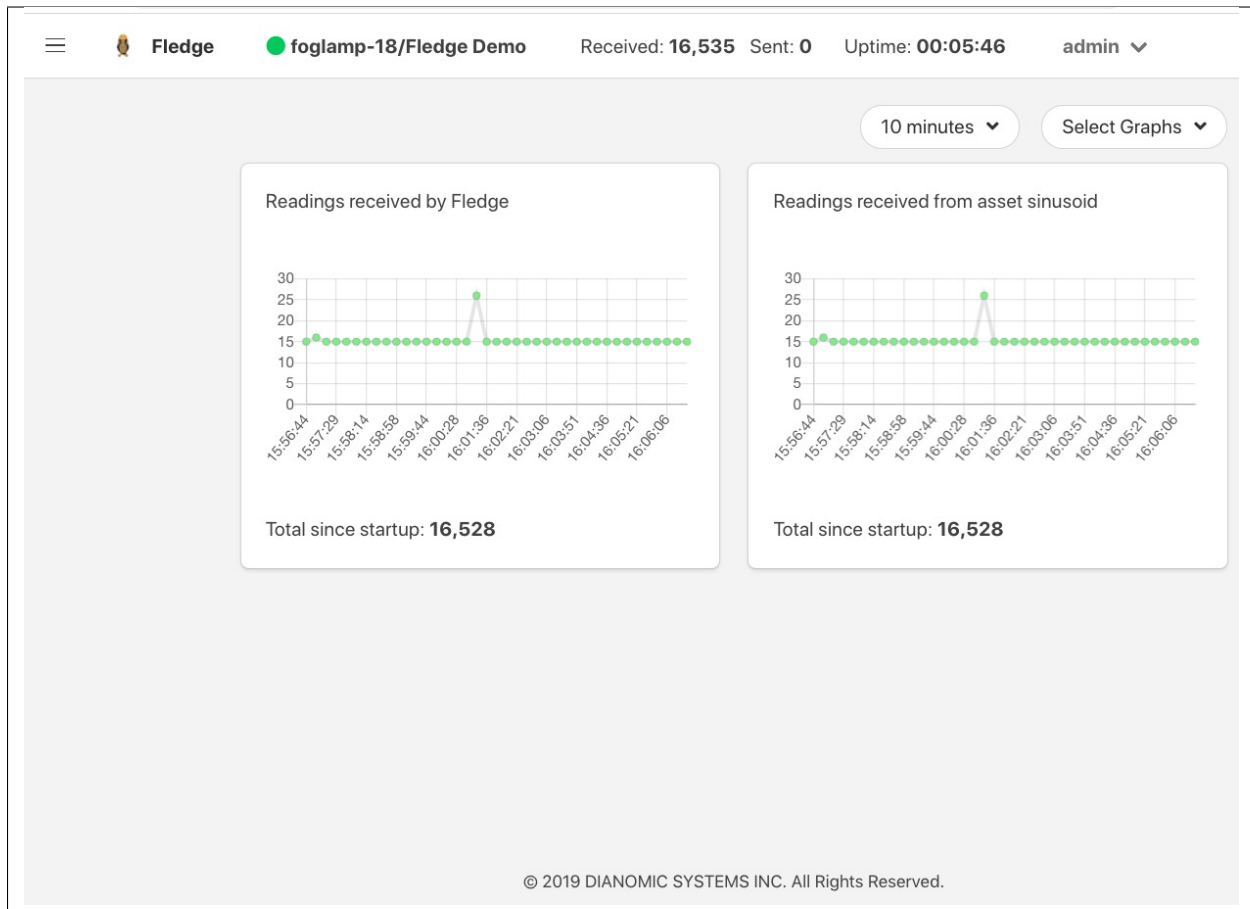
Once restarted refresh your browser page. You should be presented with a login request.



The screenshot shows the Fledge web interface. At the top, there is a navigation bar with a hamburger menu icon, the Fledge logo, a green status indicator, the text "foglamp-18/Fledge Demo", and system statistics: "Received: 16,449 Sent: 0 Uptime: 00:04:22". The main content area is a light gray background. In the center, there is a white login box containing two input fields: "username" with a user icon and "password" with a lock icon. Below these fields is a blue "Log In" button. Underneath the login box, there are three links: "Login with Certificate .", "Forgot Password?", and "Setup Instance". At the bottom of the page, there is a copyright notice: "© 2019 DIANOMIC SYSTEMS INC. All Rights Reserved."

The default username is “admin” with a password of “fledge”. Use these to login to Fledge, you should be presented with a slightly changed dashboard view.



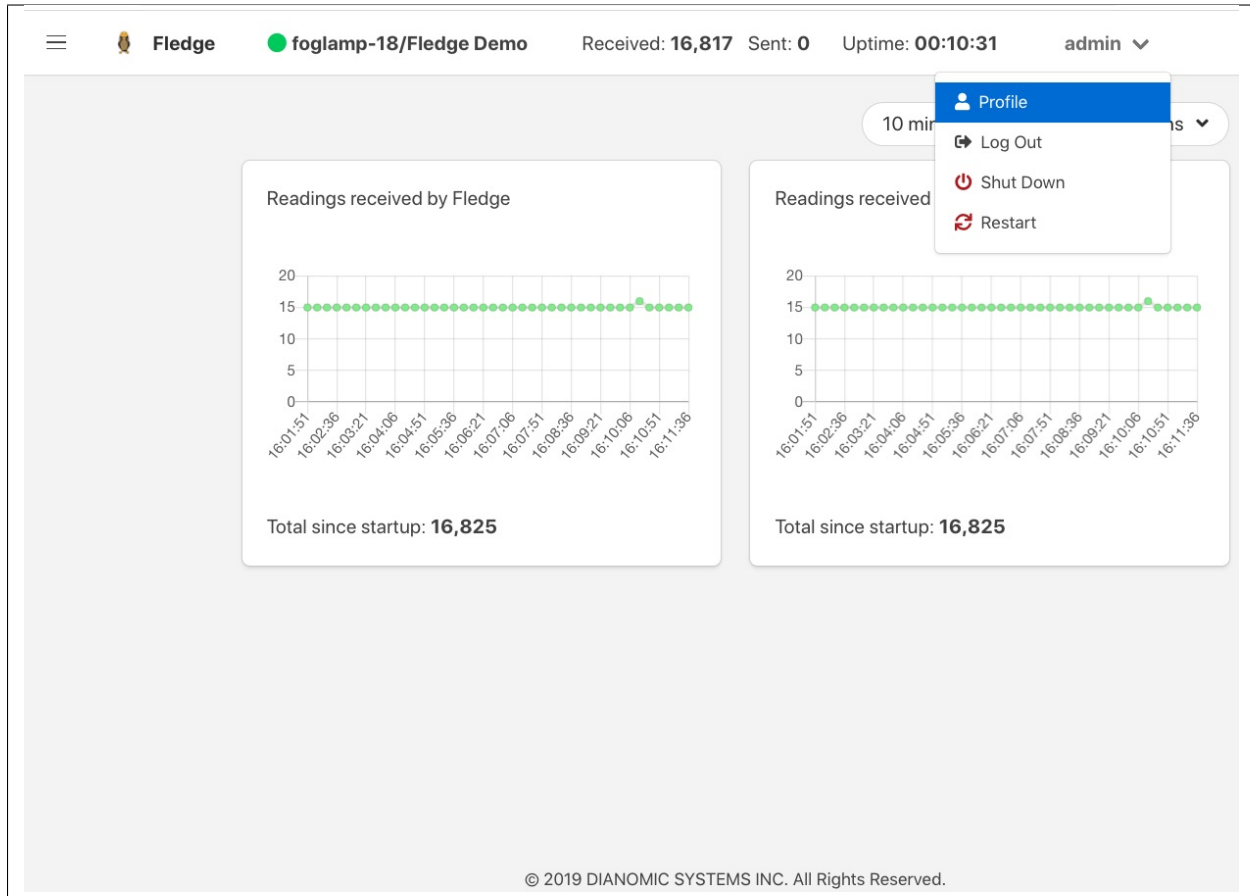


The status bar now contains the name of the user that is currently logged in and a new option has appeared in the left-hand menu, *User Management*.

### 10.2.1 Changing Your Password

The top status bar of the Fledge GUI now contains the user name on the right-hand side and a pull down arrow, selecting this arrow gives a number of options including one labeled *Profile*.

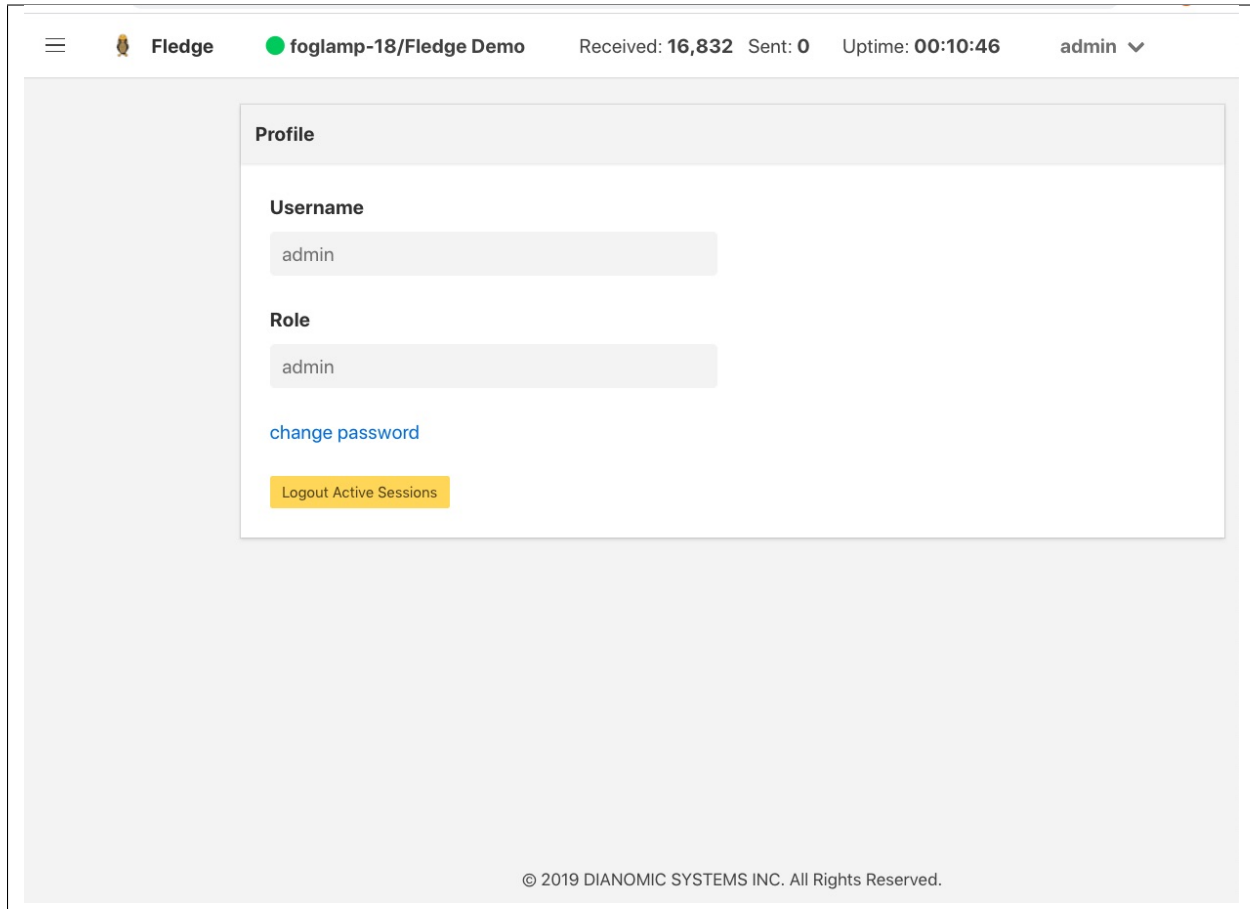




**Note:** This pulldown menu is also where the *Shutdown* and *Restart* options have moved.

Selecting the *Profile* option will display the profile for the user.





Towards the bottom of this profile display the *change password* option appears. Click on this text and a new password dialog will appear.



This popup can be used to change your password. On successfully changing your password you will be logged out of



the user interface and will be required to log back in using this new password.

## 10.2.2 Password Rotation Mechanism

Fledge provides a mechanism to limit the age of passwords in use within the system. A value for the maximum allowed age of a password is defined in the configuration page of the user interface.

The screenshot shows the Fledge user interface. The top bar displays 'Fledge', 'foglamp-18/Fledge Demo', and system statistics: 'Received: 117,981', 'Sent: 0', 'Uptime: 00:12:02', and a 'mark' dropdown. The left sidebar contains navigation links: Dashboard, Assets & Readings, South, North, Notifications, Configuration (highlighted), Schedules, Certificate Store, Backup & Restore, Logs, Audit, Notifications, Packages, System, Tasks, Support, and Settings. The main content area is titled 'Fledge Admin and User REST API' and shows configuration options for the Admin API. A 'General' tab is selected. The configuration includes: 'Enable HTTP' (checked), 'HTTP Port' (8081), 'HTTPS Port' (1995), 'Certificate Name' (fledge), 'Authentication' (mandatory), 'Authentication method' (any), 'Auth Certificate' (ca), 'Allow Ping' (checked), and 'Password Expiry Days' (30). The 'Auth Providers' section shows a JSON configuration: 

```
{
  "providers": [
    "username",
    "ldap"
  ]
}
```

Whenever a user logs into Fledge the age of their password is checked against the maximum allowed password age. If their password has reached that age then the user is not logged in, but is instead forced to enter a new password. They must then login with that new password. In addition the system maintains a history of the last three passwords the user has used and prevents them being reused.

## 10.3 User Management

Once mandatory authentication has been enabled and the currently logged in user has the role *admin*, a new option appears in the GUI, *User Management*.



The screenshot displays the Fledge User Management interface. The top navigation bar includes the Fledge logo, the instance name 'foglamp-18/Fledge Demo', and system statistics: 'Received: 18,455', 'Sent: 0', and 'Uptime: 00:37:46'. The user 'admin' is logged in. The left sidebar contains a list of navigation items, with 'User Management' currently selected. The main panel shows the 'User Management' tab, which contains a table of users. The table has columns for ID, Username, and Role. User 1 is 'admin' with role 'admin' and status 'super admin' and 'active'. User 2 is 'user' with role 'user' and actions 'change role', 'reset password', and 'delete'. An 'Add User' button is located in the top right corner of the user management panel.

ID	Username	Role
1	admin	admin
2	user	user

The user management pages allows

- Adding new users.
- Deleting users.
- Resetting user passwords.
- Changing the role of a user.

Fledge currently supports two roles for users,

- **admin:** a user with admin role is able to fully configure Fledge and also manage Fledge users
- **user:** a user with this role is able to configure Fledge but can not manage users

### 10.3.1 Adding Users

To add a new user from the *User Management* page select the *Add User* icon in the top right of the *User Management* pane. a new dialog will appear that will allow you to enter details of that user.



A screenshot of a 'Create User' dialog box. The dialog has a title bar with the text 'Create User' and a close button (an 'x' in a circle) on the right. The main area contains a section titled 'Role' with a dropdown menu showing 'user'. Below this is a text input field containing 'mark'. There are two password input fields, both masked with dots. The bottom of the dialog features a blue 'Save' button on a light gray background.

Create User

Role

user

mark

.....

.....

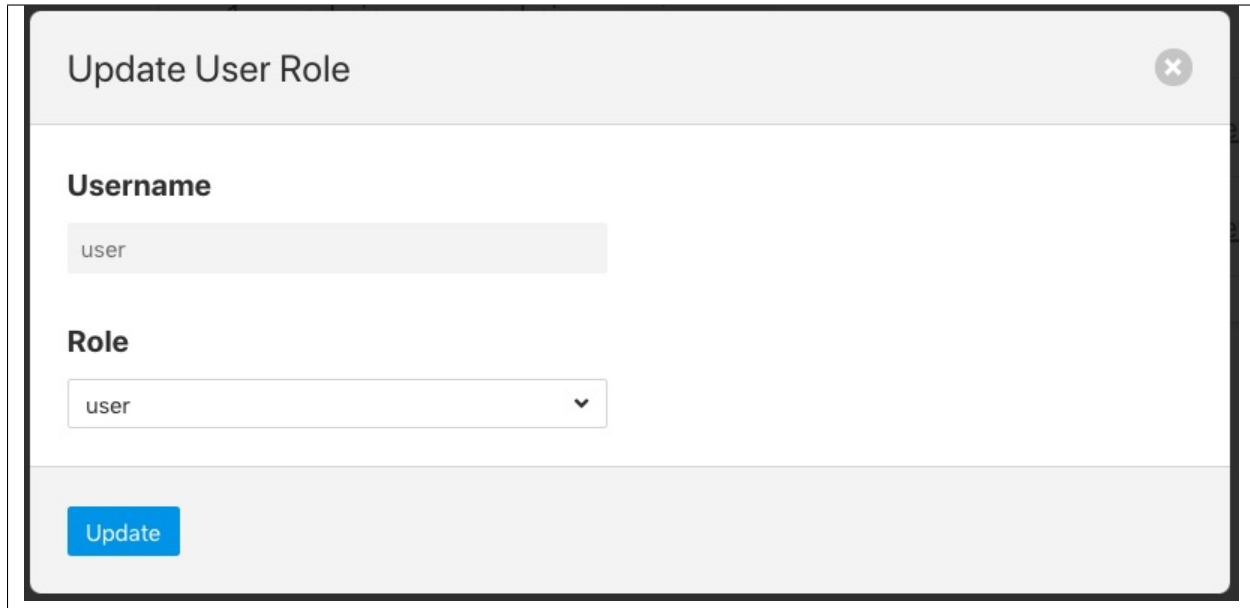
Save

You can select a role for the new user, a user name and an initial password for the user. Only users with the role *admin* can add new users.

### 10.3.2 Changing User Roles

The role that a particular user has when the login can be changed from the *User Management* page. Simply select on the *change role* link next to the user you wish to change the role of.



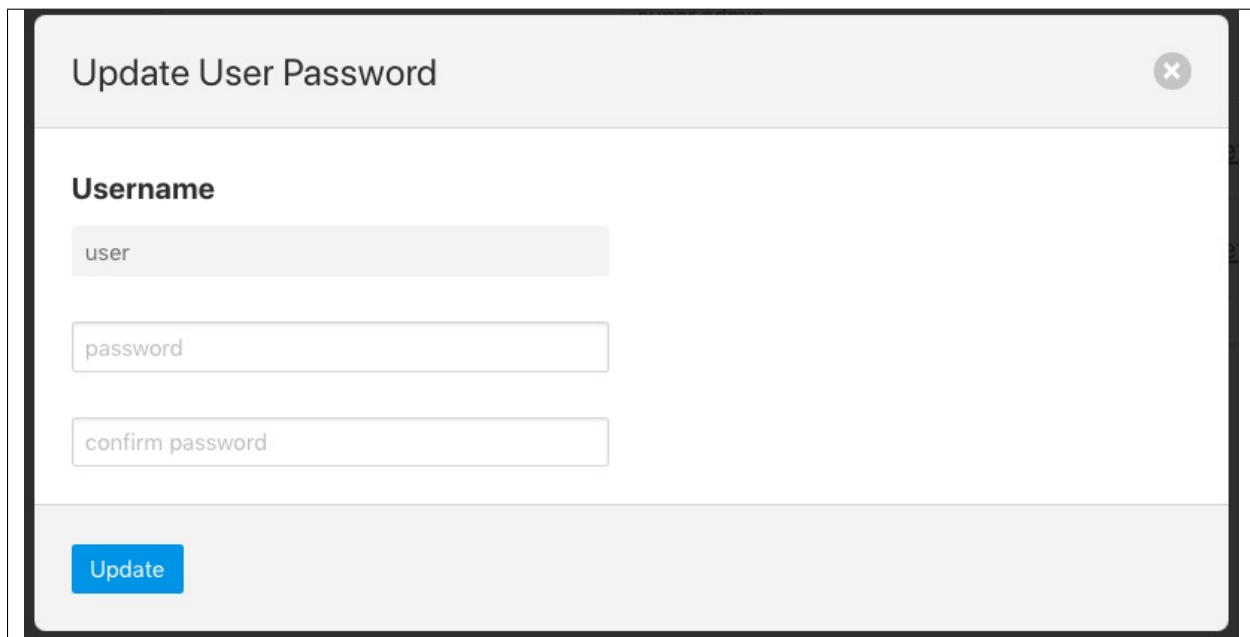


The dialog box titled "Update User Role" has a close button (X) in the top right corner. It contains two input fields: "Username" with the value "user" and "Role" with a dropdown menu showing "user". At the bottom left is a blue "Update" button.

Select the new role for the user from the drop down list and click on update. The new role will take effect the next time the user logs in.

### 10.3.3 Reset User Password

Users with the *admin* role may reset the password of other users. In the *User Management* page select the *reset password* link to the right of the user name of the user you wish to reset the password of. A new dialog will appear prompting for a new password to be created for the user.



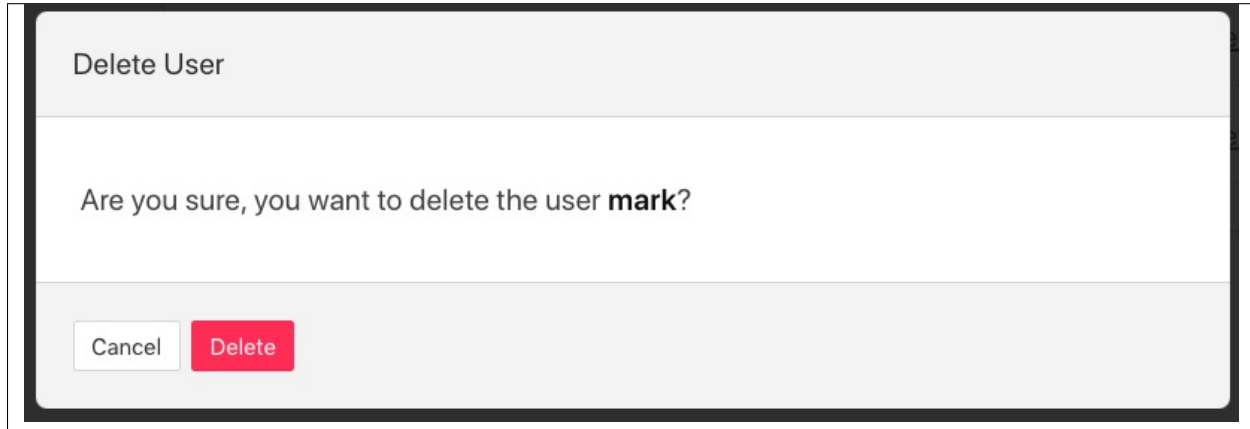
The dialog box titled "Update User Password" has a close button (X) in the top right corner. It contains three input fields: "Username" with the value "user", "password", and "confirm password". At the bottom left is a blue "Update" button.

Enter the new password and confirm that password by entering it a second time and click on *Update*.



### 10.3.4 Delete A User

Users may be deleted from the *User Management* page. Select the *delete* link to the right of the user you wish to delete. A confirmation dialog will appear. Select *Delete* and the user will be deleted.



You can not delete the last user with role *admin* as this will prevent you from being able to manage Fledge.

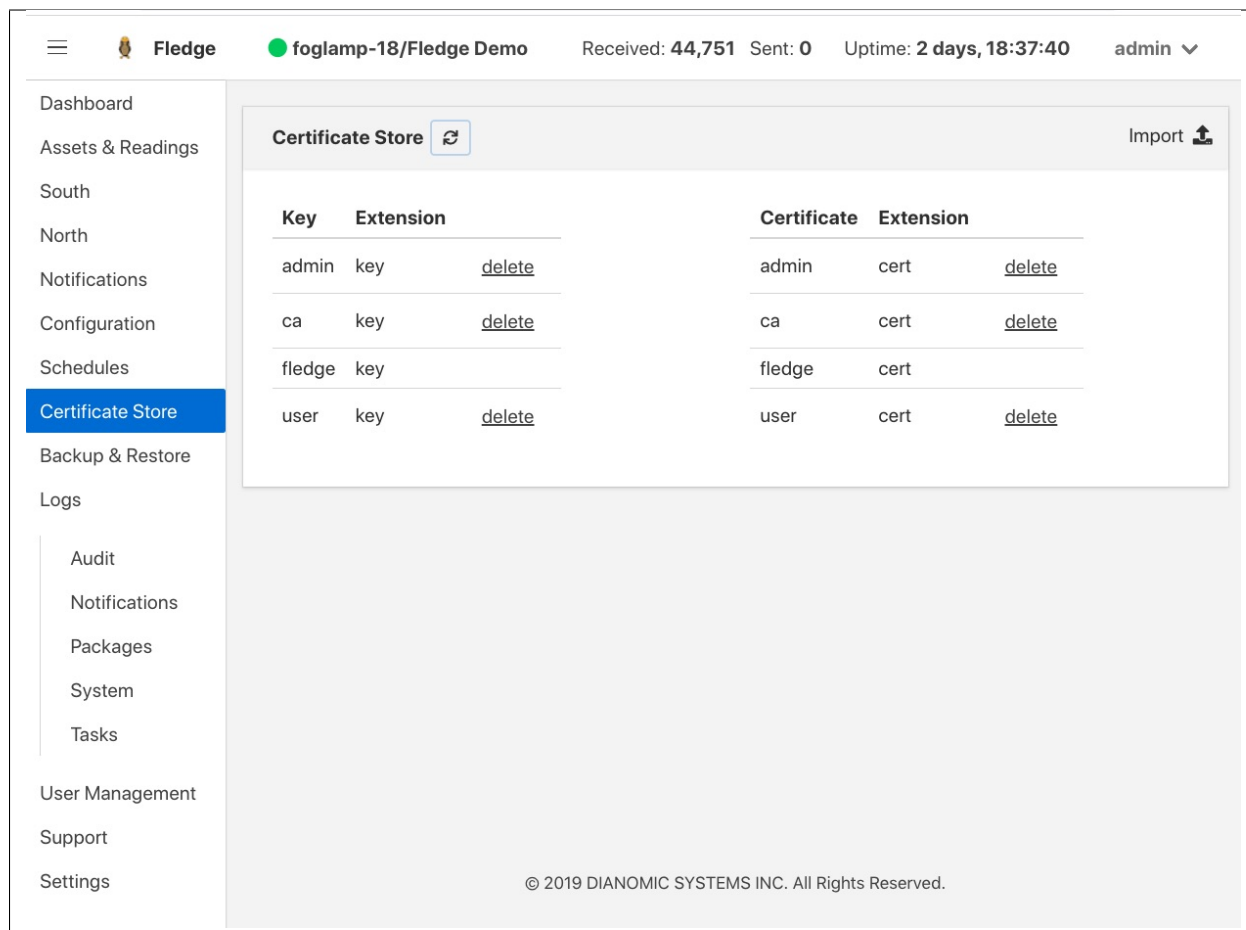
## 10.4 Certificate Store

The Fledge *Certificate Store* allows certificates to be stored that may be referenced by various components within the system, in particular these certificates are used for the encryption of the REST API traffic and authentication. They may also be used by particular plugins that require a certificate of one type or another. A number of different certificate types are supported by the certificate store;


- PEM files as created by most certificate authorities
- CRT files as used by GlobalSign, VeriSign and Thawte
- Binary CER X.509 certificates
- JSON certificates as used by Google Cloud Platform

The *Certificate Store* functionality is available in the left-hand menu by selecting *Certificate Store*. When selected it will show the current content of the store.





**Dashboard** **Assets & Readings** **South** **North** **Notifications** **Configuration** **Schedules** **Certificate Store** **Backup & Restore** **Logs** **Audit** **Notifications** **Packages** **System** **Tasks** **User Management** **Support** **Settings**

**Certificate Store** [↻](#) [Import](#) 

Key	Extension	
admin	key	<a href="#">delete</a>
ca	key	<a href="#">delete</a>
fledge	key	
user	key	<a href="#">delete</a>

Certificate	Extension	
admin	cert	<a href="#">delete</a>
ca	cert	<a href="#">delete</a>
fledge	cert	
user	cert	<a href="#">delete</a>

© 2019 DIANOMIC SYSTEMS INC. All Rights Reserved.

Certificates may be removed by selecting the delete option next to the certificate name, note that the keys and certificates can be deleted independently. The self signed certificate that is created at installation time can not be deleted.

To add a new certificate select the *Import* icon in the top right of the certificate store display.



**Upload Certificate**

**Key**

Choose file No file chosen

**Certificate**

Choose file No file chosen

☐ Overwrite

Overwrite allows to replace the existing key or certificate

**Import**

A dialog will appear that allows a key file and/or a certificate file to be selected and uploaded to the *Certificate Store*. An option allows to allow overwrite of an existing certificate. By default certificates may not be overwritten.







## TUNING FLEDGE

Many factors will impact the performance of a Fledge system

- The CPU, memory and storage performance of the underlying hardware
- The communication channel performance to the sensors
- The communications to the north systems
- The choice of storage system
- The external demands via the public REST API

Many of these are outside of the control of Fledge itself, however it is possible to tune the way Fledge will use certain resources to achieve better performance within the constraints of a deployment environment.

### 11.1 South Service Advanced Configuration

The south services within Fledge each have a set of advanced configuration options defined for them. These are accessed by editing the configuration of the south service itself. A screen with a set of tabbed panes will appear, select the tab labeled *Advanced Configuration* to view and edit the advanced configuration options.



lathe1004 South Service

Configuration **Advanced Configuration** Security Configuration

Maximum Reading Latency (mS) 5000

Maximum buffered Readings 100

Reading Rate 1

Throttle ☐

Reading Rate Per second

Minimum Log Level warning

Enabled ☒

Applications

Cancel Save

Service Info http://localhost:45437

Export Readings Delete Service

- *Maximum Reading Latency (mS)* - This is the maximum period of time for which a south service will buffer a reading before sending it onward to the storage layer. The value is expressed in milliseconds and it effectively defines the maximum time you can expect to wait before being able to view the data ingested by this south service.
- *Maximum buffered Readings* - This is the maximum number of readings the south service will buffer before attempting to send those readings onward to the storage service. This and the setting above work together to define the buffering strategy of the south service.
- *Reading Rate* - The rate at which polling occurs for this south service. This parameter only has effect if your south plugin is polled, asynchronous south services do not use this parameter. The units are defined by the setting of the *Reading Rate Per* item.
- *Throttle* - If enabled this allows the reading rate to be throttled by the south service. The service will attempt to poll at the rate defined by *Reading Rate*, however if this is not possible, because the readings are being forwarded out of the south service at a lower rate, the reading rate will be reduced to prevent the buffering in the south service from becoming overrun.
- *Reading Rate Per* - This defines the units to be used in the *Reading Rate* value. It allows the selection of per *second*, *minute* or *hour*.
- *Minimum Log Level* - This configuration option can be used to set the logs that will be seen for this service. It defines the level of logging that is sent to the syslog and may be set to *error*, *warning*, *info* or *debug*. Logs of the level selected and higher will be sent to the syslog. You may access the contents of these logs by selecting



the log icon in the bottom left of this screen.

### 11.1.1 Tuning Buffer Usage

The tuning of the south service allows the way the buffering is used within the south service to be controlled. Setting the latency value low results in frequent calls to send data to the storage service and therefore means data is more quickly available. However sending small quantities of data in each call the the storage system does not result in the most optimal use of the communications or of the storage engine itself. Setting a higher latency value results in more data being sent per transaction with the storage system and a more efficient system. The cost of this is the requirement for more in-memory storage within the south service.

Setting the *Maximum buffers Readings* value allows the user to place a cap on the amount of memory used to buffer within the south service, since when this value is reach, regardless of the age of the data and the setting of the latency parameter, the data will be sent to the storage service. Setting this to a smaller value allows tighter control on the memory footprint at the cost of less efficient use of the communication and storage service.

Tuning between performance, latency and memory usage is always a balancing act, there are situations where the performance requirements mean that a high latency will need to be incurred in order to make the most efficient use of the communications between the micro services and the transnational performance of the storage engine. Likewise the memory resources available for buffering may restrict the performance obtainable.

## 11.2 North Advanced Configuration

In a similar way to the south services, north services and tasks also have advanced configuration that can be used to tune the operation of the north side of Fledge. The north advanced configuration is accessed in much the same way as the south, select the North page and open the particular north service or task. A tabbed screen will be shown which contains an *Advanced Configuration* tab.

The screenshot shows the 'OMF North Service' configuration window. It has three tabs: 'Configuration', 'Advanced Configuration' (which is selected), and 'Security Configuration'. Under the 'Advanced Configuration' tab, there are two settings: 'Minimum Log Level' set to 'warning' and 'Data block size' set to '100'. Below these is an 'Enabled' checkbox which is currently unchecked. At the bottom of the configuration area is a section labeled 'Applications' with a plus icon. At the very bottom of the window are three buttons: a help icon (?), a 'Delete Service' button, and 'Cancel' and 'Save' buttons.



- *Minimum Log Level* - This configuration option can be used to set the logs that will be seen for this service or task. It defines the level of logging that is sent to the syslog and may be set to *error*, *warning*, *info* or *debug*. Logs of the level selected and higher will be sent to the syslog. You may access the contents of these logs by selecting the log icon in the bottom left of this screen.
- *Data block size* - This defines the number of readings that will be sent to the north plugin for each call to the *plugin\_send* entry point. This allows the performance of the north data pipeline to be adjusted, with larger blocks sizes increasing the performance, by reducing overhead, but at the cost of requiring more memory in the north service or task to buffer the data as it flows through the pipeline. Setting this value too high may cause issues for certain of the north plugins that have limitations on the number of messages they can handle within a single block.

## 11.3 Health Monitoring

The Fledge core monitors the health of other services within Fledge, this is done with the *Service Monitor* within Fledge and can be configured via the *Configuration* menu item in the Fledge user interface. In the configuration page select the *Advanced* options and then the *Service Monitor* section.

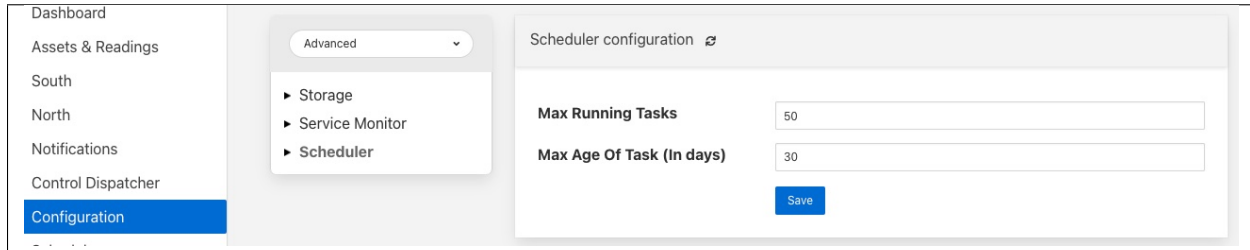
The screenshot shows the Fledge Configuration page. On the left is a sidebar menu with items: Dashboard, Assets & Readings, South, North, Notifications, Control Dispatcher, Configuration (highlighted), Schedules, Certificate Store, Backup & Restore, and Logs. The main content area has a top bar with 'Advanced' and a dropdown arrow. Below this is a 'Service Monitor' section with a link icon. The settings include: 'Health Check Interval (In seconds)' set to 5, 'Ping Timeout' set to 1, 'Max Attempts To Check Heartbeat' set to 15, and 'Restart Failed' set to 'auto' with a dropdown arrow. A 'Save' button is at the bottom right of the settings panel.

- *Health Check Interval* - This setting determines how often Fledge will send a health check request to each of the microservices within the Fledge instance. The value is expressed in seconds. Making this value small will decrease the amount of time it will take to detect a failure, but will increase the load on the system for performing health checks. Making this too frequent is likely to increase the occurrence of false failure detection.
- *Ping Timeout* - Amount of time to wait, in seconds, before declaring that a health check request has failed. Failure for a health check response to be seen within this time will make a service as unresponsive. Small values can result in busy services becoming suspect erroneously.
- *Max Attempts To Check Heartbeat* - This is the number of heartbeat requests that must fail before the core determines that the service has failed and attempts any restorative action. Reducing this value will cause the service to be declared as failed sooner and hence recovery can be performed sooner. If this value is too small then it can result in multiple instances of a service running or frequent restarts occurring. Making this too long results in loss of data.
- *Restart Failed* - Determine what action should be taken when a service is detected as failed. Two options are available, *Manual*, in which case not automatic action will be taken, or *Auto*, in which case the service will be automatically restarted.



## 11.4 Scheduler

The Fledge core contains a scheduler that is used for running periodic tasks, this scheduler has a couple of tuning parameters. To access these parameters from the Fledge User Interface, in the configuration page select the *Advanced* options and then the *Scheduler* section.



- *Max Running Tasks* - Specifies the maximum number of tasks that can be running at any one time. This parameter is designed to stop runaway tasks adversely impacting the performance of the system. When this number is reached no new tasks will be created until one or more of the currently running tasks terminated. Set this too low and you will not be able to run all the task you require in parallel. Set it too high and the system is more at risk from runaway tasks.
- *Max Age of Task* - Specifies, in days, how long a task can run for. Tasks that run longer than this will be killed by the system.

**Note:** Individual tasks have a setting that they may use to stop multiple instances of the same task running in parallel. This also helps protect the system from runaway tasks.

## 11.5 Storage

The storage layer is perhaps one of the areas that most impacts the overall performance of the Fledge instance as it is the end point for the data pipelines; the location at which all ingest pipelines in the south terminate and the point of origin for all north pipelines to external systems.

The storage system in Fledge serves two purposes

- The storage of configuration and persistent state of Fledge itself
- The buffering of reading data as it traverses the Fledge instance

The physical storage is managed by plugins that are loaded dynamically into the storage service in the same way as with other services in Fledge. In the case of the storage service it may have either one or two plugins loaded. If a single plugin is loaded this will be used for the storage of both configuration and readings; if two plugins are loaded then one will be used for storing the configuration and the other for storing the readings. Not all plugins support both classes of data.



### 11.5.1 Choosing A Storage Plugin

Fledge comes with a number of storage plugins that may be used, each one has its benefits and limitations, below is an overview of each of the plugins that are currently included with Fledge.

**sqlite** The default storage plugin that is used. It is implemented using the *SQLite* database and is capable of storing both configuration and reading data. It is optimized to allow parallelism when multiple assets are being ingested into the Fledge instance. It does however have limitations on the number of different assets that can be ingested within an instance. The precise limit is dependent upon a number of other factors, but is of the order of 900 unique asset names per instance. This is a good general purpose storage plugin and can manage reasonably high rates of data reading.

**sqlitelb** This is another *SQLite* based plugin able to store both readings and configuration data. It is designed for lower bandwidth data, hence the name suffix *lb*. It does not have the same parallelism optimization as the default *sqlite* plugin, and is therefore less good when high rate data spread across multiple assets is being ingested. However it does perform well when ingesting high rates of a single asset or low rates of a very large number of assets. It does not have any limitations on the number of different assets that can be stored within the Fledge instance.

**sqlitememory** This is a *SQLite* based plugin that uses in memory tables and can only be used to store reading data, it must be used in conjunction with another plugin that will be used to store the configuration. Reading data is stored in tables in memory and thus very high bandwidth data can be supported. If Fledge is shutdown however the data stored in these tables will be lost.

**postgres** This plugin is implemented using the *PostgreSQL* database and supports the storage of both configuration and reading data. It uses the standard Postgres storage engine and benefits from the additional features of Postgres for security and replication. It is capable of high levels of concurrency however has slightly less overall performance than the *sqlite* plugins. Postgres also does not work well with certain types of storage media, such as SD cards as it has a higher wear rate on the media.

In most cases the default *sqlite* storage plugin is perfectly acceptable, however if very high data rates, or huge volumes of data (i.e. large images at a reasonably high rate) are ingested this plugin can start to exhibit issues. This usually exhibits itself by large queues building in the south service or in extreme cases by transaction failure messages in the log for the storage service. If this happens then the recommended course of action is to either switch to a plugin that stores data in memory rather than on external storage, *sqlitememory*, or investigate the media where the data is stored. Low performance storage will adversely impact the *sqlite* plugin.

The *sqlite* plugin may also prove less than optimal if you are ingesting many hundreds of different assets in the same Fledge instance. The *sqlite* plugin has been optimized to allow concurrent south services to write to the storage in parallel. This is done by the use of multiple databases to improve the concurrency, however there is a limit, imposed by the number of open databases that can be supported. If this limit is exceeded it is recommended to switch to the *sqlitelb* plugin. There are configuration options regarding how these databases are used that can change the point at which it becomes necessary to switch to the other plugin.



## Configuring Storage Plugins

The storage plugins to use can be selected in the *Advanced* section of the *Configuration* page. Select the *Storage* category from the category tree display and the following will be displayed.

The screenshot shows the Fledge Configuration page in the 'Advanced' section. The 'Storage configuration' tab is active. The left sidebar contains a navigation menu with 'Configuration' highlighted. The main content area displays the 'Storage configuration' form with the following fields:

- Storage Plugin:** A text input field containing 'sqlite'.
- Readings Plugin:** An empty text input field.
- Database threads:** A text input field containing '1'.
- Manage Storage:** A checkbox that is currently unchecked.
- Service Port:** A text input field containing '0'.
- Management Port:** A text input field containing '0'.
- Save:** A blue button at the bottom right of the form.

- **Storage Plugin:** The name of the storage plugin to use. This will be used to store the configuration data and must be one of the supported storage plugins.

**Note:** This can not be the *sqlitememory* plugin as that plugin does not support the storage of configuration.

- **Reading Plugin:** The name of the storage plugin that will be used to store the readings data. If left blank then the *Storage Plugin* above will be used to store both configuration and readings.
- **Database threads:** Increase the number of threads used within the storage service to manage the database activity. This is not the number of threads that can be used to read or write the database and increasing this will not improve the throughput of the data.
- **Manage Storage:** This is used when an external storage application, such as the Postgres database is used that requires separate initialization. If this external process is not run by default setting this to true will cause Fledge to start the storage process. Normally this is not required as Postgres should be run as a system service and SQLite does not require it.
- **Service Port:** Normally the storage service will dynamically create a service port that will be used by the storage service. Setting this to a value other than 0 will cause a fixed port to be used. This can be useful when developing a new storage plugin or to allow access to a non-fledge application to the storage layer. This should only be changed with extreme caution.
- **Management Port:** Normally the storage service will dynamically create a management port that will be used by the storage service. Setting this to a value other than 0 will cause a fixed port to be used. This can be useful when developing a new storage plugin.

Changing will be saved once the *save* button is pressed. Fledge uses a mechanism whereby this data is not only saved in the configuration database, but also cached to a file called *storage.json* in the *etc* directory of the data directory. This is required such that Fledge can find the configuration database during the boot process. If the configuration becomes corrupt for some reason simply removing this file and restarting Fledge will cause the default configuration to be restored. The location of the Fledge data directory will depend upon how you installed Fledge and the environment variables used to run Fledge.

- Installation from a package will usually put the data directory in */usr/local/fledge/data*. However this can be overridden by setting the *\$FLEDGE\_DATA* environment variable to point at a different location.
- When running a copy of Fledge built from source the data directory can be found in *\${FLEDGE\_ROOT}/data*. Again this may be overridden by setting the *\$FLEDGE\_DATA* environment variable.



**Note:** When changing the storage service a reboot of the Fledge instance is required before the new storage plugins will be used. Also, data is not migrated from one plugin to another and hence if there is unsent data within the database this will be lost when changing the storage plugin. The `sqlite` and `sqlitelb` plugin however share the same configuration data tables and hence configuration will be preserved when changing between these databases but reading data will not.

## sqlite Plugin Configuration

The storage plugin configuration can be found in the *Advanced* section of the *Configuration* page. Select the *Storage* category from the category tree display and the plugin name from beneath that category. In the case of the *sqlite* storage plugin the following will be displayed.

The screenshot shows the Fledge Configuration page. On the left is a sidebar menu with options: Dashboard, Assets & Readings, South, North, Notifications, Control Dispatcher, Configuration (highlighted), Schedules, Certificate Store, Backup & Restore, Logs, Audit, and Notifications. The main content area is titled 'Storage Plugin' and contains a dropdown menu set to 'Advanced'. Below this is a tree view with 'Storage' expanded, showing 'sqlite', 'Service Monitor', and 'Scheduler'. The 'sqlite' plugin is selected, and its configuration is displayed in a form with the following fields: 'Pool Size' (5), 'No. Readings per database' (15), 'No. databases to allocate in advance' (3), 'Database allocation threshold' (1), 'Database allocation size' (2), and 'Purge Exclusions' (empty). A 'Save' button is at the bottom right of the form.

- **Pool Size:** The storage service uses a connection pool to communicate with the underlying database, it is this pool size that determines how many parallel operations can be invoked on the database.

This pool size is only the initial size, the storage service will grow the pool if required, however setting a realistic initial pool size will improve the ramp up performance of Fledge.

**Note:** Although the pool size denotes the number of parallel operations that can take place, database locking considerations may reduce the number of actual operations in progress at any point in time.

- **No. Readings per database:** The *sqlite* plugin support multiple readings databases, with the name of the asset used to determine which database to store the readings in. This improves the level of parallelism by reducing the lock contention when data is being written. Setting this value to 1 will cause only a single asset name to be stored within a single readings database, resulting in no contention between assets. However there is a limit on the number of databases, therefore setting this to 1 will limit the number of different assets that can be ingested into the instance.
- **No. databases to allocate in advance:** This controls how many reading databases Fledge should initially created. Creating databases is a slow process and thus is best achieved before data starts to flow through Fledge. Setting this too high will cause Fledge to allocate a large number of databases than required and waste open database connections. Ideally set this to the number of different assets you expect to ingest divided by the number of readings per database configuration above. This should give you sufficient databases to store the data you require.
- **Database allocation threshold:** The allocation of a new database is a slow process, therefore rather than wait until there are no available databases before allocating new ones, it is possible to pre-allocate database as the

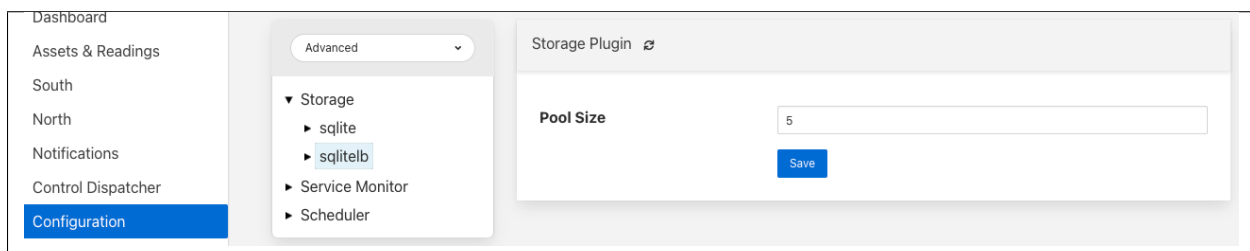


number of free databases becomes low. This value allows you to set the point at which to allocation more databases. As soon as the number of free databases declines to this value the plugin will allocate more databases.

- **Database allocation size:** The number of new databases to create whenever an allocation occurs. This effectively denotes the size of the free pool of databases that should be created.
- **Purge Exclusion:** This is not a performance settings, but allows a number of assets to be exempted from the purge process. This value is a comma separated list of asset names that will be excluded from the purge operation.

## sqlitelb Configuration

The storage plugin configuration can be found in the *Advanced* section of the *Configuration* page. Select the *Storage* category from the category tree display and the plugin name from beneath that category. In the case of the *sqlitelb* storage plugin the following will be displayed.



**Note:** The *sqlite* configuration is still present and selectable since this instance has run that storage plugin in the past and the configuration is preserved when switching between *sqlite* and *sqlitelb* plugins.

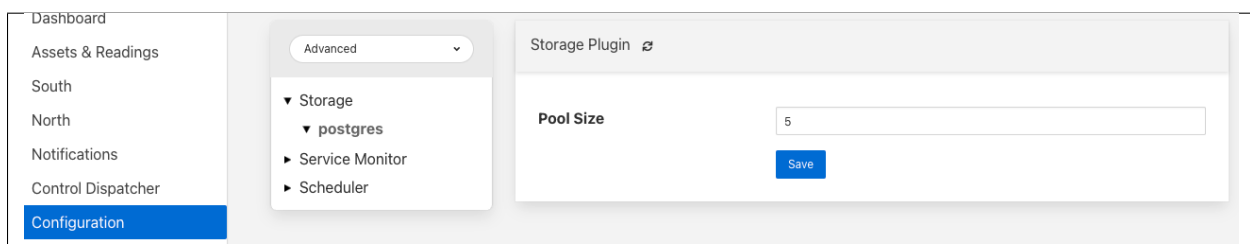
- **Pool Size:** The storage service uses a connection pool to communicate with the underlying database, it is this pool size that determines how many parallel operations can be invoked on the database.

This pool size is only the initial size, the storage service will grow the pool if required, however setting a realistic initial pool size will improve the ramp up performance of Fledge.

**Note:** Although the pool size denotes the number of parallel operations that can take place, database locking considerations may reduce the number of actual operations in progress at any point in time.

## postgres Configuration

The storage plugin configuration can be found in the *Advanced* section of the *Configuration* page. Select the *Storage* category from the category tree display and the plugin name from beneath that category. In the case of the *postgres* storage plugin the following will be displayed.





- **Pool Size:** The storage service uses a connection pool to communicate with the underlying database, it is this pool size that determines how many parallel operations can be invoked on the database.

This pool size is only the initial size, the storage service will grow the pool if required, however setting a realistic initial pool size will improve the ramp up performance of Fledge.

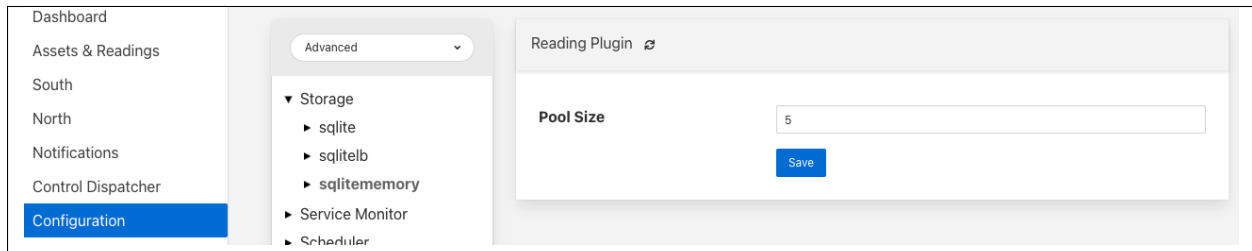
---

**Note:** Although the pool size denotes the number of parallel operations that can take place, database locking considerations may reduce the number of actual operations in progress at any point in time.

---

### sqlitememory Configuration

The storage plugin configuration can be found in the *Advanced* section of the *Configuration* page. Select the *Storage* category from the category tree display and the plugin name from beneath that category. Since this plugin only supports the storage of readings there will always be at least one other reading plugin displayed. Selecting the *sqlitememory* storage plugin the following will be displayed.



- **Pool Size:** The storage service uses a connection pool to communicate with the underlying database, it is this pool size that determines how many parallel operations can be invoked on the database.

This pool size is only the initial size, the storage service will grow the pool if required, however setting a realistic initial pool size will improve the ramp up performance of Fledge.

---

**Note:** Although the pool size denotes the number of parallel operations that can take place, database locking considerations may reduce the number of actual operations in progress at any point in time.

---



## TROUBLESHOOTING THE PI SERVER INTEGRATION

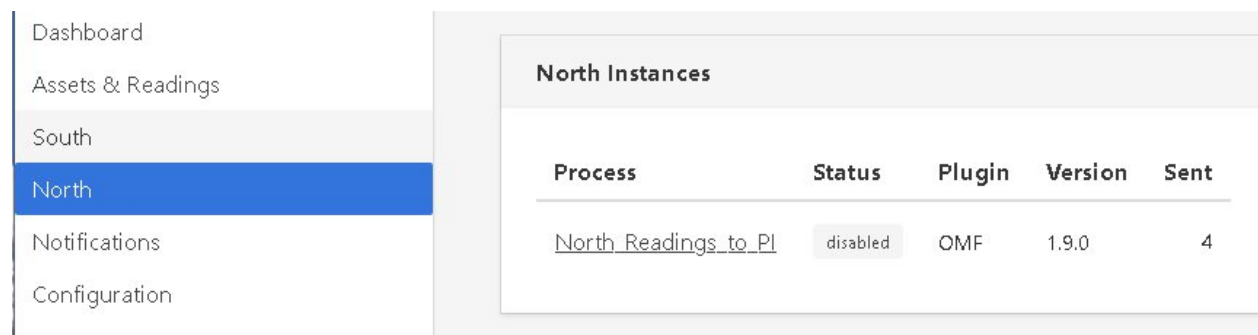
This section describes how to troubleshoot issues with the PI Server integration using Fledge version  $\geq 1.9.1$  and PI Web API 2019 SP1 1.13.0.6518

- *Log files*
- *How to check the PI Web API is installed and running*
- *Commands to check the PI Web API*
- *Error messages and causes*
- *Possible solutions to common problems*

### 12.1 Log files

Fledge logs messages at error and warning levels by default, it is possible to increase the verbosity of messages logged to include information and debug messages also. This is done by altering the minimum log level setting for the north service or task. To change the minimal log level within the graphical user interface select the north service or task, click on the advanced settings link and then select a new minimal log level from the option list presented. The name of the north instance should be used to extract just the logs about the PI Server integration, as in this example:

screenshot from the Fledge GUI



```
$ sudo cat /var/log/syslog | grep North_Readings_to_PI
```

Sample message:

```
user.info, 6,1,Mar 15 08:29:57,localhost,Fledge, North_Readings_to_PI[15506]: INFO: SendingProcess  
is starting
```

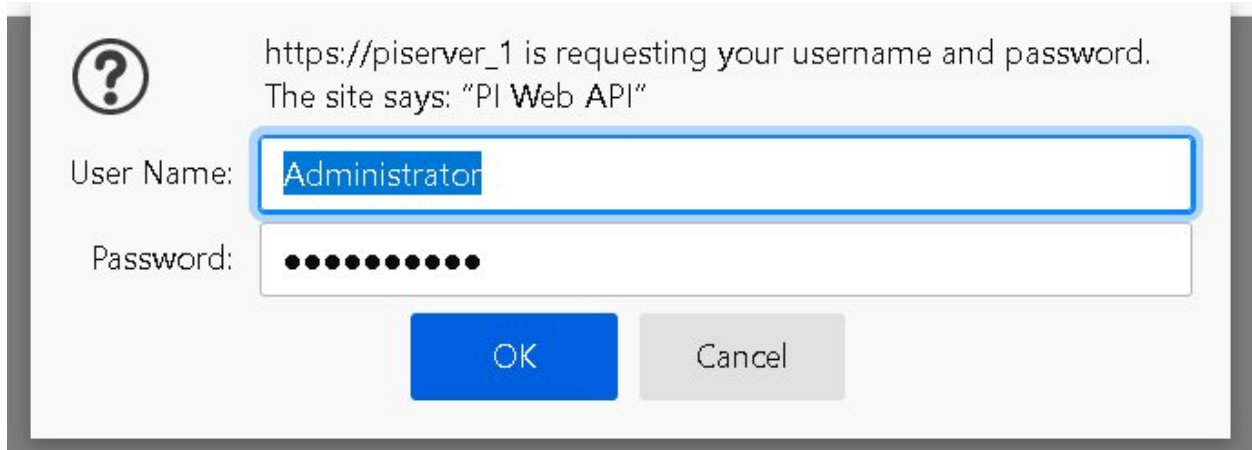
Another sample message:



North\_Readings\_to\_PI[20884]: WARNING: Error in retrieving the PIWebAPI version, The PI Web API server is not reachable, verify the network reachability

## 12.2 How to check the PI Web API is installed and running

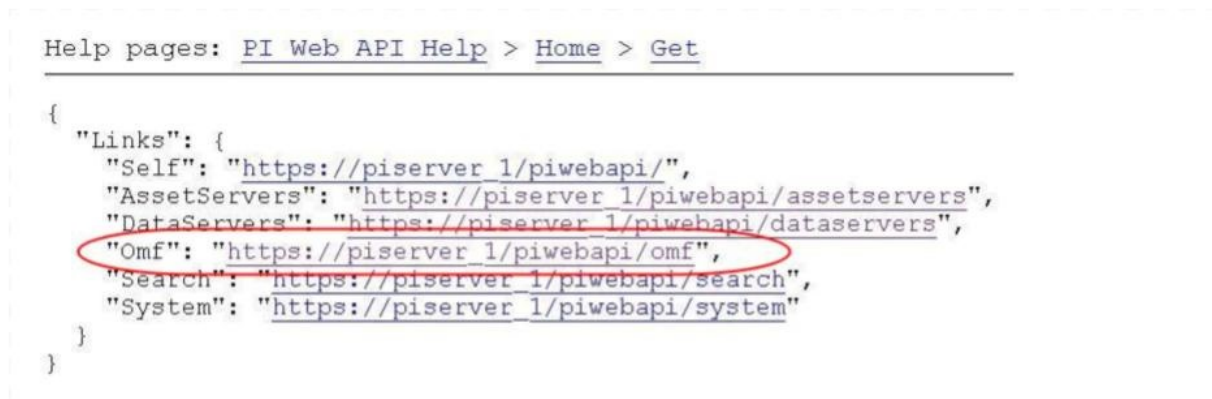
Open the URL `https://piserver_1/piwebapi` in the browser, substituting `piserver_1` with the name/address of your PI Server, to verify the reachability and proper installation of PI Web API. If PI Web API is configured for Basic authentication a prompt, similar to the one shown below, requesting entry of the user name and password will be displayed



### NOTE:

- Enter the user name and password which you set in your Fledge configuration.

The *PI Web API OMF* plugin must be installed to allow the integration with Fledge, in this screenshot the 4th row shows the proper installation of the plugin:



Select the item *System* to verify the installed version:







Help pages: [PI Web API Help](#) > [DataServer](#) > [GetPoints](#)

```
{
  "Links": {},
  "Items": [
    {
      "WebId": "F1Dq8e26YHa2kewUjKX4KLfwbwAAV010LTHy6t1MVTFTMDVQXQyNcHwMDUIMdc5NzcwOTQ4ODBMU1PQVNVKVNBU5U0FTU0VUXzE",
      "Id": 2935,
      "Name": "4273005507977094880_measurement_asset_1",
      "Path": "\\WIN-3229M01S05P\\4273005507977094880_measurement_asset_1",
      "PointClass": "classic",
      "PointType": "Float64",
      "DigitalSetName": "",
      "EngineeringUnits": "",
      "Span": 100.0,
      "Zero": 0.0,
      "Step": false,
      "Future": false,
      "DisplayDigits": -5,
      "Links": {
        "Self": "https://piserver 1/piwebapi/points/F1Dq8e26YHa2kewUjKX4KLfwbwAAV010LTHy6t1MVTFTMDVQXQyNcHwMDUIMdc5NzcwOTQ4ODBMU1PQVNVKVNBU5U0FTU0VUXzE",
        "DataServer": "https://piserver 1/piwebapi/dataservers/F1Dq8e26YHa2kewUjKX4KLfwbwAAV010LTHy6t1MVTFTMDVQXQyNcHwMDUIMdc5NzcwOTQ4ODBMU1PQVNVKVNBU5U0FTU0VUXzE",
        "Attributes": "https://piserver 1/piwebapi/points/F1Dq8e26YHa2kewUjKX4KLfwbwAAV010LTHy6t1MVTFTMDVQXQyNcHwMDUIMdc5NzcwOTQ4ODBMU1PQVNVKVNBU5U0FTU0VUXzE/attributes",
        "InterpolatedData": "https://piserver 1/piwebapi/streams/F1Dq8e26YHa2kewUjKX4KLfwbwAAV010LTHy6t1MVTFTMDVQXQyNcHwMDUIMdc5NzcwOTQ4ODBMU1PQVNVKVNBU5U0FTU0VUXzE/interpolated",
        "RecordedData": "https://piserver 1/piwebapi/streams/F1Dq8e26YHa2kewUjKX4KLfwbwAAV010LTHy6t1MVTFTMDVQXQyNcHwMDUIMdc5NzcwOTQ4ODBMU1PQVNVKVNBU5U0FTU0VUXzE/recorded",
        "PlotData": "https://piserver 1/piwebapi/streams/F1Dq8e26YHa2kewUjKX4KLfwbwAAV010LTHy6t1MVTFTMDVQXQyNcHwMDUIMdc5NzcwOTQ4ODBMU1PQVNVKVNBU5U0FTU0VUXzE/plot",
        "SummaryData": "https://piserver 1/piwebapi/streams/F1Dq8e26YHa2kewUjKX4KLfwbwAAV010LTHy6t1MVTFTMDVQXQyNcHwMDUIMdc5NzcwOTQ4ODBMU1PQVNVKVNBU5U0FTU0VUXzE/summary",
        "Value": "https://piserver 1/piwebapi/streams/F1Dq8e26YHa2kewUjKX4KLfwbwAAV010LTHy6t1MVTFTMDVQXQyNcHwMDUIMdc5NzcwOTQ4ODBMU1PQVNVKVNBU5U0FTU0VUXzE/value",
        "EndValue": "https://piserver 1/piwebapi/streams/F1Dq8e26YHa2kewUjKX4KLfwbwAAV010LTHy6t1MVTFTMDVQXQyNcHwMDUIMdc5NzcwOTQ4ODBMU1PQVNVKVNBU5U0FTU0VUXzE/end"
      }
    },
    {
      "WebId": "F1Dq8e26YHa2kewUjKX4KLfwbwAAV010LTHy6t1MVTFTMDVQXQyNcHwMDUIMdc5NzcwOTQ4ODBMU1PQVNVKVNBU5U0FTU0VUXzE",
      "Id": 2936,
      "Name": "4273005507977094880_measurement_asset_2.pl",
      "Path": "\\WIN-3229M01S05P\\4273005507977094880_measurement_asset_2.pl"
    }
  ]
}
```

### Asset Framework drill down

Following the path *AssetServers* -> Select the *Instance* -> Select the proper *Databases* -> drill down into the AF hierarchy up to the required level -> *Elements*:

Help pages: [PI Web API Help](#) > [Home](#) > [Get](#)

```
{
  "Links": {
    "Self": "https://piserver 1/piwebapi/",
    "AssetServers": "https://piserver 1/piwebapi/assetservers",
    "DataServers": "https://piserver 1/piwebapi/dataservers",
    "Omf": "https://piserver 1/piwebapi/omf",
    "Search": "https://piserver 1/piwebapi/search",
    "System": "https://piserver 1/piwebapi/system"
  }
}
```

*selecting the instance*



Help pages: [PI Web API Help](#) > [AssetServer](#) > [List](#)

```

{
  "Links": {},
  "Items": [
    {
      "WebId": "FIR5Ygb8SvdWbkmg4hgEtbBKFgV010LTHyMjMNVFTMDVQ",
      "Id": "4afc0662-56f7-496e-aa2-1804b5b04a16",
      "Name": "WIN-3228MU1805P",
      "Description": "",
      "Path": "\\WIN-3228MU1805P",
      "IsConnected": true,
      "ServerVersion": "2.10.8.440",
      "ServerTime": "2021-03-15T13:27:31.0903158Z",
      "ExtendedProperties": {},
      "Links": {
        "Self": "https://pi-server-1/piwebapi/assetserver/FIR5Ygb8SvdWbkmg4hgEtbBKFgV010LTHyMjMNVFTMDVQ",
        "Databases": "https://pi-server-1/piwebapi/assetserver/FIR5Ygb8SvdWbkmg4hgEtbBKFgV010LTHyMjMNVFTMDVQ/assetdatabases",
        "NotificationContactTemplates": "https://pi-server-1/piwebapi/assetserver/FIR5Ygb8SvdWbkmg4hgEtbBKFgV010LTHyMjMNVFTMDVQ/notificationcontacttemplates",
        "NotificationPlugins": "https://pi-server-1/piwebapi/assetserver/FIR5Ygb8SvdWbkmg4hgEtbBKFgV010LTHyMjMNVFTMDVQ/notificationplugins",
        "SecurityIdentities": "https://pi-server-1/piwebapi/assetserver/FIR5Ygb8SvdWbkmg4hgEtbBKFgV010LTHyMjMNVFTMDVQ/securityidentities",
        "SecurityMappings": "https://pi-server-1/piwebapi/assetserver/FIR5Ygb8SvdWbkmg4hgEtbBKFgV010LTHyMjMNVFTMDVQ/securitymappings",
        "UnitClasses": "https://pi-server-1/piwebapi/assetserver/FIR5Ygb8SvdWbkmg4hgEtbBKFgV010LTHyMjMNVFTMDVQ/unitclasses",
        "AnalysisRulePlugins": "https://pi-server-1/piwebapi/assetserver/FIR5Ygb8SvdWbkmg4hgEtbBKFgV010LTHyMjMNVFTMDVQ/analysisruleplugins",
        "TimeRulePlugins": "https://pi-server-1/piwebapi/assetserver/FIR5Ygb8SvdWbkmg4hgEtbBKFgV010LTHyMjMNVFTMDVQ/timeruleplugins",
        "Security": "https://pi-server-1/piwebapi/assetserver/FIR5Ygb8SvdWbkmg4hgEtbBKFgV010LTHyMjMNVFTMDVQ/security",
        "SecurityEntries": "https://pi-server-1/piwebapi/assetserver/FIR5Ygb8SvdWbkmg4hgEtbBKFgV010LTHyMjMNVFTMDVQ/securityentries"
      }
    },
    {
      "WebId": "FIR5usxPZyRsvBChZt08h2_ekAUElTRVJWRVJfMQ",
      "Id": "6745c0ba-5224-40bc-a166-d3c876fde90",
      "Name": "pi-server-1",
      "Description": "",
      "Path": "\\pi-server-1",
      "IsConnected": false,
      "ServerVersion": "",
      "ServerTime": null,
      "ExtendedProperties": {},
      "Links": {
        "Self": "https://pi-server-1/piwebapi/assetserver/FIR5usxPZyRsvBChZt08h2_ekAUElTRVJWRVJfMQ",
        "Databases": "https://pi-server-1/piwebapi/assetserver/FIR5usxPZyRsvBChZt08h2_ekAUElTRVJWRVJfMQ/assetdatabases",
        "NotificationContactTemplates": "https://pi-server-1/piwebapi/assetserver/FIR5usxPZyRsvBChZt08h2_ekAUElTRVJWRVJfMQ/notificationcontacttemplates",
        "NotificationPlugins": "https://pi-server-1/piwebapi/assetserver/FIR5usxPZyRsvBChZt08h2_ekAUElTRVJWRVJfMQ/notificationplugins",
        "SecurityIdentities": "https://pi-server-1/piwebapi/assetserver/FIR5usxPZyRsvBChZt08h2_ekAUElTRVJWRVJfMQ/securityidentities",
        "SecurityMappings": "https://pi-server-1/piwebapi/assetserver/FIR5usxPZyRsvBChZt08h2_ekAUElTRVJWRVJfMQ/securitymappings",
        "UnitClasses": "https://pi-server-1/piwebapi/assetserver/FIR5usxPZyRsvBChZt08h2_ekAUElTRVJWRVJfMQ/unitclasses",
        "AnalysisRulePlugins": "https://pi-server-1/piwebapi/assetserver/FIR5usxPZyRsvBChZt08h2_ekAUElTRVJWRVJfMQ/analysisruleplugins",
        "TimeRulePlugins": "https://pi-server-1/piwebapi/assetserver/FIR5usxPZyRsvBChZt08h2_ekAUElTRVJWRVJfMQ/timeruleplugins",
        "Security": "https://pi-server-1/piwebapi/assetserver/FIR5usxPZyRsvBChZt08h2_ekAUElTRVJWRVJfMQ/security",
        "SecurityEntries": "https://pi-server-1/piwebapi/assetserver/FIR5usxPZyRsvBChZt08h2_ekAUElTRVJWRVJfMQ/securityentries"
      }
    }
  ]
}

```

selecting the database

Help pages: [PI Web API Help](#) > [AssetServer](#) > [GetDatabases](#)

```

{
  "Links": {},
  "Items": [
    {
      "WebId": "FIR0Ygb8SvdWbkmg4hgEtbBKFgLabh3qyeRhmC40ue6qdgjV010LTHyMjMNVFTMDVQKBNFTx2JR1VQV9VJT04",
      "Id": "de6a62d-9eac-4945-82e1-4a1e9ba988e",
      "Name": "Configuration",
      "Description": "A store for configuration data.",
      "Path": "\\WIN-3228MU1805P\\Configuration",
      "ExtendedProperties": {},
      "Links": {
        "Self": "https://pi-server-1/piwebapi/assetdatabases/FIR0Ygb8SvdWbkmg4hgEtbBKFgLabh3qyeRhmC40ue6qdgjV010LTHyMjMNVFTMDVQKBNFTx2JR1VQV9VJT04",
        "Elements": "https://pi-server-1/piwebapi/assetdatabases/FIR0Ygb8SvdWbkmg4hgEtbBKFgLabh3qyeRhmC40ue6qdgjV010LTHyMjMNVFTMDVQKBNFTx2JR1VQV9VJT04/elements",
        "EventFrames": "https://pi-server-1/piwebapi/assetdatabases/FIR0Ygb8SvdWbkmg4hgEtbBKFgLabh3qyeRhmC40ue6qdgjV010LTHyMjMNVFTMDVQKBNFTx2JR1VQV9VJT04/eventframes",
        "AssetServer": "https://pi-server-1/piwebapi/assetserver/FIR0Ygb8SvdWbkmg4hgEtbBKFgLabh3qyeRhmC40ue6qdgjV010LTHyMjMNVFTMDVQ",
        "ElementCategories": "https://pi-server-1/piwebapi/assetdatabases/FIR0Ygb8SvdWbkmg4hgEtbBKFgLabh3qyeRhmC40ue6qdgjV010LTHyMjMNVFTMDVQKBNFTx2JR1VQV9VJT04/elementcategories",
        "AttributeCategories": "https://pi-server-1/piwebapi/assetdatabases/FIR0Ygb8SvdWbkmg4hgEtbBKFgLabh3qyeRhmC40ue6qdgjV010LTHyMjMNVFTMDVQKBNFTx2JR1VQV9VJT04/attributecategories",
        "TableCategories": "https://pi-server-1/piwebapi/assetdatabases/FIR0Ygb8SvdWbkmg4hgEtbBKFgLabh3qyeRhmC40ue6qdgjV010LTHyMjMNVFTMDVQKBNFTx2JR1VQV9VJT04/tablecategories",
        "AnalysisCategories": "https://pi-server-1/piwebapi/assetdatabases/FIR0Ygb8SvdWbkmg4hgEtbBKFgLabh3qyeRhmC40ue6qdgjV010LTHyMjMNVFTMDVQKBNFTx2JR1VQV9VJT04/analysiscategories",
        "AnalysisTemplates": "https://pi-server-1/piwebapi/assetdatabases/FIR0Ygb8SvdWbkmg4hgEtbBKFgLabh3qyeRhmC40ue6qdgjV010LTHyMjMNVFTMDVQKBNFTx2JR1VQV9VJT04/analysistemplates",
        "EnumerationData": "https://pi-server-1/piwebapi/assetdatabases/FIR0Ygb8SvdWbkmg4hgEtbBKFgLabh3qyeRhmC40ue6qdgjV010LTHyMjMNVFTMDVQKBNFTx2JR1VQV9VJT04/enumerations",
        "Tables": "https://pi-server-1/piwebapi/assetdatabases/FIR0Ygb8SvdWbkmg4hgEtbBKFgLabh3qyeRhmC40ue6qdgjV010LTHyMjMNVFTMDVQKBNFTx2JR1VQV9VJT04/tables",
        "Security": "https://pi-server-1/piwebapi/assetdatabases/FIR0Ygb8SvdWbkmg4hgEtbBKFgLabh3qyeRhmC40ue6qdgjV010LTHyMjMNVFTMDVQKBNFTx2JR1VQV9VJT04/security",
        "SecurityEntries": "https://pi-server-1/piwebapi/assetdatabases/FIR0Ygb8SvdWbkmg4hgEtbBKFgLabh3qyeRhmC40ue6qdgjV010LTHyMjMNVFTMDVQKBNFTx2JR1VQV9VJT04/securityentries"
      }
    },
    {
      "WebId": "FIR0Ygb8SvdWbkmg4hgEtbBKFgLeZv08hY4UmcW_0dWEeNAV010LTHyMjMNVFTMDVQKBNFTx2JR1VQV9VJT04",
      "Id": "a39e4d2d-58c8-49e1-9cd1-6fcd75611e34",
      "Name": "Eoglamp",
      "Description": "",
      "Path": "\\WIN-3228MU1805P\\Eoglamp",
      "ExtendedProperties": {},
      "Links": {
        "Self": "https://pi-server-1/piwebapi/assetdatabases/FIR0Ygb8SvdWbkmg4hgEtbBKFgLeZv08hY4UmcW_0dWEeNAV010LTHyMjMNVFTMDVQKBNFTx2JR1VQV9VJT04",
        "Elements": "https://pi-server-1/piwebapi/assetdatabases/FIR0Ygb8SvdWbkmg4hgEtbBKFgLeZv08hY4UmcW_0dWEeNAV010LTHyMjMNVFTMDVQKBNFTx2JR1VQV9VJT04/elements",
        "EventFrames": "https://pi-server-1/piwebapi/assetdatabases/FIR0Ygb8SvdWbkmg4hgEtbBKFgLeZv08hY4UmcW_0dWEeNAV010LTHyMjMNVFTMDVQKBNFTx2JR1VQV9VJT04/eventframes",
        "AssetServer": "https://pi-server-1/piwebapi/assetserver/FIR0Ygb8SvdWbkmg4hgEtbBKFgLeZv08hY4UmcW_0dWEeNAV010LTHyMjMNVFTMDVQ",
        "ElementCategories": "https://pi-server-1/piwebapi/assetdatabases/FIR0Ygb8SvdWbkmg4hgEtbBKFgLeZv08hY4UmcW_0dWEeNAV010LTHyMjMNVFTMDVQKBNFTx2JR1VQV9VJT04/elementcategories",
        "AttributeCategories": "https://pi-server-1/piwebapi/assetdatabases/FIR0Ygb8SvdWbkmg4hgEtbBKFgLeZv08hY4UmcW_0dWEeNAV010LTHyMjMNVFTMDVQKBNFTx2JR1VQV9VJT04/attributecategories",
        "TableCategories": "https://pi-server-1/piwebapi/assetdatabases/FIR0Ygb8SvdWbkmg4hgEtbBKFgLeZv08hY4UmcW_0dWEeNAV010LTHyMjMNVFTMDVQKBNFTx2JR1VQV9VJT04/tablecategories",
        "AnalysisCategories": "https://pi-server-1/piwebapi/assetdatabases/FIR0Ygb8SvdWbkmg4hgEtbBKFgLeZv08hY4UmcW_0dWEeNAV010LTHyMjMNVFTMDVQKBNFTx2JR1VQV9VJT04/analysiscategories",
        "AnalysisTemplates": "https://pi-server-1/piwebapi/assetdatabases/FIR0Ygb8SvdWbkmg4hgEtbBKFgLeZv08hY4UmcW_0dWEeNAV010LTHyMjMNVFTMDVQKBNFTx2JR1VQV9VJT04/analysistemplates",
        "EnumerationData": "https://pi-server-1/piwebapi/assetdatabases/FIR0Ygb8SvdWbkmg4hgEtbBKFgLeZv08hY4UmcW_0dWEeNAV010LTHyMjMNVFTMDVQKBNFTx2JR1VQV9VJT04/enumerations",
        "Tables": "https://pi-server-1/piwebapi/assetdatabases/FIR0Ygb8SvdWbkmg4hgEtbBKFgLeZv08hY4UmcW_0dWEeNAV010LTHyMjMNVFTMDVQKBNFTx2JR1VQV9VJT04/tables",
        "Security": "https://pi-server-1/piwebapi/assetdatabases/FIR0Ygb8SvdWbkmg4hgEtbBKFgLeZv08hY4UmcW_0dWEeNAV010LTHyMjMNVFTMDVQKBNFTx2JR1VQV9VJT04/security",
        "SecurityEntries": "https://pi-server-1/piwebapi/assetdatabases/FIR0Ygb8SvdWbkmg4hgEtbBKFgLeZv08hY4UmcW_0dWEeNAV010LTHyMjMNVFTMDVQKBNFTx2JR1VQV9VJT04/securityentries"
      }
    }
  ]
}

```

Proceed with the drill down operation up to the desired level/asset.



## 12.4 Error messages and causes

Some error messages and causes:

Message	Cause
North_Readings_to_PI[20884]: WARNING: Error in retrieving the PIWebAPI version, The <b>PI Web API server is not reachable</b> , verify the network reachability	Fledge is not able to reach the machine in which PI Server is running due to a network problem or a firewall restriction.
North_Readings_to_PI[5838]: WARNING: Error in retrieving the PIWebAPI version, <b>503 Service Unavailable</b>	Fledge is able to reach the machine in which PI Server is executing but the PI Web API is not running.
North_Readings_to_PI[24485]: ERROR: Sending JSON data error : <b>Container not found.</b> 4273005507977094880_1measurement_sin_4816_asset_1 - WIN-4M7ODKB0RH2:443 /piwebapi/omf	Fledge is able to interact with PI Web API but there is an attempt to store data in a PI Point that does not exist.

## 12.5 OMF Plugin Data

The OMF north plugin must create type information within the OMF subsystem of the PI Server before any data can be sent. This type information is persisted within the PI Server between sessions and must also be persisted within Fledge for each connection to a PI Server. This is done using the plugin data persistence features of the Fledge north plugin.

This results in an important connection between a north service or task and a PI Server, which does add extra constraints as to what may be done at each end. It is very important this data is kept synchronized between the two ends. In normal circumstances this is not a problem, but there are some actions that can cause problems and require action on both ends.

**Delete a north service or task using the OMF plugin** If a north service or task using the OMF plugin is deleted then the persisted data of the plugin is also lost. This is Fledge's record of what types have been created in the PI Server and is no longer synchronized following the deletion of the north service. Any new service or task that is created and connected to the same PI Server will receive duplicate type errors from the PI Server. There are two possible solutions to this problem;

- Remove the type data from the PI Server such that neither end has the type information.
- Before deleting the north service or task export the plugin persisted data and import that data into the new service or task.

**Cleanup a PI Server and reuse an existing OMF North service or task** This is the opposite problem to that stated above, the plugin will try to send data thinking that the types have already been created in the PI Server and receive an error. Fledge will automatically correct for this and create new types. These new types however will be created with new names, which may not be the desired behavior. Type names are created using a fixed algorithm. To re-use the previous names, stopping the north service and deleting the plugin persisted data will reset the algorithm and recreate the types using the names that had been previously used.

**Taking an existing Fledge north task or service and moving it to a new PI Server** This new PI Server will not have the type information from the old and we will once again get errors when sending data due to these missing types. Fledge will automatically correct for this and create new types. These new types however will be created with new names, which may not be the desired behavior. Type names are created using a fixed algorithm. To re-use the previous names, stopping the north service and deleting the plugin persisted data will reset the algorithm and recreate the types using the names that had been previously used.



## 12.5.1 Managing Plugin Persisted Data

This is not a feature that users would ordinarily need to be concerned with, however it is possible to enable *Developer Features* in the Fledge User Interface that will provide a mechanism to manage this data.

### Enable Develop Features

Navigate to the *Settings* page of the GUI and toggle on the *Developer Features* check box on the bottom left of the page.

### Viewing Persisted Data

In order to view the persisted data for the plugins of a service open either the *North* or *South* page on the user interface and select your service or task. An page will open that allows you to update the configuration of the plugin. This contains a set of tabs that may be selected, when *Developer Features* are enabled one of these tabs will be labeled *Developer*.

OMF North Service

Configuration Advanced Configuration Security Configuration Developer

Endpoint: PI Web API

Send full structure: ☒

Naming Scheme: Concise

Server hostname: 54.160.93.60

Server port, 0=use the default: 0

Producer Token: omf\_north\_0001

Data Source: readings

Static Data:

Sleep Time Retry: 1

Maximum Retry: 3

HTTP Timeout: 10

Integer Format: int64

Number Format: float64

Compression: ☐

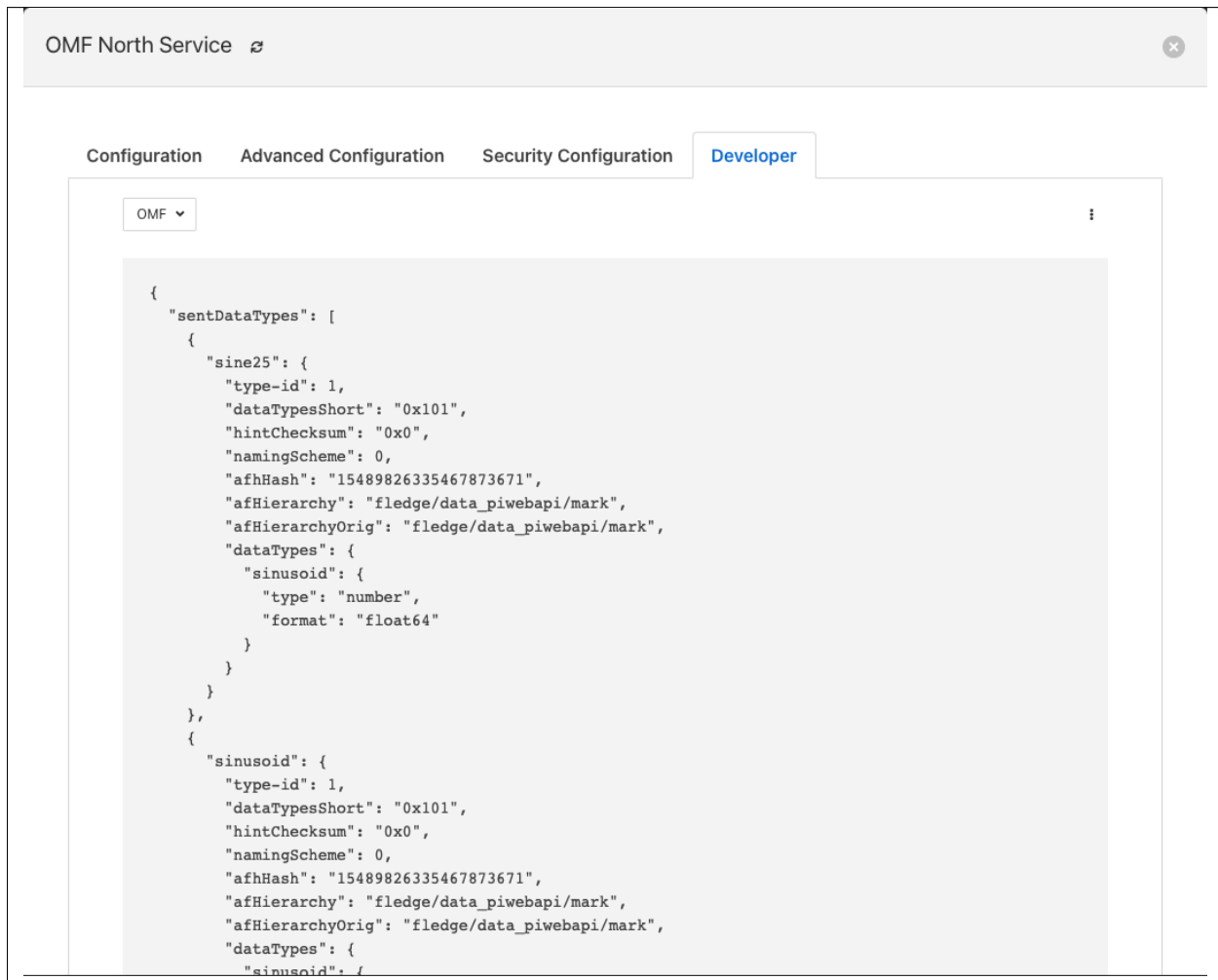
Default Asset Framework Location: /fledge/data\_piwebapi/mark

Asset Framework hierarchy rules: 1 | {}

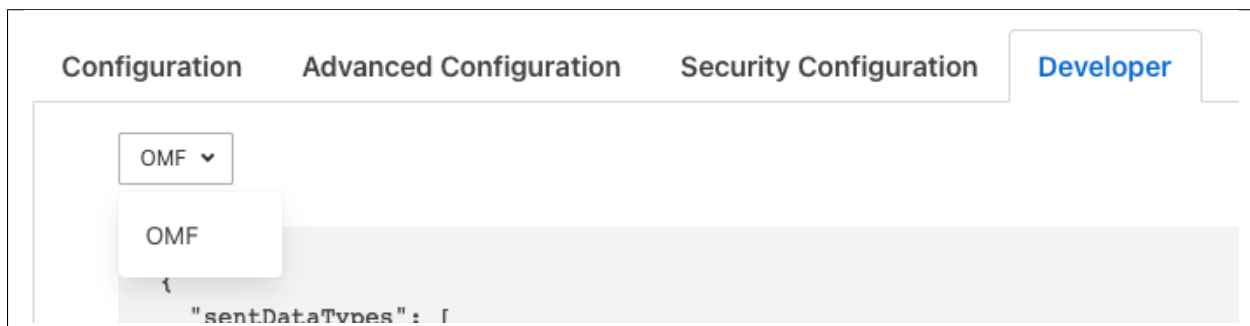
The *Developer* tab will allow the viewing of the persisted data for all of the plugins in that service, filters and either north or south plugins, for which data is persisted.



Persisted data is only written when a plugin is shutdown, therefore in order to get the most up to date view of the data it is recommended that service is disabled before viewing the persisted data. It is possible to view the persisted data of a running service, however this will be a snapshot taken from the last time the service was shutdown.

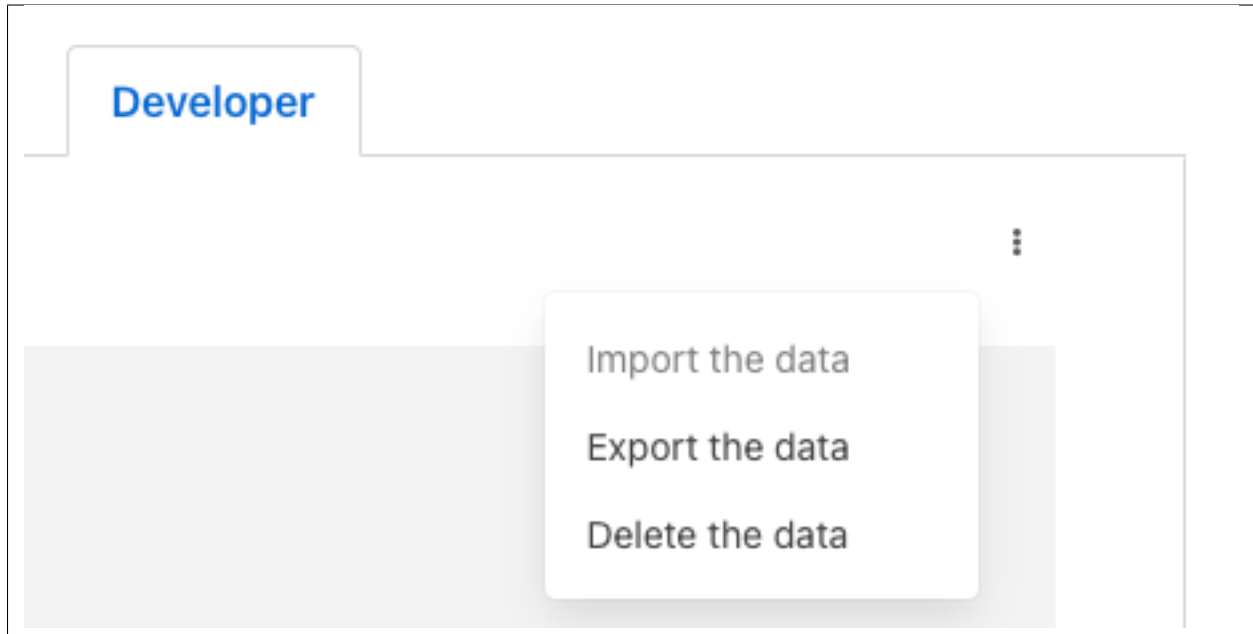


It is possible for more than one plugin within a pipeline to persist data, in order to select between the plugins that have persisted data a menu is provided in the top left which will list all those plugins for which data can be viewed.



As well as viewing the persisted data it is also possible to perform other actions, such as *Delete*, *Export* and *Import*. These actions are available via a menu that appears in the top right of the screen.





**Note:** The service must be disabled before use of the Delete or Import features and to get the latest values when performing an Export.

## 12.5.2 Understanding The OMF Persisted Data

The persisted data takes the form of a JSON document, the following is an example for a Fledge instance configured with just the Sinusoid plugin.

```
{
  "sentDataTypes": [
    {
      "sinusoid": {
        "type-id": 1,
        "dataTypesShort": "0x101",
        "hintChecksum": "0x0",
        "namingScheme": 0,
        "afhHash": "15489826335467873671",
        "afHierarchy": "fledge/data_piwebapi/mark",
        "afHierarchyOrig": "fledge/data_piwebapi/mark",
        "dataTypes": {
          "sinusoid": {
            "type": "number",
            "format": "float64"
          }
        }
      }
    }
  ]
}
```

The *SentDataTypes* is a JSON array of object, with each object representing one data type that has been sent to the PI



Server. The key/value pairs within the object are as follow

Key	Description
type-id	An index of the different types sent for this asset. Each time a new type is sent to the PI Server for this asset this index will be incremented.
dataType-sShort	A summary of the types in the datatypes of the asset. The value is an encoded number that contains the count of each of base types, integer, float and string, in the datapoints of this asset.
hintChecksum	A checksum of the OMFHints used to create this type. 0 if no OMF Hint was used.
nam-ingScheme	The current OMF naming scheme when the type was sent.
afhHash	A Hash of the AF settings for the type.
afHierarchy	The AF Hierarchy location.
afHierarchyOrig	The original setting of AF Hierarchy. This may differ from the above if specific AF rules are in place.
dataTypes	The data type sent to the PI Server. This is an actually OMF type definition and is the exact type definition sent to the PI Web API endpoint.

## 12.6 Possible solutions to common problems

**Recreate a single or a sets of PI Server objects and resend all the data for them to the PI Server on the Asset Framework hierarchy**

**procedure:**

- disable the 1st north instance
- delete the objects in the PI Server, AF + Data archive, that are to be recreated or were partially sent.
- create a new **DISABLED** north instance using a new, unique name and having the same AF hierarchy as the 1st north instance
- install *fledge-filter-asset* on the new north instance
- configure *fledge-filter-asset* with a rule like the following one

```
{
  "rules": [
    {
      "asset_name": "asset_4",
      "action": "include"
    }
  ],
  "defaultAction": "exclude"
}
```

- enable the 2nd north instance
- let the 2nd north instance send the desired amount of data and then disable it
- enable the 1st north instance

**note:**

- the 2nd north instance will be used only to recreate the objects and resend the data
- the 2nd north instance will resend all the data available for the specified *included* assets



- there will be some data duplicated for the recreated assets because part of the information will be managed by both the north instances

**Recreate all the PI Server objects and resend all the data to the PI Server on a different Asset Framework hierarchy level****procedure:**

- disable the 1st north instance
- create a new north instance using a new, unique name and having a new AF hierarchy (North option 'Asset Framework hierarchies tree')

**note:**

- this solution will create a set of new objects unrelated to the previous ones
- all the data stored in Fledge will be sent

**Recreate all the PI Server objects and resend all the data to the PI Server on the same Asset Framework hierarchy level of the 1****procedure:**

- disable the 1st north instance
- delete properly the objects on the PI Server, AF + Data archive, that were eventually partially deleted
- stop / start PI Web API
- create a new north instance 2nd using the same AF hierarchy (North option 'Asset Framework hierarchies tree')

**note:**

- all the types will be recreated on the PI Server. If the structure of each asset, number and types of the properties, does not change the data will be accepted and laced into the PI Server without any error. PI Web API 2019 SP1 1.13.0.6518 will accept the data.
- Using PI Web API 2019 SP1 1.13.0.6518 the PI Server creates objects with the compression feature disabled. This will cause any data that was previously loaded and is still present in the Data Archive, to be duplicated.

**Recreate all the PI Server objects and resend all the data to the PI Server on the same Asset Framework hierarchy level of the 1****procedure:**

- disable the 1st north instance
- delete all the objects on the PI Server side, both in the AF and in the Data Archive, sent by the 1st north instance
- stop / start PI Web API
- create a new north instance using the same AF hierarchy (North option 'Asset Framework hierarchies tree')

**note:**

- all the data stored in Fledge will be sent







## PLUGIN DEVELOPER GUIDE

### 13.1 Source Code Documentation

Documentation of source code will be available

Single PDF document will be available

Fledge makes extensive use of plugin components to extend the base functionality of the platform. In particular, plugins are used to;

- Extend the set of sensors and actuators that Fledge supports.
- Extend the set of services to which Fledge will push accumulated data gathered from those sensors.
- The mechanism by which Fledge buffers data internally.
- Filter plugins may be used to augment, edit or remove data as it flows through Fledge.
- Rule plugins extend the rules that may trigger the delivery of notifications at the edge.
- Notification delivery plugins allow for new delivery mechanisms to be integrated into Fledge.

This chapter presents the plugins that are bundled with Fledge, how to write and use new plugins to support different sensors, protocols, historians and storage devices. It will guide you through the process and entry points that are required for the various different types of plugin.

There are also numerous plugins that are available as separate packages or in separate repositories that may be used with Fledge.

### 13.2 Plugins

In this version of Fledge you have six types of plugins:

- **South Plugins** - They are responsible for communication between Fledge and the sensors and actuators they support. Each instance of a Fledge South microservice will use a plugin for the actual communication to the sensors or actuators that that instance of the South microservice supports.
- **North Plugins** - They are responsible for taking reading data passed to them from the South bound service and doing any necessary conversion to the data and providing the protocol to send that converted data to a north-side task.
- **Storage Plugins** - They sit between the Storage microservice and the physical data storage mechanism that stores the Fledge configuration and readings data. Storage plugins differ from other plugins in that they are written exclusively in C/C++, however they share the same common attributes and entry points that the other filter must support.



- **Filter Plugins** - Filter plugins are used to modify data as it flows through Fledge. Filter plugins may be combined into a set of ordered filters that are applied as a pipeline to either the south ingress service or the north egress task that sends data to external systems.
- **Notification Rule Plugins** - These are used by the optional notification service in order to evaluate data that flows into the notification service to determine if a notification should be sent.
- **Notification Delivery Plugins** - These plugins are used by the optional notification service to deliver a notification to a system when a notification rule has triggered. These plugins allow the mechanisms to deliver notifications to be extended.

### 13.2.1 Plugins in this version of Fledge

This version of Fledge provides the following plugins in the main repository:

Type	Name	Initial Status	Description	Availability	Notes
Storage	SQLite	Enabled	SQLite storage for data and metadata	Ubuntu: x86_64 Ubuntu Core: x86, ARM Raspbian	
Storage	Postgres	Disabled	PostgreSQL storage for data and metadata	Ubuntu: x86_64 Ubuntu Core: x86, ARM Raspbian	
North	OMF	Disabled	OSIssoft Message Format sender to PI Connector Relay OMF	Ubuntu: x86_64 Ubuntu Core: x86, ARM Raspbian	It works with PI Connector Relay OMF 1.2.X and 2.2. The plugin also works against EDS and OCS.

In addition to the plugins in the main repository, there are many other plugins available in separate repositories, a list of the is maintained within this document.

### 13.2.2 Installing New Plugins

As a general rule and unless the documentation states otherwise, plugins should be installed in two ways:

- When the plugin is available as **package**, it should be installed when **Fledge is running**. This is the required method because the package executed pre and post-installation tasks that require Fledge to run.
- When the plugin is available as **source code**, it should be installed when **Fledge is either running or not**. You will want to manually move the plugin code into the right location where Fledge is installed, add pre-requisites and execute the REST commands necessary to start the plugin **after** you have started Fledge if it is not running when you start this process.

For example, this is the command to use to install the *OpenWeather* South plugin:

```
$ sudo systemctl status fledge.service
fledge.service - LSB: Fledge
   Loaded: loaded (/etc/init.d/fledge; bad; vendor preset: enabled)
   Active: active (running) since Wed 2018-05-16 01:32:25 BST; 4min 1s ago
     Docs: man:systemd-sysv-generator(8)
    CGroup: /system.slice/fledge.service
            └─13741 python3 -m fledge.services.core
              └─13746 /usr/local/fledge/services/storage --address=0.0.0.0 --port=40138
```

(continues on next page)



(continued from previous page)

```

May 16 01:36:09 ubuntu python3[13741]: Fledge[13741] INFO: scheduler: fledge.services.
↳core.scheduler.scheduler: Process started: Schedule 'stats collection' process
↳'stats coll
                                ['tasks/statistics', '--port=40138', '--
↳address=127.0.0.1', '--name=stats collector']
...
Fledge v1.3.1 running.
Fledge Uptime: 266 seconds.
Fledge records: 0 read, 0 sent, 0 purged.
Fledge does not require authentication.
=== Fledge services:
fledge.services.core
=== Fledge tasks:
$
$ sudo cp fledge-south-openweathermap-1.2-x86_64.deb /var/cache/apt/archives/.
$ sudo apt install /var/cache/apt/archives/fledge-south-openweathermap-1.2-x86_64.deb
Reading package lists... Done
Building dependency tree
Reading state information... Done
Note, selecting 'fledge-south-openweathermap' instead of '/var/cache/apt/archives/
↳fledge-south-openweathermap-1.2-x86_64.deb'
The following packages were automatically installed and are no longer required:
  linux-headers-4.4.0-109 linux-headers-4.4.0-109-generic linux-headers-4.4.0-119_
↳linux-headers-4.4.0-119-generic linux-headers-4.4.0-121 linux-headers-4.4.0-121-
↳generic
  linux-image-4.4.0-109-generic linux-image-4.4.0-119-generic linux-image-4.4.0-121-
↳generic linux-image-extra-4.4.0-109-generic linux-image-extra-4.4.0-119-generic
  linux-image-extra-4.4.0-121-generic
Use 'sudo apt autoremove' to remove them.
The following NEW packages will be installed
  fledge-south-openweathermap
0 to upgrade, 1 to newly install, 0 to remove and 0 not to upgrade.
Need to get 0 B/3,404 B of archives.
After this operation, 0 B of additional disk space will be used.
Selecting previously unselected package fledge-south-openweathermap.
(Reading database ... 211747 files and directories currently installed.)
Preparing to unpack ../fledge-south-openweathermap-1.2-x86_64.deb ...
Unpacking fledge-south-openweathermap (1.2) ...
Setting up fledge-south-openweathermap (1.2) ...
openweathermap plugin installed.
$
$ fledge status
Fledge v1.3.1 running.
Fledge Uptime: 271 seconds.
Fledge records: 36 read, 0 sent, 0 purged.
Fledge does not require authentication.
=== Fledge services:
fledge.services.core
fledge.services.south --port=42066 --address=127.0.0.1 --name=openweathermap
=== Fledge tasks:
$

```

You may also install new plugins directly from within the Fledge GUI, however you will need to have setup your Linux machine to include the Fledge package repository in the list of repositories the Linux package manager searches for new packages.



## 13.3 Representing Data

The key purpose of Fledge and the plugins is the manipulation of data, that data is passed around the system and represented in a number of ways. This section will introduce the data representation formats used at various locations within the Fledge system. Conceptually the unit of data that we use is a reading. The reading represents the state of a monitored device at a point in time and has a number of elements.

Name	Description
asset	The name of the asset or device to which the data refers
timestamp	The point in time at which these values were observed.
data points	A set of named values for the data held for the asset

There are actually two timestamps within a reading and these may be different. There is a *user\_ts*, which is the time the plugin assigned to the reading data and may come from the device itself and the *ts*. The *ts* timestamp is set by the system when the data is read into Fledge. Unless the plugin is able to determine a timestamp from the device the *user\_ts* is usually the same as the *ts*.

The data points themselves are a set of name and value pairs, with the values supporting a number of different data types. These will be described below.

Reading data is nominally stored and passed between the APIs using JSON, however for convenience it is access in different ways within the different languages that can be used to implement Fledge components and plugins. In JSON a reading is represented as a JSON DICT whereas in C++ a Reading is a class, as is a data point. The way the different data point types are represented is outline below.

Type	JSON	C++	Python
Integer	An integer	An int	An integer
Floating Point	A floating point value	A double	A floating point
Boolean	A string either "true" or "false"	A bool	A boolean
String	A string	A std::string pointer	A string
List of numbers	An array of floating point values	A std::vector<double>	A list of floating point values
2 Dimensional list of numbers	A list of lists of floating point values	A std::vector of std::vector<double> pointers	A list of lists of floating point values
Data buffer	A base64 encoded string with a header	A Databuffer class	A 1 dimensional numpy array of values
Image	A base64 encoded string with a header	A DPLImage class	A 2 dimensional numpy array of pixels. In the case of RGB images each pixels is an array

## 13.4 Writing and Using Plugins

A plugin has a small set of external entry points that must exist in order for Fledge to load and execute that plugin. Currently plugins may be written in either Python or C/C++, the set of entry points is the same for both languages. The entry points detailed here will be presented for both languages, a more in depth discussion of writing plugins in C/C++ will then follow.



### 13.4.1 Common Fledge Plugin API

Every plugin provides at least one common API entry point, the *plugin\_info* entry point. It is used to obtain information about a plugin before it is initialized and used. It allows Fledge to determine what type of plugin it is, e.g. a South bound plugin or a North bound plugin, obtain default configuration information for the plugin and determine version information.

#### Plugin Information

The information entry point is implemented as a call, *plugin\_info*, that takes no arguments. Data is returned from this API call as a JSON document with certain well known properties.

A typical Python implementation of this would simply return a fixed dictionary object that encodes the required properties.

```
def plugin_info():
    """ Returns information about the plugin.

    Args:
    Returns:
        dict: plugin information
    Raises:
    """

    return {
        'name': 'DHT11 GPIO',
        'version': '1.0',
        'mode': 'poll',
        'type': 'south',
        'interface': '1.0',
        'config': _DEFAULT_CONFIG
    }
```

These are the properties returned by the JSON document:

- **name** - A textual name that will be used for reporting purposes for this plugin.
- **version** - This property allows the version of the plugin to be communicated to the plugin loader. This is used for reporting purposes only and has no effect on the way Fledge interacts with the plugin.
- **mode** - A set of options that defines how the plugin operates. Multiple values can be given, the different options are separated from each other using the | symbol.
- **type** - The type of the plugin, used by the plugin loader to determine if the plugin is being used correctly. The type is a simple string and may be *south*, *north*, *filter*, *rule* or *delivery*.

---

**Note:** If you browse the Fledge code you may find old plugins with type *device*: this was the type used to indicate a South plugin and it is now deprecated.

---

- **interface** - This property reports the version of the plugin API to which this plugin was written. It allows Fledge to support upgrades of the API whilst being able to recognise the version that a particular plugin is compliant with. Currently all interfaces are version 1.0.
- **configuration** - This allows the plugin to return a JSON document which contains the default configuration of the plugin. This is in line with the extensible plugin mechanism of Fledge, each plugin will return a set of configuration items that it wishes to use, this will then be used to extend the set of Fledge configuration items. This structure, a JSON document, includes default values but no actual values for each configuration option.



The first time Fledge's configuration manager sees a category it will register the category and create values for each item using the default value in the configuration document. On subsequent calls the value already in the configuration manager will be used. This mechanism allows the plugin to extend the set of configuration variables whilst giving the user the opportunity to modify the value of these configuration items. It also allow new versions of plugins to add new configuration items whilst retaining the values of previous items. And new items will automatically be assigned the default value for that item. As an example, a plugin that wishes to maintain two configuration variables, say a GPIO pin to use and a polling interval, would return a configuration document that looks as follows:

```
{
  'pollInterval': {
    'description': 'The interval between poll calls to the device poll routine,
    ↪expressed in milliseconds.',
    'type': 'integer',
    'default': '1000'
  },
  'gpiopin': {
    'description': 'The GPIO pin into which the DHT11 data pin is connected',
    'type': 'integer',
    'default': '4'
  }
}
```

The various values that may appear in the *mode* item are shown in the table below

Mode	Description
poll	The plugin is a polled plugin and <i>plugin_poll</i> will be called periodically to obtain new values.
async	The plugin is an asynchronous plugin, <i>plugin_poll</i> will not be called and the plugin will be supplied with a callback function that it calls each time it has a new value to pass to the system. The <i>plugin_register_ingest</i> entry point will be called to register the callback with the plugin. The <i>plugin_start</i> call will be called once to initiate the asynchronous delivery of data.
none	This is equivalent to poll.
control	The plugin support a control flow to the device the plugin is connected to. The must supply the control entry points <i>plugin_write</i> and <i>plugin_operation</i> .

A C/C++ plugin returns the same information as a structure, this structure includes the JSON configuration document as a simple C string.

```
#include <plugin_api.h>

extern "C" {

/**
 * The plugin information structure
 */
static PLUGIN_INFORMATION info = {
    "MyPlugin",           // Name
    "1.0.1",              // Version
    0,                    // Flags
    PLUGIN_TYPE_SOUTH,    // Type
    "1.0.0",              // Interface version
    default_config         // Default configuration
};
```

(continues on next page)



(continued from previous page)

```

/**
 * Return the information about this plugin
 */
PLUGIN_INFORMATION *plugin_info()
{
    return &info;
}

```

In the above example the constant `default_config` is a string that contains the JSON configuration document. In order to make the JSON easier to manage a special macro is defined in the `plugin_api.h` header file. This macro is called `QUOTE` and is designed to ease the quoting requirements to create this JSON document.

```

const char *default_config = QUOTE({
    "plugin" : {
        "description" : "My example plugin in C++",
        "type" : "string",
        "default" : "MyPlugin",
        "readonly" : "true"
    },
    "asset" : {
        "description" : "The name of the asset the plugin will produce",
        "type" : "string",
        "default" : "MyAsset"
    }
});

```

The `flags` items contains a bitmask of flag values used to pass information regarding the behavior and requirements of the plugin. The flag values currently supported are shown below

Flag Name	Description
SP_COMMON	Used exclusively by storage plugins. The plugin supports the common table access needed to store configuration
SP_READINGS	Used exclusively by storage plugins. The plugin supports the storage of reading data
SP_ASYNC	The plugin is an asynchronous plugin, <code>plugin_poll</code> will not be called and the plugin will be supplied with a callback function that it calls each time it has a new value to pass to the system. The <code>plugin_register_ingest</code> entry point will be called to register the callback with the plugin. The <code>plugin_start</code> call will be called once to initiate the asynchronous delivery of data. This applies only to south plugins.
SP_PERSISTENT_DATA	The plugin wishes to persist data between executions
SP_INGEST	Non-south plugin wishes to ingest new data into the system. Used by notification plugins
SP_GET_MANAGEMENT	The plugin requires access to the management API interface for the service
SP_GET_STORAGE	The plugin requires access to the storage service
SP_DEPRECATED	The plugin should be considered to be deprecated. New service can not use this plugin, but existing services may continue to use it
SP_BUILT_IN	The plugin is not implemented as an external package but is built into the system
SP_CONTROL	The plugin implement control features

These flag values may be combined by use of the or operator where more than one of the above options is supported.



## Plugin Initialization

The plugin initialization is called after the service that has loaded the plugin has collected the plugin information and resolved the configuration of the plugin but before any other calls will be made to the plugin. The initialization routine is called with the resolved configuration of the plugin, this includes values as opposed to the defaults that were returned in the *plugin\_info* call.

This call is used by the plugin to do any initialization or state creation it needs to do. The call returns a handle which will be passed into each subsequent call of the plugin. The handle allows the plugin to have state information that is maintained and passed to it whilst allowing for multiple instances of the same plugin to be loaded by a service if desired. It is equivalent to a this or self pointer for the plugin, although the plugin is not defined as a class.

In Python a simple example of a sensor that reads a GPIO pin for data, we might choose to use that configured GPIO pin as the handle we pass to other calls.

```
def plugin_init(config):
    """ Initialise the plugin.

    Args:
        config: JSON configuration document for the device configuration category
    Returns:
        handle: JSON object to be used in future calls to the plugin
    Raises:
        """

    handle = config['gpiopin']['value']
    return handle
```

A C/C++ plugin should return a value in a *void* pointer that can then be dereferenced in subsequent calls. A typical C++ implementation might create an instance of a class and use that instance as the handle for the plugin.

```
/**
 * Initialise the plugin, called to get the plugin handle
 */
PLUGIN_HANDLE plugin_init(ConfigCategory *config)
{
    MyPluginClass *plugin = new MyPluginClass();

    plugin->configure(config);

    return (PLUGIN_HANDLE)plugin;
}
```

It should also be observed in the above C/C++ example the *plugin\_init* call is passed a pointer to a *ConfigCategory* class that encapsulates the JSON configuration category for the plugin. Details of the *ConfigCategory* class are available in the section .



## Plugin Shutdown

The plugin shutdown method is called as part of the shutdown sequence of the service that loaded the plugin. It gives the plugin the opportunity to do any cleanup operations before terminating. As with all calls it is passed the handle of our plugin instance. Plugins can not prevent the shutdown and do not have to implement any actions. In our simple sensor example there is nothing to do in order to shutdown the plugin.

A C/C++ plugin might use this *plugin\_shutdown* call to delete the plugin class instance it created in the corresponding *plugin\_init* call.

```
/**
 * Shutdown the plugin
 */
void plugin_shutdown(PLUGIN_HANDLE *handle)
{
    MyPluginClass *plugin = (MyPluginClass *)handle;

    delete plugin;
}
```

## Plugin Reconfigure

The plugin reconfigure method is called whenever the configuration of the plugin is changed. It allows for the dynamic reconfiguration of the plugin whilst it is running. The method is called with the handle of the plugin and the updated configuration document. The plugin should take whatever action it needs to and return a new or updated copy of the handle that will be passed to future calls.

The plugin reconfigure method is shared between most but not all plugin types. In particular it does not exist for the shorted lived plugins that are created to perform a single operation and then terminated. These are the north plugins and the notification delivery plugins.

Using a simple Python example of our sensor reading a GPIO pin, we extract the new pin number from the new configuration data and return that as the new handle for the plugin instance.

```
def plugin_reconfigure(handle, new_config):
    """ Reconfigures the plugin, it should be called when the configuration of the
    ↪ plugin is changed during the
        operation of the device service.
        The new configuration category should be passed.

    Args:
        handle: handle returned by the plugin initialisation call
        new_config: JSON object representing the new configuration category for the
    ↪ category
    Returns:
        new_handle: new handle to be used in the future calls
    Raises:
        """

    new_handle = new_config['gpiopin']['value']
    return new_handle
```

In C/C++ the *plugin\_reconfigure* method is very similar, note however that the *plugin\_reconfigure* call is passed the JSON configuration category as a string and not a *ConfigCategory*, it is easy to parse and create the C++ class however, a name for the category must be given however.



```
/**
 * Reconfigure the plugin
 */
void plugin_reconfigure(PLUGIN_HANDLE *handle, string& newConfig)
{
    ConfigCategory      config("newConfiguration", newConfig);
    MyPluginClass       *plugin = (MyPluginClass *)*handle;

    plugin->configure(&config);
}
```

It should be noted that the *plugin\_reconfigure* call may be delivered in a separate thread for a C/C++ plugin and that the plugin should implement any mutual exclusion mechanisms that are required based on the actions of the *plugin\_reconfigure* method.

### 13.4.2 Configuration Lifecycle

Fledge has a very particular way of handling configuration, there are a number of design aims that have resulted in the configuration system within Fledge.

- A desire to allow the plugins to define their own configuration elements.
- Dynamic configuration that allows for maximum uptime during configuration changes.
- A descriptive way to define the configuration such that user interfaces can be built without prior knowledge of the elements to be configured.
- A common approach that will work across many different languages.

Fledge divides its configuration in categories. A category being a collection of configuration items. A category is also the smallest item of configuration that can be subscribed to by the code. This subscription mechanism is the way that Fledge facilitates dynamic reconfiguration. It allows a service to subscribe to one or more configuration categories, whenever an item within a category changes the central configuration manager will call a handler to pass the newly updated configuration category. This handler may be within a services or between services using the micro service management API that every service must support. The mechanism however is transparent to the code involved.

The configuration items within a category are JSON object, the object key is the name of the configuration item, the object itself contains data about that item. As an example, if we wanted to have a configuration item called *MaxRetries* that is an integer with a default value of 5, then we would configured it using the JSON object

```
"MaxRetries" : {
    "type" : "integer",
    "default" : "5"
}
```

We have used the properties *type* and *default* to define properties of the configuration item *MaxRetries*. These are not the only properties that a configuration item can have, the full set of properties are



Property	Description
default	The default value for the configuration item. This is always expressed as a string regardless of the type of the configuration item.
deprecated	A boolean flag to indicate that this item is no longer used and will be removed in a future release.
description	A description of the configuration item used in the user interface to give more details of the item. Commonly used as a mouse over help prompt.
display-name	The string to use in the user interface when presenting the configuration item. Generally a more user friendly form of the item name. Item names are referenced within the code.
length	The maximum length of the string value of the item.
mandatory	A boolean flag to indicate that this item can not be left blank.
maximum	The maximum value for a numeric configuration item.
minimum	The minimum value for a numeric configuration item.
options	Only used for enumeration type elements. This is a JSON array of string that contains the options in the enumeration.
order	Used in the user interface to give an indication of how high up in the dialogue to place this item.
read-only	A boolean property that can be used to include items that can not be altered by the API.
rule	A validation rule that will be run against the value. This must evaluate to true for the new value to be accepted by the API
type	The type of the configuration item. The list of types supported are; integer, float, string, password, enumeration, boolean, JSON, URL, IPV4, IPV6, script, code, X509 certificate and northTask.
validity	An expression used to determine if the configuration item is valid. Used in the UI to gray out one value based on the value of others.
value	The current value of the configuration item. This is not included when defining a set of default configuration in, for example, a plugin.

Of the above properties of a configuration item *type*, *default* and *description* are mandatory, all other may be omitted.

Configuration data is stored by the storage service and is maintained by the configuration in the core Fledge service. When code requires configuration it would create a configuration category with a set of items as a JSON document. It would then register that configuration category with the configuration manager. The configuration manager is responsible for storing the data in the storage layer, as it does this it first checks to see if there is already a configuration category from a previous execution of the code. If one does exist then the two are merged, this merging process allows updates to the software to extend the configuration category whilst maintaining any changes in values made by the user.

Dynamic reconfiguration within Fledge code is supported by allowing code to subscribe for changes in a configuration category. The services that load plugin will automatically register for the plugin configuration category and when changes are seen will call the *plugin\_reconfigure* entry point of the plugin with the new configuration. This allows the plugins to receive the updated configuration and take what actions it must in order to honour the changes to configuration. This allows for configuration to be changed without the need to stop and restart the services, however some plugins may need to close connections and reopen them, which may cause a slight interruption in the process of gathering data. That choice is up to the developers of the individual plugins.



## Discovery

It is possible using this system to do a limited amount of discovery and tailoring of plugin configuration. A typical case when discovery might be used is to discover devices on a network that can be monitored. This can be achieved by putting the discovery code in the *plugin\_info* entry point and having that discovery code alter the default configuration that is returned as part of the plugin information structure.

Any example of this might be to have an enumeration in the configuration that enumerates the devices to be monitored. The discovery code would then populate the enumerations options item with the various devices it discovered when the *plugin\_info* call was made.

An example of the *plugin\_info* entry point that does this might be as follows

```
/**
 * Return the information about this plugin
 */
PLUGIN_INFORMATION *plugin_info()
{
    DeviceDiscovery discover;

    char *config = discover.discover(default_config, "discovered");
    info.config = config;
    return &info;
}
```

The configuration in *default\_config* is assumed to have an enumeration item called *discovered*

```
"discovered" : {
    "description" : "The discovered devices, select 'Manual' to manually enter an
↳IP address",
    "type" : "enumeration",
    "options" : [ "Manual" ],
    "default" : "Manual",
    "displayName": "Devices",
    "mandatory": "true",
    "order" : "2"
},
"IP" : {
    "description" : "The IP address of your device, used to add a device that
↳could not be discovered",
    "type" : "string",
    "default" : "127.0.0.1",
    "displayName": "IP Address",
    "mandatory": "true",
    "order" : "3",
    "validity" : "discovered == \"Manual\""
},
```

Note the use of the *Manual* option to allow entry of devices that could not be discovered.

The *discover* method does the actual discovery and manipulates the JSON configuration to add the *options* element of the configuration item.

The code that connects to the device should then look at the *discovered* configuration item, if it finds it set to *Manual* then it will get an IP address from the *IP* configuration item. Otherwise it uses the information in the *discovered* item to connect, note that this need not just be an IP address, you can format the data in a way that is more user friendly and have the connection code extract what it needs or create a table in the *discover* method to allow for user meaningful strings to be mapped to network addresses.



The example here was written in C++, there is nothing that is specific to C++ however and the same approach can be taken in Python.

One thing to note however, the *plugin\_info* call is used in the display of available plugins, discovery code that is very slow will impact the performance of plugin selection.

## 13.5 South Plugins

South plugins are used to communicate with sensors and actuators, there are two modes of plugin operation; *asyncio* and *polled*.

### 13.5.1 Polled Mode

Polled mode is the simplest form of South plugin that can be written, a poll routine is called at an interval defined in the plugin configuration. The South service determines the type of the plugin by examining at the mode property in the information the plugin returns from the *plugin\_info* call.

#### Plugin Poll

The plugin *poll* method is called periodically to collect the readings from a poll mode sensor. As with all other calls the argument passed to the method is the handle returned by the initialization call, the return of the method should be the JSON payload of the readings to return.

The JSON payload returned, as a Python dictionary, should contain the properties; asset, timestamp, key and readings.

Property	Description
asset	The asset key of the sensor device that is being read
timestamp	A timestamp for the reading data
key	A UUID which is the unique key of this reading
readings	The reading data itself as a JSON object

It is important that the *poll* method does not block as this will prevent the proper operation of the South microservice. Using the example of our simple DHT11 device attached to a GPIO pin, the *poll* routine could be:

```
def plugin_poll(handle):
    """ Extracts data from the sensor and returns it in a JSON document as a Python_
    dict.

    Available for poll mode only.

    Args:
        handle: handle returned by the plugin initialisation call
    Returns:
        returns a sensor reading in a JSON document, as a Python dict, if it is_
    available
        None - If no reading is available
    Raises:
        DataRetrievalError
    """

    try:
        humidity, temperature = Adafruit_DHT.read_retry(Adafruit_DHT.DHT11, handle)
```

(continues on next page)



(continued from previous page)

```

    if humidity is not None and temperature is not None:
        time_stamp = str(datetime.now(tz=timezone.utc))
        readings = { 'temperature': temperature , 'humidity' : humidity }
        wrapper = {
            'asset': 'dht11',
            'timestamp': time_stamp,
            'key': str(uuid.uuid4()),
            'readings': readings
        }
        return wrapper
    else:
        return None

except Exception as ex:
    raise exceptions.DataRetrievalError(ex)

return None

```

### 13.5.2 Async IO Mode

In asyncio mode the plugin inserts itself into the event processing loop of the South Service itself. This is a more complex mechanism and is intended for plugins that need to block or listen for incoming data via a network.

#### Plugin Start

The *plugin\_start* method, as with other plugin calls, is called with the plugin handle data that was returned from the *plugin\_init* call. The *plugin\_start* call will only be called once for a plugin, it is the responsibility of *plugin\_start* to install the plugin code into the python event handling system for asyncio. Assuming an example whereby the interface to a sensor is via HTTP and the sensor will make HTTP POST calls to our plugin in order to send data into Fledge, a *plugin\_start* for this scenario would create a web application endpoint for reception of the POST command.

```

loop = asyncio.get_event_loop()
app = web.Application(middlewares=[middleware.error_middleware])
app.router.add_route('POST', '/', SensorPhoneIngest.render_post)
handler = app.make_handler()
coro = loop.create_server(handler, host, port)
server = asyncio.ensure_future(coro)

```

This code first gets the event loop for this Python execution, it then creates the web application and adds a route for the POST request. In this case it is calling the *render\_post* method of the object *SensorPhone*. It then goes on to create the handler and install the web server instance into the event system.

#### Async Data Callback

The async data callback is used for incoming sensor data and passing that reading data into the Fledge ingest process. Unlike the poll mechanism, this is done from within the callback rather than by passing the data back to the South service itself. A plugin entry point, *plugin\_register\_ingest* is called by the south service before the plugin is started to register the callback with the plugin. The plugin would usually save the callback function and the reference data for later use.



```
def plugin_register_ingest(handle, callback, ingest_ref):
    """Required plugin interface component to communicate to South C server

    Args:
        handle: handle returned by the plugin initialisation call
        callback: C opaque object required to passed back to C->ingest method
        ingest_ref: C opaque object required to passed back to C->ingest method
    """
    global c_callback, c_ingest_ref
    c_callback = callback
    c_ingest_ref = ingest_ref
```

The plugin then uses these saved references when it has data to be ingested. A new reading is constructed and passed to the callback function using *async\_ingest* object that should be imported by the plugin.

```
import async_ingest
```

Then for each reading to be ingested the data is sent to the ingest thread of the south plugin using the following construct.

```
data = {
    'asset': self.asset_name,
    'timestamp': utils.local_timestamp(),
    'readings': reads
}
async_ingest.ingest_callback(c_callback, c_ingest_ref, data)
```

```
message['status'] = code
return web.json_response(message)
```

### 13.5.3 Set Point Control

South plugins can also be used to exert control on the underlying device to which they are connected. This is not intended for use as a substitute for real time control systems, but rather as a mechanism to make non-time critical changes to a device or to trigger an operation on the device.

To make a south plugin support control features there are two steps that need to be taken

- Tag the plugin as supporting control
- Add the entry points for control

#### Enable Control

A plugin enables control features by means of the mode field in the plugin information dict which is returned by the *plugin\_info* entry point of the plugin. The flag value *control* should be added to the mode field of the plugin. Multiple flag values are separated by the pipe symbol '|'.

```
# plugin information dict
{
    'name': 'Sinusoid Poll plugin',
    'version': '1.9.2',
    'mode': 'poll|control',
    'type': 'south',
    'interface': '1.0',
```

(continues on next page)



(continued from previous page)

```
'config': _DEFAULT_CONFIG
}
```

Adding this flag will cause the south service to do a number of things when it loads the plugin;

- The south service will attempt to resolve the two control entry points.
- A toggle will be added to the advanced configuration category of the service that will permit the disabling of control services.
- A security category will be added to the south service that contains the access control lists and permissions associated with the service.

## Control Entry Points

Two entry points are supported for control operations in the south plugin

- **plugin\_write**: which is used to set the value of a parameter within the plugin or device
- **plugin\_operation**: which is used to perform an operation on the plugin or device

The south plugin can support one or both of these entry points as appropriate for the plugin.

## Write Entry Point

The write entry point is used to set data in the plugin or write data into the device.

The plugin write entry point is defined as follows

```
def plugin_write(handle, name, value)
```

Where the parameters are;

- **handle** the handle of the plugin instance
- **name** the name of the item to be changed
- **value** a string presentation of the new value to assign to the item

The return value defines if the write was successful or not. True is returned for a successful write.

```
def plugin_write(handle, name, value):
    """ Setpoint write operation

    Args:
        handle: handle returned by the plugin initialisation call
        name: Name of parameter to write
        value: Value to be written to that parameter
    Returns:
        bool: Result of the write operation
    """
    _LOGGER.info("plugin_write(): name={}, value={}".format(name, value))
    return True
```

In this case we are merely printing the parameter name and the value to be set for this parameter. Normally control would be used for making a change with the connected device itself, such as changing a PLC register value. This is simply an example to demonstrate the API.



## Operation Entry Point

The plugin will support an operation entry point. This will execute the given operation synchronously, it is expected that this operation entry point will be called using a separate thread, therefore the plugin should implement operations in a thread safe environment.

The plugin write operation entry point is defined as follows

```
def plugin_operation(handle, operation, params)
```

Where the parameters are;

- **handle** the handle of the plugin instance
- **operation** the name of the operation to be executed
- **params** a list of name/value tuples that are passed to the operation

The *operation* parameter should be used by the plugin to determine which operation is to be performed. The actual parameters are passed in a list of key/value tuples as strings.

The return from the call is a boolean result of the operation, a failure of the operation or a call to an unrecognized operation should be indicated by returning a false value. If the operation succeeds a value of true should be returned.

The following example shows the implementation of the plugin operation entry point.

```
def plugin_operation(handle, operation, params):
    """ Setpoint control operation

    Args:
        handle: handle returned by the plugin initialisation call
        operation: Name of operation
        params: Parameter list
    Returns:
        bool: Result of the operation
    """
    _LOGGER.info("plugin_operation(): operation={}, params={}".format(operation,
↪params))
    return True
```

In the case of a real machine the operation would most likely cause an action on a machine, for example a request to the machine to re-calibrate itself. Above example is just a demonstration of the API.

### 13.5.4 A South Plugin Example In Python: the DHT11 Sensor

Let's try to put all the information together and write a plugin. We can continue to use the example of an inexpensive sensor, the DHT11, used to measure temperature and humidity, directly wired to a Raspberry PI. This plugin is available on github, .

First, here is a set of links where you can find more information regarding this sensor:

- 
- 
-



### The Hardware

The DHT sensor is directly connected to a Raspberry PI 2 or 3. You may decide to buy a sensor and a resistor and solder them yourself, or you can buy a ready-made circuit that provides the correct output to wire to the Raspberry PI. shows a DHT11 with resistor that you can buy online.

The sensor can be directly connected to the Raspberry PI GPIO (General Purpose Input/Output). An introduction to the GPIO and the pinset is available . In our case, you must connect the sensor on these pins:

- **VCC** is connected to PIN #2 (5v Power)
- **GND** is connected to PIN #6 (Ground)
- **DATA** is connected to PIN #7 (BCM 4 - GPCLK0)

shows the sensor wired to the Raspberry PI and is a zoom into the wires used.

### The Software

For this plugin we use the ADAFruit Python Library (links to the GitHub repository are above). First, you must install the library (in future versions the library will be provided in a ready-made package):

```
$ git clone https://github.com/adafruit/Adafruit_Python_DHT.git
Cloning into 'Adafruit_Python_DHT'...
remote: Counting objects: 249, done.
remote: Total 249 (delta 0), reused 0 (delta 0), pack-reused 249
Receiving objects: 100% (249/249), 77.00 KiB | 0 bytes/s, done.
Resolving deltas: 100% (142/142), done.
$ cd Adafruit_Python_DHT
$ sudo apt-get install build-essential python-dev
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
build-essential python-dev
...
$ sudo python3 setup.py install
running install
running bdist_egg
running egg_info
creating Adafruit_DHT.egg-info
...
$
```

### The Plugin

This is the code for the plugin:

```
# -*- coding: utf-8 -*-

# FLEDGE_BEGIN
# See: http://fledge-iot.readthedocs.io/
# FLEDGE_END

""" Plugin for a DHT11 temperature and humidity sensor attached directly
    to the GPIO pins of a Raspberry Pi
```

(continues on next page)



(continued from previous page)

*This plugin uses the Adafruit DHT library, to install this perform the following steps:*

```
git clone https://github.com/adafruit/Adafruit_Python_DHT.git
cd Adafruit_Python_DHT
sudo apt-get install build-essential python-dev
sudo python setup.py install
```

*To access the GPIO pins fledge must be able to access /dev/gpiomem, the default access for this is owner and group read/write. Either Fledge must be added to the group or the permissions altered to allow Fledge access to the device.*

"""

```
from datetime import datetime, timezone
import uuid
```

```
from fledge.common import logger
from fledge.services.south import exceptions
```

```
__author__ = "Mark Riddoch"
__copyright__ = "Copyright (c) 2017 OSIsoft, LLC"
__license__ = "Apache 2.0"
__version__ = "${VERSION}"
```

```
_DEFAULT_CONFIG = {
    'plugin': {
        'description': 'Python module name of the plugin to load',
        'type': 'string',
        'default': 'dht11'
    },
    'pollInterval': {
        'description': 'The interval between poll calls to the device poll routine_
↳expressed in milliseconds.',
        'type': 'integer',
        'default': '1000'
    },
    'gpiopin': {
        'description': 'The GPIO pin into which the DHT11 data pin is connected',
        'type': 'integer',
        'default': '4'
    }
}
```

```
_LOGGER = logger.setup(__name__)
""" Setup the access to the logging system of Fledge """
```

```
def plugin_info():
    """ Returns information about the plugin.

    Args:
    Returns:
        dict: plugin information
    Raises:
```

(continues on next page)



(continued from previous page)

```

"""

return {
    'name': 'DHT11 GPIO',
    'version': '1.0',
    'mode': 'poll',
    'type': 'south',
    'interface': '1.0',
    'config': _DEFAULT_CONFIG
}

def plugin_init(config):
    """ Initialise the plugin.

    Args:
        config: JSON configuration document for the device configuration category
    Returns:
        handle: JSON object to be used in future calls to the plugin
    Raises:
        """

    handle = config['gpiopin']['value']
    return handle

def plugin_poll(handle):
    """ Extracts data from the sensor and returns it in a JSON document as a Python_
    ↪dict.

    Available for poll mode only.

    Args:
        handle: handle returned by the plugin initialisation call
    Returns:
        returns a sensor reading in a JSON document, as a Python dict, if it is_
    ↪available
        None - If no reading is available
    Raises:
        DataRetrievalError
        """

    try:
        humidity, temperature = Adafruit_DHT.read_retry(Adafruit_DHT.DHT11, handle)
        if humidity is not None and temperature is not None:
            time_stamp = str(datetime.now(tz=timezone.utc))
            readings = {'temperature': temperature, 'humidity': humidity}
            wrapper = {
                'asset': 'dht11',
                'timestamp': time_stamp,
                'key': str(uuid.uuid4()),
                'readings': readings
            }
            return wrapper
        else:
            return None

```

(continues on next page)



(continued from previous page)

```

except Exception as ex:
    raise exceptions.DataRetrievalError(ex)

return None

def plugin_reconfigure(handle, new_config):
    """ Reconfigures the plugin, it should be called when the configuration of the
    ↪ plugin is changed during the
        operation of the device service.
        The new configuration category should be passed.

    Args:
        handle: handle returned by the plugin initialisation call
        new_config: JSON object representing the new configuration category for the
    ↪ category
    Returns:
        new_handle: new handle to be used in the future calls
    Raises:
        """

    new_handle = new_config['gpiopin']['value']
    return new_handle

def plugin_shutdown(handle):
    """ Shutdowns the plugin doing required cleanup, to be called prior to the device
    ↪ service being shut down.

    Args:
        handle: handle returned by the plugin initialisation call
    Returns:
    Raises:
        """
    pass

```

## Building Fledge and Adding the Plugin

If you have not built Fledge yet, follow the steps described . After the build, you can optionally install Fledge following steps.

- If you have started Fledge from the build directory, copy the structure of the *fledge-south-dht11/python/* directory into the *python* directory:

```

$ cd ~/Fledge
$ cp -R ~/fledge-south-dht11/python/fledge/plugins/south/dht11 python/fledge/plugins/
↪ south/
$

```

- If you have installed Fledge by executing `sudo make install`, copy the structure of the *fledge-south-dht11/python/* directory into the installed *python* directory:

```

$ sudo cp -R ~/fledge-south-dht11/python/fledge/plugins/south/dht11 /usr/local/fledge/
↪ python/fledge/plugins/south/
$

```



**Note:** If you have installed Fledge using an alternative *DESTDIR*, remember to add the path to the destination directory to the `cp` command.

---

- Add service

```
$ curl -sX POST http://localhost:8081/fledge/service -d '{"name": "dht11", "type":  
→"south", "plugin": "dht11", "enabled": true}'
```

**Note:** Each plugin repo has its own debian packaging script and documentation, And that is the recommended way to go! As above method(s) may need explicit action for linux and/or python dependencies installation.

---

### Using the Plugin

Once south plugin is added as an enabled service, You are ready to use the DHT11 plugin.

```
$ curl -X GET http://localhost:8081/fledge/service | jq
```

Let's see what we have collected so far:

```
$ curl -s http://localhost:8081/fledge/asset | jq  
[  
  {  
    "count": 158,  
    "asset_code": "dht11"  
  }  
]  
$
```

Finally, let's extract some values:

```
$ curl -s http://localhost:8081/fledge/asset/dht11?limit=5 | jq  
[  
  {  
    "timestamp": "2017-12-30 14:41:39.672",  
    "reading": {  
      "temperature": 19,  
      "humidity": 62  
    }  
  },  
  {  
    "timestamp": "2017-12-30 14:41:35.615",  
    "reading": {  
      "temperature": 19,  
      "humidity": 63  
    }  
  },  
  {  
    "timestamp": "2017-12-30 14:41:34.087",  
    "reading": {  
      "temperature": 19,  
      "humidity": 62  
    }  
  },  
]
```

(continues on next page)



(continued from previous page)

```
{
  "timestamp": "2017-12-30 14:41:32.557",
  "reading": {
    "temperature": 19,
    "humidity": 63
  }
},
{
  "timestamp": "2017-12-30 14:41:31.028",
  "reading": {
    "temperature": 19,
    "humidity": 63
  }
}
]
$
```

Clearly we will not see many changes in temperature or humidity, unless we place our thumb on the sensor or we blow warm breathe on it :-)

```
$ curl -s http://localhost:8081/fledge/asset/dht11?limit=5 | jq
[
  {
    "timestamp": "2017-12-30 14:43:16.787",
    "reading": {
      "temperature": 25,
      "humidity": 95
    }
  },
  {
    "timestamp": "2017-12-30 14:43:15.258",
    "reading": {
      "temperature": 25,
      "humidity": 95
    }
  },
  {
    "timestamp": "2017-12-30 14:43:13.729",
    "reading": {
      "temperature": 24,
      "humidity": 95
    }
  },
  {
    "timestamp": "2017-12-30 14:43:12.201",
    "reading": {
      "temperature": 24,
      "humidity": 95
    }
  },
  {
    "timestamp": "2017-12-30 14:43:05.616",
    "reading": {
      "temperature": 22,
      "humidity": 95
    }
  }
]
```

(continues on next page)



(continued from previous page)

```

    }
}
$

```

Needless to say, the North plugin will send the buffered data to the PI system using the OMF plugin or any other north system using the appropriate north plugin.



06\_dht11\_tags\_in\_PI.jpg

## 13.6 South Plugins in C

South plugins written in C/C++ are no different in use to those written in Python, it is merely a case that they are implemented in a different language. The same options of polled or asynchronous methods still exist and the enduser of Fledge is not aware in which language the plugin has been written.

### 13.6.1 Polled Mode

Polled mode is the simplest form of South plugin that can be written, a poll routine is called at an interval defined in the plugin advanced configuration. The South service determines the type of the plugin by examining the mode property in the information the plugin returns from the *plugin\_info* call.

#### Plugin Poll

The plugin *poll* method is called periodically to collect the readings from a poll mode sensor. As with all other calls the argument passed to the method is the handle returned by the *plugin\_init* call, the return of the method should be a *Reading* instance that contains the data read.

The *Reading* class consists of

Property	Description
assetName	The asset key of the sensor device that is being read
userTimestamp	A timestamp for the reading data
datapoints	The reading data itself as a set of datapoint instances

More detail regarding the *Reading* class can be found in the section .

It is important that the *poll* method does not block as this will prevent the proper operation of the South microservice. Using the example of our simple DHT11 device attached to a GPIO pin, the *poll* routine could be:

```

/**
 * Poll for a plugin reading
 */
Reading plugin_poll(PLUGIN_HANDLE *handle)
{
    DHT11 *dht11 = (DHT11*)handle;

```

(continues on next page)



(continued from previous page)

```

    return dht11->takeReading();
}

```

Where our *DHT11* class has a method *takeReading* as follows

```

/**
 * Take reading from sensor
 *
 * @param firstReading This flag indicates whether this is the first reading to be
 * taken from sensor,
 * if so get it reliably even if takes multiple retries.
 * Subsequently (firstReading=false),
 * if reading from sensor fails, last good reading is returned.
 */
Reading DHT11::takeReading(bool firstReading)
{
    static uint8_t sensorData[4] = {0,0,0,0};

    bool valid = false;
    unsigned int count=0;
    do {
        valid = readSensorData(sensorData);
        count++;
    } while(!valid && firstReading && count < MAX_SENSOR_READ_RETRIES);

    if (firstReading && count >= MAX_SENSOR_READ_RETRIES)
        Logger::getLogger()->error("Unable to get initial valid reading from
DHT11 sensor connected to pin %d even after %d tries", m_pin, MAX_SENSOR_READ_
RETRIES);

    vector<Datapoint *> vec;

    ostringstream tmp;
    tmp << ((unsigned int)sensorData[0]) << "." << ((unsigned int)sensorData[1]);
    DatapointValue dpv1(stod(tmp.str()));
    vec.push_back(new Datapoint("Humidity", dpv1));

    ostringstream tmp2;
    tmp2 << ((unsigned int)sensorData[2]) << "." << ((unsigned_
int)sensorData[3]);
    DatapointValue dpv2(stod(tmp2.str()));
    vec.push_back(new Datapoint("Temperature", dpv2));

    return Reading(m_assetName, vec);
}

```

We are creating two *DatapointValues* for the Humidity and Temperature values returned by reading the DHT11 sensor.



## Plugin Poll Returning Multiple Values

It is possible in a C/C++ plugin to have a plugin that returns multiple readings in a single call to a poll routine. This is done by setting the interface version of 2.0.0 rather than 1.0.0. In this interface version the *plugin\_poll* call returns a vector of *Reading* rather than a single *Reading*.

```
/**
 * Poll for a plugin reading
 */
std::vector<Reading *> *plugin_poll(PLUGIN_HANDLE *handle)
{
    Modbus *modbus = (Modbus *)handle;

    if (!handle)
        throw runtime_error("Bad plugin handle");
    return modbus->takeReading();
}
```

## 13.6.2 Async IO Mode

In asyncio mode the plugin runs either a separate thread or uses some incoming event from a device or callback mechanism to trigger sending data to Fledge. The asynchronous mode uses two additional entry points to the plugin, one to register a callback on which the plugin sends data, *plugin\_register\_ingest* and another to start the asynchronous behavior *plugin\_start*.

### Plugin Register Ingest

The *plugin\_register\_ingest* call is used to allow the south service to pass a callback function to the plugin that the plugin uses to send data to the service every time the plugin has some new data.

```
/**
 * Register ingest callback
 */
void plugin_register_ingest(PLUGIN_HANDLE *handle, INGEST_CB cb, void *data)
{
    MyPluginClass *plugin = (MyPluginClass *)handle;

    if (!handle)
        throw new exception();
    plugin->registerIngest(data, cb);
}
```

The plugin should store the callback function pointer and the data associated with the callback such that it can use that information to pass a reading to the south service. The following code snippets show how a plugin class might store the callback and data and then use it to send readings into Fledge at a later stage.

```
/**
 * Record the ingest callback function and data in member variables
 */
void MyPluginClass::registerIngest(void *data, INGEST_CB cb)
{
    @param data The Ingest function data
    @param cb The callback function to call
}
```

(continues on next page)



(continued from previous page)

```

        m_ingest = cb;
        m_data = data;
    }

    /**
     * Called when a data is available to send to the south service
     *
     * @param points      The points in the reading we must create
     */
    void MyPluginClass::ingest(Reading& reading)
    {
        (*m_ingest)(m_data, reading);
    }

```

## Plugin Start

The *plugin\_start* method, as with other plugin calls, is called with the plugin handle data that was returned from the *plugin\_init* call. The *plugin\_start* call will only be called once for a plugin, it is the responsibility of *plugin\_start* to take whatever action is required in the plugin in order to start the asynchronous actions of the plugin. This might be to start a thread, register an endpoint for a remote connection or call an entry point in a third party library to start asynchronous processing.

```

    /**
     * Start the Async handling for the plugin
     */
    void plugin_start(PLUGIN_HANDLE *handle)
    {
        MyPluginClass *plugin = (MyPluginClass *)handle;

        if (!handle)
            return;
        plugin->start();
    }

    /**
     * Start the asynchronous processing thread
     */
    void MyPluginClass::start()
    {
        m_running = true;
        m_thread = new thread(threadWrapper, this);
    }

```



### 13.6.3 Set Point Control

South plugins can also be used to exert control on the underlying device to which they are connected. This is not intended for use as a substitute for real time control systems, but rather as a mechanism to make non-time critical changes to a device or to trigger an operation on the device.

To make a south plugin support control features there are two steps that need to be taken

- Tag the plugin as supporting control
- Add the entry points for control

#### Enable Control

A plugin enables control features by means of the flags in the plugin information data structure which is returned by the *plugin\_info* entry point of the plugin. The flag value *SP\_CONTROL* should be added to the flags of the plugin.

```
/**
 * The plugin information structure
 */
static PLUGIN_INFORMATION info = {
    PLUGIN_NAME,           // Name
    VERSION,               // Version
    SP_CONTROL,            // Flags - add control
    PLUGIN_TYPE_SOUTH,     // Type
    "1.0.0",              // Interface version
    CONFIG                 // Default configuration
};
```

Adding this flag will cause the south service to do a number of things when it loads the plugin;

- The south service will attempt to resolve the two control entry points.
- A toggle will be added to the advanced configuration category of the service that will permit the disabling of control services.
- A security category will be added to the south service that contains the access control lists and permissions associated with the service.

#### Control Entry Points

Two entry points are supported for control operations in the south plugin

- **plugin\_write**: which is used to set the value of a parameter within the plugin or device
- **plugin\_operation**: which is used to perform an operation on the plugin or device

The south plugin can support one or both of these entry points as appropriate for the plugin.



## Write Entry Point

The write entry point is used to set data in the plugin or write data into the device.

The plugin write entry point is defined as follows

```
bool plugin_write(PLUGIN_HANDLE *handle, string name, string value)
```

Where the parameters are;

- **handle** the handle of the plugin instance
- **name** the name of the item to be changed
- **value** a string presentation of the new value to assign top the item

The return value defines if the write was successful or not. True is returned for a successful write.

```
bool plugin_write(PLUGIN_HANDLE *handle, string& name, string& value)
{
    Random *random = (Random *)handle;

    return random->write(operation, name, value);
}
```

In this case the main logic of the write operation is implemented in a class that contains all the plugin logic. Note that the assumption here, and a design pattern often used by plugin writers, is that the *PLUGIN\_HANDLE* is actually a pointer to a C++ class instance.

In this case the implementation in the plugin class is as follows:

```
bool Random::write(string& name, string& value)
{
    if (name.compare("mode") == 0)
    {
        if (value.compare("relative") == 0)
        {
            m_mode = RELATIVE_MODE;
        }
        else if (value.compare("absolute") == 0)
        {
            m_mode = ABSOLUTE_MODE;
        }
        Logger::getLogger()->error("Unknown mode requested '%s' ignored.",
↪value.c_str());
        return false;
    }
    else
    {
        Logger::getLogger()->error("Unknown control item '%s' ignored.", name.c_
↪str());
        return false;
    }
    return true;
}
```

In this case the code is relatively simple as we assume there is a single control parameter that can be written, the mode of operation. We look for the known name and if a different name is passed an error is logged and false is returned. If the correct name is passed in we then check the value and take the appropriate action. If the value is not a recognized value then an error is logged and we again return false.



In this case we are merely setting a value within the plugin, this could equally well be done via configuration and would in that case be persisted between restarts. Normally control would not be used for this, but rather for making a change with the connected device itself, such as changing a PLC register value. This is simply an example to demonstrate the mechanism.

### Operation Entry Point

The plugin will support an operation entry point. This will execute the given operation synchronously, it is expected that this operation entry point will be called using a separate thread, therefore the plugin should implement operations in a thread safe environment.

The plugin write operation entry point is defined as follows

```
bool plugin_operation(PLUGIN_HANDLE *handle, string& operation, int count, PLUGIN_
↳PARAMETER **params)
```

Where the parameters are;

- **handle** the handle of the plugin instance
- **operation** the name of the operation to be executed
- **count** the number of parameters
- **params** a set of name/value pairs that are passed to the operation

The *operation* parameter should be used by the plugin to determine which operation is to be performed, that operation may also be passed a number of parameters. The count of these parameters are passed to the plugin in the *count* argument and the actual parameters are passed in an array of key/value pairs as strings.

The return from the call is a boolean result of the operation, a failure of the operation or a call to an unrecognized operation should be indicated by returning a false value. If the operation succeeds a value of true should be returned.

The following example shows the implementation of the plugin operation entry point.

```
bool plugin_operation(PLUGIN_HANDLE *handle, string& operation, int count, PLUGIN_
↳PARAMETER **params)
{
    Random *random = (Random *)handle;

    return random->operation(operation, count, params);
}
```

In this case the main logic of the operation is implemented in a class that contains all the plugin logic. Note that the assumption here, and a design pattern often used by plugin writers, is that the *PLUGIN\_HANDLE* is actually a pointer to a C++ class instance.

In this case the implementation in the plugin class is as follows:

```
/**
 * SetPoint operation. We support reseeding the random number generator
 */
bool Random::operation(const std::string& operation, int count, PLUGIN_PARAMETER_
↳**params)
{
    if (operation.compare("seed") == 0)
    {
        if (count)
        {
```

(continues on next page)



(continued from previous page)

```

        if (params[0]->name.compare("seed"))
        {
            long seed = strtol(params[0]->value.c_str(), NULL, 10);

            srand(seed);
        }
        else
        {
            return false;
        }
    }
    else
    {
        srand(time(0));
    }
    Logger::getLogger()->info("Reseeded random number generator");
    return true;
}
Logger::getLogger()->error("Unrecognised operation %s", operation.c_str());
return false;
}

```

In this example, the operation method checks the name of the operation to perform, only a single operation is supported by this plugin. If this operation name differs the method will log an error and return false. If the operation is recognized it will check for any arguments passed in, retrieve and use it. In this case an optional *seed* argument may be passed.

There is no actual machine connected here, therefore the operation occurs within the plugin. In the case of a real machine the operation would most likely cause an action on a machine, for example a request to the machine to re-calibrate itself.

### 13.6.4 A South Plugin Example In C/C++: the DHT11 Sensor

Using the same example as before, the DHT11 temperature and humidity sensor, let's look at how to create the plugin in C/C++.

#### The Software

For this plugin we use the wiringpi C library to connect to the hardware of the Raspberry Pi

```

$ sudo apt-get install wiringpi
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
wiringpi
...
$

```



## The Plugin

This is the code for the plugin.cpp file that provides the plugin API:

```
/*
 * Fledge south plugin.
 *
 * Copyright (c) 2018 OSisoft, LLC
 *
 * Released under the Apache 2.0 Licence
 *
 * Author: Amandeep Singh Arora
 */
#include <dht11.h>
#include <plugin_api.h>
#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
#include <string>
#include <logger.h>
#include <plugin_exception.h>
#include <config_category.h>
#include <rapidjson/document.h>
#include <version.h>

using namespace std;
#define PLUGIN_NAME "dht11_V2"

/**
 * Default configuration
 */
const static char *default_config = QUOTE({
    "plugin" : {
        "description" : "DHT11 C south plugin",
        "type" : "string",
        "default" : PLUGIN_NAME,
        "readonly": "true"
    },
    "asset" : {
        "description" : "Asset name",
        "type" : "string",
        "default" : "dht11",
        "order": "1",
        "displayName": "Asset Name",
        "mandatory" : "true"
    },
    "pin" : {
        "description" : "Rpi pin to which DHT11 is attached",
        "type" : "integer",
        "default" : "7",
        "displayName": "Rpi Pin"
    }
});

/**
 * The DHT11 plugin interface
 */
```

(continues on next page)



(continued from previous page)

```

extern "C" {

/**
 * The plugin information structure
 */
static PLUGIN_INFORMATION info = {
    PLUGIN_NAME,           // Name
    VERSION,               // Version
    0,                    // Flags
    PLUGIN_TYPE_SOUTH,     // Type
    "1.0.0",              // Interface version
    default_config         // Default configuration
};

/**
 * Return the information about this plugin
 */
PLUGIN_INFORMATION *plugin_info()
{
    return &info;
}

/**
 * Initialise the plugin, called to get the plugin handle
 */
PLUGIN_HANDLE plugin_init(ConfigCategory *config)
{
    unsigned int pin;

    if (config->itemExists("pin"))
    {
        pin = stoul(config->getValue("pin"), nullptr, 0);
    }

    DHT11 *dht11= new DHT11(pin);

    if (config->itemExists("asset"))
        dht11->setAssetName(config->getValue("asset"));
    else
        dht11->setAssetName("dht11");

    Logger::getLogger()->info("m_assetName set to %s", dht11->getAssetName());

    return (PLUGIN_HANDLE)dht11;
}

/**
 * Poll for a plugin reading
 */
Reading plugin_poll(PLUGIN_HANDLE *handle)
{
    DHT11 *dht11 = (DHT11*)handle;
    return dht11->takeReading();
}

/**
 * Reconfigure the plugin

```

(continues on next page)



(continued from previous page)

```

*/
void plugin_reconfigure(PLUGIN_HANDLE *handle, string& newConfig)
{
    ConfigCategory      conf("dht", newConfig);
    DHT11 *dht11 = (DHT11*)*handle;

    if (conf.itemExists("asset"))
        dht11->setAssetName(conf.getValue("asset"));
    if (conf.itemExists("pin"))
    {
        unsigned int pin = stoul(conf.getValue("pin"), nullptr, 0);
        dht11->setPin(pin);
    }
}

/**
 * Shutdown the plugin
 */
void plugin_shutdown(PLUGIN_HANDLE *handle)
{
    DHT11 *dht11 = (DHT11*)handle;
    delete dht11;
}
};

```

The full source code, including the *DHT11* class can be found in [GitHub](#)

## Building Fledge and Adding the Plugin

If you have not built Fledge yet, follow the steps described . After the build, you can optionally install Fledge following steps.

- Clone the *fledge-south-dht* repository

```

$ git clone https://github.com/fledge-iot/fledge-south-dht.git
...
$

```

- Set the environment variable `FLEDGE_ROOT` to the directory in which you built Fledge

```

$ export FLEDGE_ROOT=~/.fledge
$

```

- Go to the location in which you cloned the *fledge-south-dht* repository and create a build directory and run `cmake` in that directory

```

$ cd ~/.fledge-south-dht
$ mkdir build
$ cd build
$ cmake ..
...
$

```

- Now make the plugin



```
$ make
$
```

- If you have started Fledge from the build directory, copy the plugin into the destination directory

```
$ mkdir -p $FLEDGE_ROOT/plugins/south/dht
$ cp libdht.so $FLEDGE_ROOT/plugins/south/dht
$
```

- If you have installed Fledge by executing `sudo make install`, copy the plugin into the destination directory

```
$ sudo mkdir -p /usr/local/fledge/plugins/south/dht
$ sudo cp libdht.so /usr/local/fledge/plugins/south/dht
$
```

**Note:** If you have installed Fledge using an alternative *DESTDIR*, remember to add the path to the destination directory to the `cp` command.

- Add service

```
$ curl -sX POST http://localhost:8081/fledge/service -d '{"name": "dht", "type":
↪ "south", "plugin": "dht", "enabled": true}'
```

You may now use the C/C++ plugin in exactly the same way as you used a Python plugin earlier.

## 13.7 C++ Support Classes

A number of support classes exist within the common library that forms part of every Fledge plugin.

### 13.7.1 Reading

The *Reading* class and the associated *Datapoint* and *DatapointValue* classes provide the mechanism within C++ classes to manipulated the reading asset data. The public part of the *Reading* class is currently defined as follows;

```
class Reading {
public:
    Reading(const std::string& asset, Datapoint *value);
    Reading(const std::string& asset, std::vector<Datapoint *> values);
    Reading(const std::string& asset, std::vector<Datapoint *> values,
↪const std::string& ts);
    Reading(const Reading& orig);

    ~Reading();
    void addDatapoint(Datapoint *value);
    Datapoint *removeDatapoint(const std::string&
↪name);

    std::string toJSON(bool minimal = false) const;
    std::string getDatapointsJSON() const;
    // Return AssetName
    const std::string& getAssetName() const { return m_asset;
↪ };
};
```

(continues on next page)



(continued from previous page)

```

        // Set AssetName
        void setAssetName(std::string assetName) {
    ↪ m_asset = assetName; };
        unsigned int getDatapointCount() { return m_values.
    ↪ size(); };
        void removeAllDatapoints();
        // Return Reading datapoints
        const std::vector<Datapoint *> getReadingData() const { return m_
    ↪ values; };
        // Return reference to Reading datapoints
        std::vector<Datapoint *>& getReadingData() { return m_values; };
        unsigned long getId() const { return m_id; };
        unsigned long getTimestamp() const { return
    ↪ (unsigned long)m_timestamp.tv_sec; };
        unsigned long getUserTimestamp() const { return
    ↪ (unsigned long)m_userTimestamp.tv_sec; };
        void setId(unsigned long id) { m_id = id; }
    ↪;
        void setTimestamp(unsigned long ts) { m_
    ↪ timestamp.tv_sec = (time_t)ts; };
        void setTimestamp(struct timeval tm) { m_
    ↪ timestamp = tm; };
        void setTimestamp(const std::string&
    ↪ timestamp);
        void getTimestamp(struct timeval *tm) {
    ↪ *tm = m_timestamp; };
        void setUserTimestamp(unsigned long uTs) {
    ↪ m_userTimestamp.tv_sec = (time_t)uTs; };
        void setUserTimestamp(struct timeval tm) {
    ↪ m_userTimestamp = tm; };
        void setUserTimestamp(const std::string&
    ↪ timestamp);
        void getUserTimestamp(struct timeval *tm)
    ↪ { *tm = m_userTimestamp; };

        typedef enum dateTimeFormat { FMT_DEFAULT, FMT_STANDARD, FMT_ISO8601 }
    ↪ readingTimeFormat;

        // Return Reading asset time - ts time
        const std::string getAssetDateTime(readingTimeFormat dateTimeFmt =
    ↪ FMT_DEFAULT, bool addMs = true) const;
        // Return Reading asset time - user_ts time
        const std::string getAssetDateUserTime(readingTimeFormat dateTimeFmt
    ↪ = FMT_DEFAULT, bool addMs = true) const;
    }

```

The *Reading* class contains a number of items that are mapped to the JSON representation of data that is sent to the Fledge storage service and are used by the various services and plugins within Fledge.

- **Asset Name:** The name of the asset. The asset name is set in the constructor of the reading and retrieved via the *getAssetName()* method.
- **Timestamp:** The timestamp when the reading was first seen within Fledge.
- **User Timestamp:** The timestamp for the actual data in the reading. This may differ from the value of Timestamp if the device itself is able to supply a timestamp value.
- **Datapoints:** The actual data of a reading stored in a Datapoint class.



The *Datapoint* class provides a name for each data point within a *Reading* and the tagged type data for the reading value. The public definition of the *Datapoint* class is as follows;

```
class Datapoint {
public:
    /**
     * Construct with a data point value
     */
    Datapoint(const std::string& name, DatapointValue& value) : m_
    ↪name(name), m_value(value);
    ~Datapoint();
    /**
     * Return asset reading data point as a JSON
     * property that can be included within a JSON
     * document.
     */
    std::string toJSONProperty();
    const std::string getName() const;
    void setName(std::string name);
    const DatapointValue getData() const;
    DatapointValue& getData();
}
```

Closely associated with the *Datapoint* is the *DatapointValue* which uses a tagged union to store the values. The public definition of the *DatapointValue* is as follows;

```
class DatapointValue {
public:
    /**
     * Construct with a string
     */
    DatapointValue(const std::string& value)
    {
        m_value.str = new std::string(value);
        m_type = T_STRING;
    };
    /**
     * Construct with an integer value
     */
    DatapointValue(const long value)
    {
        m_value.i = value;
        m_type = T_INTEGER;
    };
    /**
     * Construct with a floating point value
     */
    DatapointValue(const double value)
    {
        m_value.f = value;
        m_type = T_FLOAT;
    };
    /**
     * Construct with an array of floating point values
     */
    DatapointValue(const std::vector<double>& values)
    {
        m_value.a = new std::vector<double>(values);
```

(continues on next page)



(continued from previous page)

```

        m_type = T_FLOAT_ARRAY;
    };

    /**
     * Construct with an array of Datapoints
     */
    DatapointValue(std::vector<Datapoint*>& values, bool isDict)
    {
        m_value.dpa = values;
        m_type = isDict? T_DP_DICT : T_DP_LIST;
    }

    /**
     * Construct with an Image
     */
    DatapointValue(const DPIImage& value)
    {
        m_value.image = new DPIImage(value);
        m_type = T_IMAGE;
    }

    /**
     * Construct with a DataBuffer
     */
    DatapointValue(const DataBuffer& value)
    {
        m_value.dataBuffer = new DataBuffer(value);
        m_type = T_DATABUFFER;
    }

    /**
     * Construct with an Image Pointer, the
     * image becomes owned by the datapointValue
     */
    DatapointValue(DPIImage *value)
    {
        m_value.image = value;
        m_type = T_IMAGE;
    }

    /**
     * Construct with a DataBuffer
     */
    DatapointValue(DataBuffer *value)
    {
        m_value.dataBuffer = value;
        m_type = T_DATABUFFER;
    }

    /**
     * Construct with a 2 dimensional array of floating point values
     */
    DatapointValue(const std::vector< std::vector<double> >& values)
    {
        m_value.a2d = new std::vector< std::vector<double> >();
        for (auto row : values)
        {

```

(continues on next page)



(continued from previous page)

```

        m_value.a2d->push_back(std::vector<double>(row));
    }
    m_type = T_2D_FLOAT_ARRAY;
};

/**
 * Copy constructor
 */
DatapointValue(const DatapointValue& obj);

/**
 * Assignment Operator
 */
DatapointValue& operator=(const DatapointValue& rhs);

/**
 * Destructor
 */
~DatapointValue();

/**
 * Set the value of a datapoint, this may
 * also cause the type to be changed.
 * @param value    An integer value to set
 */
void setValue(long value)
{
    m_value.i = value;
    m_type = T_INTEGER;
}

/**
 * Set the value of a datapoint, this may
 * also cause the type to be changed.
 * @param value    A floating point value to set
 */
void setValue(double value)
{
    m_value.f = value;
    m_type = T_FLOAT;
}

/** Set the value of a datapoint to be an image
 * @param value The image to set in the data point
 */
void setValue(const DPImage& value)
{
    m_value.image = new DPImage(value);
    m_type = T_IMAGE;
}

/**
 * Return the value as a string
 */
std::string toString() const;

```

(continues on next page)



(continued from previous page)

```

/**
 * Return string value without trailing/leading quotes
 */
std::string toStringValue() const { return *m_value.str; };

/**
 * Return long value
 */
long toInt() const { return m_value.i; };

/**
 * Return double value
 */
double toDouble() const { return m_value.f; };

// Supported Data Tag Types
typedef enum DatapointTag
{
    T_STRING,
    T_INTEGER,
    T_FLOAT,
    T_FLOAT_ARRAY,
    T_DP_DICT,
    T_DP_LIST,
    T_IMAGE,
    T_DATABUFFER,
    T_2D_FLOAT_ARRAY
} dataTagType;

/**
 * Return the Tag type
 */
dataTagType getType() const
{
    return m_type;
}

std::string getTypeStr() const
{
    switch(m_type)
    {
        case T_STRING: return std::string("STRING");
        case T_INTEGER: return std::string("INTEGER");
        case T_FLOAT: return std::string("FLOAT");
        case T_FLOAT_ARRAY: return std::string("FLOAT_ARRAY");
        case T_DP_DICT: return std::string("DP_DICT");
        case T_DP_LIST: return std::string("DP_LIST");
        case T_IMAGE: return std::string("IMAGE");
        case T_DATABUFFER: return std::string("DATABUFFER");
        case T_2D_FLOAT_ARRAY: return std::string("2D_FLOAT_
↪ARRAY");

        default: return std::string("INVALID");
    }
}

/**
 * Return array of datapoints
 */

```

(continues on next page)



(continued from previous page)

```

std::vector<Datapoint*>& getDpVec()
{
    return m_value.dpa;
}

/**
 * Return array of float
 */
std::vector<double*>& getDpArr()
{
    return m_value.a;
}

/**
 * Return 2D array of float
 */
std::vector<std::vector<double*>*>& getDp2DArr()
{
    return m_value.a2d;
}

/**
 * Return the Image
 */
DPIImage *getImage()
{
    return m_value.image;
}

/**
 * Return the DataBuffer
 */
DataBuffer *getDataBuffer()
{
    return m_value.dataBuffer;
}
};

```

The *DatapointValue* can store data in as a number of types

Type	C++ Representation
T_STRING	Pointer to std::string
T_INTEGER	long
T_FLOAT	double
T_FLOAT_ARRAY	Pointer to std::vector<double>
T_2D_FLOAT_ARRAY	Pointer to std::vector<std::vector<double>>
T_DP_DICT	Pointer to std::vector<Datapoint *>
T_DP_LIST	Pointer to std::vector<Datapoint *>
T_IMAGE	Pointer to DPIImage
T_DATABUFFER	Pointer to DataBuffer



### 13.7.2 Configuration Category

The *ConfigCategory* class is a support class for managing configuration information within a plugin and is passed to the plugin entry points. The public definition of the class is as follows;

```
class ConfigCategory {
public:
    enum ItemType {
        UnknownType,
        StringItem,
        EnumerationItem,
        JsonItem,
        BoolItem,
        NumberItem,
        DoubleItem,
        ScriptItem,
        CategoryType,
        CodeItem
    };

    ConfigCategory(const std::string& name, const std::string& json);
    ConfigCategory() {};
    ConfigCategory(const ConfigCategory& orig);
    ~ConfigCategory();

    void addItem(const std::string& name,
        ↪const std::string description,
        ↪const std::string def,
        ↪const std::string description,
        ↪std::string& value,
        ↪options);
    void removeItems();
    void removeItemsType(ItemType type);
    void keepItemsType(ItemType type);
    bool extractSubcategory(ConfigCategory &
        ↪subCategories);
    void setDescription(const std::string&
        ↪description);
    std::string getName() const;
    std::string getDescription() const;
    unsigned int getCount() const;
    bool itemExists(const std::string& name)
        ↪const;
    bool setItemDisplayName(const std::string&
        ↪name, const std::string& displayName);
    std::string getValue(const std::string& name)
        ↪const;
    std::string getType(const std::string& name)
        ↪const;
    std::string getDescription(const std::string&
        ↪name) const;
    std::string getDefault(const std::string& name)
        ↪const;
    bool setDefault(const std::string& name,
        ↪const std::string& value);
```

(continues on next page)



(continued from previous page)

```

std::string
↪name) const;
std::vector<std::string>
↪const;
std::string
↪const;
std::string
↪const;
std::string
↪const;
bool
↪const;
bool
↪name) const;
bool
bool
bool
↪const;
bool
↪const;
bool
↪const;
std::string
std::string
↪const;
ConfigCategory&
ConfigCategory&
void
void
std::string
↪itemName) const;
enum ItemAttribute
↪MANDATORY_ATTR, FILE_ATTR;
std::string
↪itemName,
ItemAttribute
↪itemAttribute) const;
}

getDisplayNames(const std::string& name)
getOptions(const std::string& name)
getLength(const std::string& name)
getMinimum(const std::string& name)
getMaximum(const std::string& name)
isString(const std::string& name)
isEnumeration(const std::string& name)
isJSON(const std::string& name) const;
isBool(const std::string& name) const;
isNumber(const std::string& name)
isDouble(const std::string& name)
isDeprecated(const std::string& name)
toJson(const bool full=false) const;
itemsToJson(const bool full=false)
operator=(ConfigCategory const& rhs);
operator+=(ConfigCategory const& rhs);
setItemsValueFromDefault();
checkDefaultValuesOnly() const;
itemToJson(const std::string& name,
ItemAttribute attr) const;

```

Although *ConfigCategory* is a complex class, only a few of the methods are commonly used within a plugin

- **itemExists:** - used to test if an expected configuration item exists within the configuration category.
- **getValue:** - return the value of a configuration item from within the configuration category
- **isBool:** - tests if a configuration item is of boolean type
- **isNumber:** - tests if a configuration item is a number
- **isDouble:** - tests if a configuration item is valid to be represented as a double
- **isString:** - tests if a configuration item is a string



### 13.7.3 Logger

The *Logger* class is used to write entries to the syslog system within Fledge. A singleton *Logger* exists which can be obtained using the following code snippet;

```
Logger *logger = Logger::getLogger();
logger->error("An error has occurred within the plugin processing");
```

It is then possible to log messages at one of five different log levels; *debug*, *info*, *warn*, *error* or *fatal*. Messages may be logged using standard printf formatting strings. The public definition of the *Logger* class is as follows;

```
class Logger {
public:
    Logger(const std::string& application);
    ~Logger();
    static Logger *getLogger();
    void debug(const std::string& msg, ...);
    void printLongString(const std::string&);
    void info(const std::string& msg, ...);
    void warn(const std::string& msg, ...);
    void error(const std::string& msg, ...);
    void fatal(const std::string& msg, ...);
    void setMinLevel(const std::string& level);
};
```

The various log levels should be used as follows;

- **debug**: should be used to output messages that are relevant only to a programmer that is debugging the plugin.
- **info**: should be used for information that is meaningful to the end users, but should not normally be logged.
- **warn**: should be used for warning messages that will normally be logged but reflect a condition that does not prevent the plugin from operating.
- **error**: should be used for conditions that cause a temporary failure in processing within the plugin.
- **fatal**: should be used for conditions that cause the plugin to fail processing permanently, possibly requiring a restart of the microservice in order to resolve.

## 13.8 Hybrid Plugins

In addition to plugins written in Python and C/C++ it is possible to have a hybrid plugin that is a combination of an existing plugin and configuration for that plugin. This is useful in a situation whereby there are multiple sensors or devices that you connect to Fledge that have common configuration. It allows devices to be added without repeating the common configuration.

Using our example of a *DHT11* sensor connected to a GPIO pin, if we wanted to create a new plugin for a *DHT11* that was always connected to pin 4 then we could do this by creating a JSON file as below that supplies a fixed default value for the GPIO pin.

```
{
  "description" : "A DHT11 sensor connected to GPIO pin 4",
  "name" : "DHT11-4",
  "connection" : "DHT11",
  "defaults" : {
    "pin" : {
      "default" : "4"
    }
  }
}
```

(continues on next page)



(continued from previous page)

```

    }
}
}

```

This creates a new hybrid plugin called DHT11-4 that is installed by copying this file into the `plugins/south/DHT11-4` directory of your installation. Once installed it can be treated as any other south plugin within Fledge. The effect of this hybrid plugin is to load the *DHT11* plugin and always set the configuration parameter called “pin” to the value “4”. The item “pin” will be hidden from the user in the Fledge GUI when they create the instance of the plugin. This allows for a simpler and more streamlined user experience when adding plugins with common configuration.

The items in the JSON file are;

Name	Description
description	A description of the hybrid plugin. This will appear the right of the selection list in the Fledge user interface when the plugin is selected.
name	The name of the plugin itself. This must match the filename of the JSON file and also the name of the directory the file is placed in.
connection	The name of the underlying plugin that will be used as the basis for this hybrid plugin. This must be a C/C++ or Python plugin, it can not be another hybrid plugin.
defaults	The set of values to default in this hybrid plugin. These are configuration parameters of the underlying plugin that will be fixed in the hybrid plugin. Each hybrid plugin can have one or many values here.

It may not be difficult to enter the GPIO pin in each case in this example, where it becomes more useful is for plugins such as *Modbus* where a complex map is required to be entered in a JSON document. By using a hybrid plugin we can define the map we need once and then add new sensors of the same type without having to repeat the map. An example of this would be the Flir AX8 camera that requires a total of 176 Modbus registers to be mapped into 88 different values in an asset. A hybrid plugin *fledge-south-FlirAX8* defines that mapping once and as a result adding a new Flir AX8 camera is as simple as selecting the FlirAX8 hybrid plugin and entering the IP address of the camera.

## 13.9 North Plugins

North plugins are used in North tasks and micro services to extract data buffered in Fledge and send it Northbound, i.e. to a server or a service in the Cloud or in an Enterprise data center. North plugins may be written in Python or C/C++, a number of different north plugins are available as examples that may be used when creating new plugins.

A north plugin has a limited number of entry points that it must support, these entry points are the same for both Python and C/C++ north plugins.

Entry Point	Description
plugin_info	Return information about the plugin including the configuration for the plugin. This is the same as <code>plugin_info</code> in all other types of plugin and is part of the standard plugin interface.
plugin_init	Also part of the standard plugin interface. This call is passed the request configuration of the plugin and should be used to do any initialization of the plugin.
plugin_send	This entry point is the north plugin specific entry point that is used to send data from Fledge. This will be called repeatedly with blocks of readings.
plugin_shutdown	Part of the standard plugin interface, this will be called when the plugin is no longer required and will be the final call to the plugin.
plugin_register	Register the callback function used for control writes and operations.



The life cycle of a plugin is very similar regardless of if it is written in Python or C/C++, the *plugin\_info* call is made first to determine data about the plugin. The plugin is then initialized by calling the *plugin\_init* entry point. The *plugin\_send* entry point will be called multiple times to send the actual data and finally the *plugin\_shutdown* entry point will be called.

In the following sections each of these calls will be described in detail and samples given in both C/C++ and Python.

### 13.9.1 Python Plugins

Python plugins are loaded dynamically and executed either within a task, known as the *sending\_task* or *north* task. This code is implemented in C++ and embedded a Python interpreter that is used to run the Python plugin.

#### The *plugin\_info* call

The *plugin\_info* call is the first call that will be made to a plugin and is called only once. It is part of the standard plugin interface that is implemented by north, south, filter, notification rule and notification delivery plugins. No arguments are passed to this call and it should return a *plugin information structure* as a Python dict.

A typical implementation for a simple north plugin simply returns a DICT as follows

```
def plugin_info():
    """ Used only once when call will be made to a plugin.

    Args:
    Returns:
        Information about the plugin including the configuration for the plugin
    """
    return {
        'name': 'http',
        'version': '1.9.1',
        'type': 'north',
        'interface': '1.0',
        'config': _DEFAULT_CONFIG
    }
```

The items in the structure returned by *plugin\_info* are

Name	Description
name	The name of the plugin
version	The version of the plugin. Typically this is the same as the version of Fledge it is designed to work with but is not constrained to be the same.
type	The type of the plugin, in this case the type will always be <i>north</i>
interface	The version of the plugin interface that the plugin supports. In this case the version is 1.0
config	The DICT that defines the configuration that the plugin has as default.

In the case above *\_DEFAULT\_CONFIG* is another Python DICT that contains the defaults for the plugin configuration and will be covered in the Configuration section.



## Configuration

Configuration within Fledge is represented in a JSON structure that defines a name, value, default, type and a number of other optional parameters. The configuration process works by the plugins having a default configuration that they return from the `plugin_init` call. The Fledge configuration code will then combine this with a copy of that configuration that it holds. On the first time a service is created, with no previously held configuration, the configuration manager will take the default values and make those the actual values. The user may then update these to set non-default values. In subsequent executions of the plugin these values will be combined with the defaults to create the in use configuration that is passed to the `plugin_init` entry point. The mechanism is designed to allow initial execution of a plugin, but also to allow upgrade of a plugin to create new configuration items for the plugins whilst preserving previous configuration values set by the user.

A sample default configuration of http north python based plugin is shown below.

```
{
  "plugin": {
    "description": "HTTP North Plugin",
    "type": "string",
    "default": "http_north",
    "readonly": "true"
  },
  "url": {
    "description": "Destination URL",
    "type": "string",
    "default": "http://localhost:6683/sensor-reading",
    "order": "1",
    "displayName": "URL"
  },
  "source": {
    "description": "Source of data to be sent on the stream. May be either_
↪readings or statistics.",
    "type": "enumeration",
    "default": "readings",
    "options": ["readings", "statistics"],
    "order": "2",
    "displayName": "Source"
  },
  "verifySSL": {
    "description": "Verify SSL certificate",
    "type": "boolean",
    "default": "false",
    "order": "3",
    "displayName": "Verify SSL"
  }
}
```

Items marked as “*readonly*” : “*true*” will not be presented to the user. The *displayName* and *order* properties are only used by the user interface to display the configuration item. The description, type and default are used by the API to verify the input and also set the initial values when a new configuration item is created.

Rules can also be given to the user interface to define the validity of configuration items based upon the values of others, or example

```
{
  "applyFilter": {
    "description": "Should filter be applied before processing data",
    "type": "boolean",
```

(continues on next page)



(continued from previous page)

```

        "default": "false",
        "order": "4",
        "displayName": "Apply Filter"
    },
    "filterRule": {
        "description": "JQ formatted filter to apply (only applicable if applyFilter_
↪is True)",
        "type": "string",
        "default": "[]",
        "order": "5",
        "displayName": "Filter Rule",
        "validity": "applyFilter == \"true\""
    }
}

```

This will only allow entry to the *filterRule* configuration item if the *applyFilter* item has been set to true.

## The plugin\_init call

The *plugin\_init* call will be invoked after the *plugin\_info* call has been called to obtain the information regarding the plugin. This call is designed to allow the plugin to do any initialization that is required and also creates the handle will be used in all subsequent calls to identify the instance of the plugin.

The *plugin\_init* is passed a Python DICT as the only argument, this DICT contains the modified configuration for the plugin that is created by taking the default plugin configuration returned by *plugin\_info* and adding to that the values the user has configured previously. This is the working configuration that the plugin should use.

The typical implementation of the *plugin\_init* call will create an instance of a Python class which is the main body of the plugin. An object will then be returned which is the handle that will be passed into subsequent calls. This handle in a simple plugin, is commonly a Python DICT that is the configuration of the plugin, however any values may be returned. The caller treats the handle as opaque data that it stores and passed to further calls to the plugin, it will never look inside that object or have any expectations as to what is stored within that object.

The *fledge-north-http* plugin implementation of *plugin\_init* is shown below as an example

```

def plugin_init(data):
    """ Used for initialization of a plugin.

    Args:
        data - Plugin configuration
    Returns:
        Dictionary of a Plugin configuration
    """
    global http_north, config
    http_north = HttpNorthPlugin()
    config = data
    return config

```

In this case the plugin creates an object that implements the functionality and stores that object in a global variable. This can be done as only one instance of the north plugin exists within a single process. It is however perhaps better practice to return the instance of the class in the handle rather than use a global variable. Using a global is not recommended for filter plugins as multiple instances of a filter may exist within a single process. In this case the plugin uses the configuration as the handle it returns.



## The `plugin_send` call

The `plugin_send` call is the main entry point of a north plugin, it is used to send set of readings north to the destination system. It is responsible for both the communication to that system and the translation of the internal representation of the reading data to the representation required by the external system.

The communication performed by the `plugin_send` routine should use the Python 3 asynchronous I/O primitives, the definition of the `plugin_send` entry point must also use the `async` keyword.

The `plugin_send` entry point is passed 3 arguments, the plugin handle, the data to send and a `stream_id`.

```
async def plugin_send(handle, payload, stream_id):
```

The handle is the opaque data returned by the call to `plugin_init` and may be used by the plugin to store data between invocations. The `payload` is a set of readings that should be sent, see below for more details on payload handling. The `stream_id` is an integer that uniquely identifies the connection from this Fledge instance to the destination system. This id can be used if the plugin needs to have a unique identifier but in most cases can be ignored.

The `plugin_send` call returns three values, a boolean that indicates if any data has been sent, the object id of the last reading sent and the number of readings sent.

The code below is the `plugin_send` entry point for the http north plugin.

```
async def plugin_send(handle, payload, stream_id):
    """ Used to send the readings block from north to the configured destination.

    Args:
        handle - An object which is returned by plugin_init
        payload - A List of readings block
        stream_id - An Integer that uniquely identifies the connection from Fledge_
        ↪instance to the destination system
    Returns:
        Tuple which consists of
        - A Boolean that indicates if any data has been sent
        - The object id of the last reading which has been sent
        - Total number of readings which has been sent to the configured destination
    """
    try:
        is_data_sent, new_last_object_id, num_sent = await http_north.send_
        ↪payloads(payload)
    except asyncio.CancelledError:
        pass
    else:
        return is_data_sent, new_last_object_id, num_sent
```

## The `plugin_shutdown` call

The `plugin_shutdown` call is the final entry that is required for Python north plugin, it is called by the north service or task just prior to the task terminating or in a north service if the configuration is allowed, see reconfiguration below. The `plugin_shutdown` call is passed the plugin handle and should perform any cleanup required by the plugin.

```
def plugin_shutdown(handle):
    """ Used when plugin is no longer required and will be final call to shutdown the_
    ↪plugin. It should do any necessary cleanup if required.

    Args:
        handle - Plugin handle which is returned by plugin_init
```

(continues on next page)



(continued from previous page)

```
Returns:
"""
```

The call should not return any data. Once called the handle should no longer be regarded as valid and no further calls will be made to the plugin using this handle.

## Reconfiguration

Unlike other plugins within Fledge the north plugins do not have a reconfiguration entry point, this is due to the original nature of the north implementation in Fledge which used short lived tasks in order to send data out the north. Each new execution created a new task with new configuration, it was therefore felt that reconfiguration added a complexity to the north plugins that could be avoided.

Since the introduction of the feature that allows the north to be run as an always on service however this has become an issue. It is resolved by closing down the plugin, calling *plugin\_shutdown* and then restarting by calling *plugin\_init* to pass new configuration and retrieve a new plugin handle with that new configuration.

## Payload Handling

The payload that is passed to the *plugin\_send* routine is a Python list of readings, each reading is encoded as a Python DICT. The properties of the reading dict are;

Key	Description
id	The ID of the reading. Each reading is given an integer id that is an increasing value, it is these id values that are used to track how much data is sent via north plugin. One of the returns from the <i>plugin_send</i> routine is the id of the last reading that was successfully sent.
asset_code	The asset code of the reading. Typical a south service will generate reading for one or more asset codes. These asset codes are used to identify the source of the data. Multiple asset codes may appear in a single block of readings passed to the <i>plugin_send</i> routine.
reading	A nested Python DICT that stores the actual data points associated to the reading. These reading DICT's will contain a key/value pair for each data point within the asset. The value of this pair is the value of the data point and may be numeric, string, an array, or a nested object.
ts	The timestamp when the reading was first seen by the system.
user_ts	The timestamp of the data in the reading. This may be the same as <i>ts</i> above or in some cases may be a timestamp that has been received from the source of the data itself. This timestamp is the one that should be considered the most accurately represents the timestamp of the data.

A sample payload is shown below.

```
[{'reading': {'sinusoid': 0.0}, 'asset_code': 'sinusoid', 'id': 1, 'ts': '2021-09-27
↪06:55:52.692000+00:00', 'user_ts': '2021-09-27 06:55:49.947058+00:00'},
{'reading': {'sinusoid': 0.104528463}, 'asset_code': 'sinusoid', 'id': 2, 'ts': '2021-
↪09-27 06:55:52.692000+00:00', 'user_ts': '2021-09-27 06:55:50.947110+00:00'}]
```



## 13.9.2 C/C++ Plugins

The flow of a C/C++ plugin is very similar to that of a Python plugin, the entry points vary slightly compared to Python, mostly for language reasons.

### The `plugin_info` entry point

The `plugin_info` is again the first entry point that will be called, in the case a C/C++ plugin it will return a pointer to a `PLUGIN_INFORMATION` structure, this structure contains the same elements there are seen in the Python DICT that is returned by Python plugins.

```
static PLUGIN_INFORMATION info = {
    PLUGIN_NAME,           // Name
    VERSION,               // Version
    0,                     // Flags
    PLUGIN_TYPE_NORTH,     // Type
    "1.0.0",               // Interface version
    default_config          // Configuration
}
```

It should be noted that the `PLUGIN_INFORMATION` structure instance is declared as static. All global variables declared with a C/C++ plugin should be declared as static as the mechanism for loading the plugins will share global variables between plugins. Using true global variables can create unexpected interactions between plugins.

The items are

Name	Description
name	The name of the plugin.
version	The version of the plugin expressed as a string. This usually but not always matches the current version of Fledge.
flags	A bitmap of flags that give extra information about the plugin.
inter-face	The interface version, currently north plugins are at interface version 1.0.0.
config	The default configuration for the plugin. In C/C++ plugins this is returned as a string containing the JSON structure.

A number of flags are supported by the plugins, however a small subset are supported in north plugins, this subset consists of

Name	Description
SP_PERSIST_DATA	The plugin persists data and uses the data persistence API extensions.
SP_BUILTIN	The plugin is builtin with the Fledge core package. This should not be used for any user added plugins.

A typical implementation of the `plugin_info` entry would merely return the `PLUGIN_INFORMATION` structure for the plugin.

```
PLUGIN_INFORMATION *plugin_info()
{
    return &info;
}
```

More complex implementations may tailor the content of the information returned based upon some criteria determined at run time. An example of such a scenario might be to tailor the default configuration based upon some element of



discovery that occurs at run time. For example if the plugin is designed to send data to another service the *plugin\_info* entry point could perform some service discovery and update a set of options for an enumerated type in the default configuration. This would allow the user interface to give the user a selection list of all the service instances that it found when the plugin was run.

### The *plugin\_init* entry point

The *plugin\_init* entry point is called once the configuration of the plugin has been constructed by combining the default configuration with any stored configuration that the user has set for the plugin. The configuration is passed as a pointer to a C++ object of class *ConfigCategory*. This object may then be used to extract data from the configuration.

The *plugin\_init* call should be used to initialize the plugin itself and to extract the configuration for the *ConfigCategory* instance and store within the instance of the plugin. Details regarding the use of the *ConfigCategory* class can be found in the C++ Support Class section of the Plugin Developers Guide. Typically the north plugin will create an instance of a class that implements the functionality required, store the configuration in that class and return a pointer to that instance as the handle for the plugin. This will ensure that subsequent calls can access that class instance and the associated state, since all future calls will be passed the handle as an argument.

The following is perhaps the most generic form of the *plugin\_init* call.

```
PLUGIN_HANDLE plugin_init(ConfigCategory *configData)
{
    return (PLUGIN_HANDLE) (new myNorthPlugin(configData));
}
```

In this case it assumes we have a class, *myNorthPlugin* that implements the functionality of the plugin. The constructor takes the *ConfigCategory* pointer as an argument and performs all required initialization from that configuration category.

### The *plugin\_send* entry point

The *plugin\_send* entry point, as with Python plugins already describe, is the heart of a north plugin. It is called with the plugin handle and a block of readings data to be sent north. Typically the *plugin\_send* will extract the object created in the *plugin\_init* call from the handle and then call the functionality within that object to perform whatever translation and communication logic is required to send the reading data.

```
uint32_t plugin_send(PLUGIN_HANDLE handle, std::vector<Reading *>& readings)
{
    myNorthPlugin *plugin = (myNorthPlugin *)handle;
    return plugin->send(readings);
}
```

The block of readings is sent as a C++ standard template library vector of pointers to instance of the *Reading* class, also covered above in the section on C++ Support Classes.

The return from the *plugin\_send* function should be a count of the number of readings sent by the plugin.



### The `plugin_shutdown` entry point

The `plugin_shutdown` entry point is called when the plugin is no longer required. It should do any necessary cleanup required. As with other entry points, it is called with the handle that was returned by `plugin_init`. In the case of our simple plugin that might simply be to delete the C++ object that implements the plugin functionality.

```
uint32_t plugin_shutdown(PLUGIN_HANDLE handle)
{
    myNorthPlugin *plugin = (myNorthPlugin *)handle;
    delete plugin;
}
```

### The `plugin_register` entry point

The `plugin_register` entry point is used to pass two function pointers to the plugin. These functions pointers are the functions that should be called when either a set point write or a set point operation is required. The plugin should store these function pointers for later use.

```
void plugin_register(PLUGIN_HANDLE handle, (bool (*write)(char *name, char *value,
↳ControlDestination destination, ...), int (*operation)(char *operation, int,
↳paramCount, char *parameters[], ControlDestination destination, ...))
{
    myNorthPlugin *plugin = (myNorthPlugin *)handle;
    plugin->setpointCallbacks(write, operation);
}
```

This call will only be made if the plugin included the `SP_CONTROL` option in the flags field of the `PLUGIN_INFORMATION` structure.

## 13.9.3 Set Point Control

Fledge supports multiple paths for set point control, one of these paths allows for a north service to be bi-directional, with the north plugin receiving a trigger from the system north of Fledge to perform a set point control. This trigger may be the north plugin polling the system or a protocol response from the north.

Set point control is only available for north services, it is not supported for north tasks and will be ignored.

When the north plugin requires a set point write operation to be performed it calls the `write` callback that was passed to the plugin in the `plugin_register` entry point. This callback takes a number of arguments;

- The name of the set point to be written.
- The value to write to the set point. This is expressed as a string always.
- The destination of the write operation. This is passed using the `ControlDestination` enumerated type. Currently this may be one of
  - **DestinationBroadcast**: send the write operation to all south services that support control.
  - **DestinationAsset**: send the write request to the south service responsible for ingesting the given asset. The asset is passed as the next argument in the `write` call.
  - **DestinationService**: send the write request to the named south service.

For example if the north plugin wishes to write the set point called *speed* with the value 28 in the south service called *Motor Control* it would make a call as follows.



```
(*m_write)("speed", "28", DestinationService, "Motor Control");
```

Assuming the member variable *m\_write* was used to store the function pointer of the *write* callback.

If the north plugin requires an operation to be performed, rather than a write, then it should call the *operation* called which was passed to it in the *plugin\_register* call. This callback takes a set of arguments;

- The name of the operation to execute.
- The number of parameters the operation should be passed.
- An array of parameters, as strings, to pass to the operation
- The destination of the operation, this is the same set of destinations as per the write call.

## 13.10 Storage Plugins

Storage plugins are used to interact with the Storage Microservice and provide the persistent storage of information for Fledge.

The current version of Fledge comes with three storage plugins:

- The **SQLite plugin**: this is the default plugin and it is used for general purpose storage on constrained devices.
- The **SQLite In Memory plugin**: this plugin can be used in conjunction with one of the other storage plugins and will provide an in memory storage system for reading data only. Configuration data is stored using the *SQLite* or *PostgreSQL* plugins.
- The **PostgreSQL plugin**: this plugin can be set on request (or it can be built as a default plugin from source) and it is used for a more significant demand of storage on relatively larger systems.

### 13.10.1 Data and Metadata

Persistency is split in two blocks:

- **Metadata persistency**: it refers to the storage of metadata for Fledge, such as the configuration of the plugins, the scheduling of jobs and tasks and the the storage of statistical information.
- **Data persistency**: it refers to the storage of data collected from sensors and devices by the South microservices. The *SQLite In Memory* plugin is an example of a storage plugin designed to store only the data.

In the current implementation of Fledge, metadata and data use the same Storage plugin by default. Administrators can select different plugins for these two categories of data, with the most common configuration of this type to use the *SQLite In Memory* storage service for data and *SQLite* for the metadata. This is set by editing the storage configuration file. Currently there is no interface within Fledge to change the storage configuration.

The storage configuration file is stored in the Fledge data directory as *etc/storage.json*, the default storage configuration file is

```
{
  "plugin": {
    "value": "sqlite",
    "description": "The main storage plugin to load"
  },
  "readingPlugin": {
    "value": "",
    "description": "The storage plugin to load for readings data. If blank the main_
↪storage plugin is used."
  }
}
```

(continues on next page)



(continued from previous page)

```

},
"threads": {
  "value": "1",
  "description": "The number of threads to run"
},
"managedStatus": {
  "value": "false",
  "description": "Control if Fledge should manage the storage provider"
},
"port": {
  "value": "0",
  "description": "The port to listen on"
},
"managementPort": {
  "value": "0",
  "description": "The management port to listen on."
}
}

```

This sets the storage plugin to use as the *SQLite* plugin and leaves the *readingPlugin* blank. If the *readingPlugin* is blank then readings will be stored via the main plugin, if it is populated then a separate plugin will be used to store the readings. As an example, to store the readings in the *SQLite In Memory* plugin the storage.json file would be

```

{
  "plugin": {
    "value": "sqlite",
    "description": "The main storage plugin to load"
  },
  "readingPlugin": {
    "value": "sqlitememory",
    "description": "The storage plugin to load for readings data. If blank the main_
↪storage plugin is used."
  },
  "threads": {
    "value": "1",
    "description": "The number of threads to run"
  },
  "managedStatus": {
    "value": "false",
    "description": "Control if Fledge should manage the storage provider"
  },
  "port": {
    "value": "0",
    "description": "The port to listen on"
  },
  "managementPort": {
    "value": "0",
    "description": "The management port to listen on."
  }
}

```

Fledge must be restarted for changes to the storage.json file to take effect.

In addition to the definition of the plugins to use, the storage.json file also has a number of other configuration options for the storage service.

- **threads:** The number of threads to use to accept incoming REST requests. This is normally set to 1, increasing the number of threads has minimal impact on performance in normal circumstances.



- **managedStatus:** This configuration option allows Fledge to manage the underlying storage system. If, for example you used a database server and you wished Fledge to start and stop that server as part of the Fledge start up and shut down procedure you would set this option to “true”.
- **port:** This option can be used to make the storage service listen on a fixed port. This is normally not required, but can be used for diagnostic purposes.
- **managementPort:** As with *port* above this can be used for diagnostic purposes to fix the management API port for the storage service.

### 13.10.2 Common Elements for Storage Plugins

In designing the Storage API and plugins, we have first of all considered that there may be a large number of use cases for data and metadata persistence, therefore we have designed a flexible architecture that poses very few limitations. In practice, this means that developers can build their own Storage plugin and they can rely on anything they want to use as persistent storage. They can use a memory structure, or even a pass-through library, a file, a message queue system, a time series database, a relational database, NoSQL or something else.

After having praised the flexibility of the Storage plugins, let’s provide guidelines about the basic functionality they should provide, bearing in mind that such functionality may not be relevant for some use cases.

- **Metadata persistency:** As mentioned before, one of the main reasons to use a Storage plugin is to safely store the configuration of the Fledge components. Since the configuration must survive to a system crash or reboot, it is fair to say that such information should be stored in one or more files or in a database system.
- **Data buffering:** The second most important feature of a Storage plugin is the ability to buffer (or store) data coming from the outside world, typically from the South microservices. In some cases this feature may not be necessary, since administrators may want to send data to other systems as soon as possible, using a North task of microservice. Even in situations where data can be sent up North instantaneously, you should consider these scenarios:
  - Fledge may be installed in areas where the network is unreliable. The North plugins will provide the logic of retrying to gain connectivity and resending data when the connection has been lost in the middle of the transfer operations.
  - North services may rely on the use of networks that provide time windows to operate.
  - Historians and other systems may work better when data is transferred in blocks instead of a constant streaming.
- **Data purging:** Data may persist for the time needed by any specific use case, but it is pretty common that after a while (it can be seconds or minutes, but also day or months) data is no longer needed in Fledge. For this reason, the Storage plugin is able to purge data. Purging may be by time or by space usage, in conjunction with the fact that data may have been already transferred to other systems.
- **Data backup/restore:** Data, but especially metadata (i.e. configuration), can be backed up and stored safely on other systems. In case of crash and recovery, the same data may be restored into Fledge. Fledge provides a set of generic API to execute backup and restore operations.



## 13.11 Filter Plugins

Filter plugins provide a mechanism to alter the data stream as it flows through a fledge instance, filters may be applied in south or north micro-services and may form a pipeline of multiple processing elements through which the data flows. Filters applied in a south service will only process data that is received by the south service, whilst filters placed in the north will process all data that flows out of that north interface.

Filters may;

- augment data by adding static metadata or calculated values to the data
- remove data from the stream
- add data to the stream
- modify data in the stream

It should be noted that there are some alternatives to creating a filter if you wish to make simple changes to the data stream. There are a number of existing filters that provide a degree of programmability. These include the which allows an arbitrary mathematical formula to be applied to the data or the which allows a small include Python script to be applied to the data.

Filter plugins may be written in C++ or Python and have a very simple interface. The plugin mechanism and a subset of the API is common between all types of plugins including filters.

### 13.11.1 Configuration

Filters use the same configuration mechanism as the rest of Fledge, using a JSON document to describe the configuration parameters. As with any other plugin the structure is defined by the plugin and retrieve by the `plugin_info` entry point. This is then matched with the database content to pass the configured values to the `plugin_init` entry point.

### 13.11.2 C++ Filter Plugin API

The filter API consists of a small number of C function entry points, these are called in a strict order and based on the same set of common API entry points for all Fledge plugins.

#### Plugin Information

The *plugin\_info* entry point is the first entry point that is called in a filter plugin and returns the plugin information structure. This is the exact same call that every Fledge plugin must support and is used to determine the type of the plugin and the configuration category defaults for the plugin.

A typical implementation of *plugin\_info* would merely return a pointer to a static `PLUGIN_INFORMATION` structure.

```
PLUGIN_INFORMATION *plugin_info()
{
    return &info;
}
```



## Plugin Initialise

The *plugin\_init* entry point is called after *plugin\_info* has been called and before any data is passed to the filter. It is called at the phase where the service is setting up the filter pipeline and provides the filter with its configuration category that now contains the user supplied values and the destination to which the filter will send the output of the filter.

```
PLUGIN_HANDLE plugin_init(ConfigCategory* config,
                          OUTPUT_HANDLE *outHandle,
                          OUTPUT_STREAM output)
{
}
```

The *config* parameter is the configuration category with the user supplied values inserted, the *outHandle* is a handle for the next filter in the chain and the *output* is a function pointer to call to send the data to the next filter in the chain. The *outHandle* and *output* arguments should be stored for future use in the *plugin\_ingest* when data is to be forwarded within the pipeline.

The *plugin\_init* function returns a handle that will be passed to all subsequent plugin calls. This handle can be used to store state that needs to be passed between calls. Typically the *plugin\_init* call will create a C++ class that implements the filter and return a point to the instance as the handle. The instance can then be used to store the state of the filter, including the output handle and callback that needs to be used.

Filter classes can also be used to buffer data between calls to the *plugin\_ingest* entry point, allowing a filter to defer the processing of the data until it has a sufficient quantity of buffered data available to it.

## Plugin Ingest

The *plugin\_ingest* entry point is the workhorse of the filter, it is called with sets of readings to process and then passes on the new set of readings to the next filter in the pipeline. The process of passing on the data to the next filter is via the *OUTPUT\_STREAM* function pointer. A filter does not have to output data each time it ingests data, it is free to output no data or to output more or less data than it was called with.

```
void plugin_ingest(PLUGIN_HANDLE *handle,
                  READINGSET *readingSet)
{
}
```

The number of readings that a filter is called with will depend on the environment it is run in and what any filters earlier in the filter pipeline have produced. A filter that requires a particular sample size in order to process a result should therefore be prepared to buffer data across multiple calls to *plugin\_ingest*. Several examples of filters that do this are available for reference.

The *plugin\_ingest* call may send data onwards in the filter pipeline by using the stored *output* and *outHandle* parameters passed to *plugin\_init*.

```
(*output)(outHandle, readings);
```



## Plugin Reconfigure

As with other plugin types the filter may be reconfigured during its lifetime. When a reconfiguration operation occurs the *plugin\_reconfigure* method will be called with the new configuration for the filter.

```
void plugin_reconfigure(PLUGIN_HANDLE *handle, const std::string& newConfig)
{
}
```

## Plugin Shutdown

As with other plugins a shutdown call exists which may be used by the plugin to perform any cleanup that is required when the filter is shut down.

```
void plugin_shutdown(PLUGIN_HANDLE *handle)
{
}
```

## C++ Helper Class

It is expected that filters will be written as C++ classes, with the plugin handle being used as a mechanism to store and pass the pointer to the instance of the filter class. In order to make it easier to write filters a base *FledgeFilter* class has been provided, it is recommended that you derive your specific filter class from this base class in order to simplify the implementation

```
class FledgeFilter {
public:
    FledgeFilter(const std::string& filterName,
                ConfigCategory& filterConfig,
                OUTPUT_HANDLE *outHandle,
                OUTPUT_STREAM output);
    ~FledgeFilter() {};
    const std::string&
        getName() const { return m_name; };
    bool
        isEnabled() const { return m_enabled; };
    ConfigCategory&
        getConfig() { return m_config; };
    void
        disableFilter() { m_enabled = false; };
    void
        setConfig(const std::string& newConfig);
public:
    OUTPUT_HANDLE*
        m_data;
    OUTPUT_STREAM
        m_func;
protected:
    std::string
        m_name;
    ConfigCategory
        m_config;
    bool
        m_enabled;
};
```



### 13.11.3 C++ Filter Example

The following example is a simple data processing example. It applies the `log()` function to numeric data in the data stream

#### Plugin Interface

Most plugins written in C++ have a source file that encapsulates the C API to the plugin, this is traditionally called `plugin.cpp`. The example plugin follows this model with the content of `plugin.cpp` shown below.

The first section includes the filter class that is the actual implementation of the filter logic and defines the JSON configuration category. This uses the *QUOTE* macro in order to make the JSON definition more readable.

```
/*
 * Fledge "log" filter plugin.
 *
 * Copyright (c) 2020 Dianomic Systems
 *
 * Released under the Apache 2.0 Licence
 *
 * Author: Mark Riddoch
 */

#include <logFilter.h>
#include <version.h>

#define FILTER_NAME "log"
const static char *default_config = QUOTE({
    "plugin" : {
        "description" : "Log filter plugin",
        "type" : "string",
        "default" : FILTER_NAME,
        "readonly": "true"
    },
    "enable": {
        "description": "A switch that can be used to enable or_
↳disable execution of the log filter.",
        "type": "boolean",
        "displayName": "Enabled",
        "default": "false"
    },
    "match" : {
        "description" : "An optional regular expression to match in_
↳the asset name.",
        "type": "string",
        "default": "",
        "order": "1",
        "displayName": "Asset filter"}
});

using namespace std;
```

We then define the plugin information contents that will be returned by the `plugin_info` call.

```
/**
 * The Filter plugin interface
 */
```

(continues on next page)



(continued from previous page)

```
extern "C" {

/**
 * The plugin information structure
 */
static PLUGIN_INFORMATION info = {
    FILTER_NAME,           // Name
    VERSION,               // Version
    0,                     // Flags
    PLUGIN_TYPE_FILTER,    // Type
    "1.0.0",               // Interface version
    default_config         // Default plugin configuration
};
};
```

The final section of this file consists of the entry points themselves and the implementation. The majority of this consist of calls to the LogFilter class that in this case implements the logic of the filter.

```
/**
 * Return the information about this plugin
 */
PLUGIN_INFORMATION *plugin_info()
{
    return &info;
}

/**
 * Initialise the plugin, called to get the plugin handle.
 * We merely create an instance of our LogFilter class
 *
 * @param config      The configuration category for the filter
 * @param outHandle   A handle that will be passed to the output stream
 * @param output      The output stream (function pointer) to which data is passed
 * @return            An opaque handle that is used in all subsequent calls to the
 * → plugin
 */
PLUGIN_HANDLE plugin_init(ConfigCategory* config,
                          OUTPUT_HANDLE *outHandle,
                          OUTPUT_STREAM output)
{
    LogFilter *log = new LogFilter(FILTER_NAME,
                                   *config,
                                   outHandle,
                                   output);

    return (PLUGIN_HANDLE)log;
}

/**
 * Ingest a set of readings into the plugin for processing
 *
 * @param handle      The plugin handle returned from plugin_init
 * @param readingSet  The readings to process
 */
void plugin_ingest(PLUGIN_HANDLE *handle,
                  READINGSET *readingSet)
{

```

(continues on next page)



(continued from previous page)

```

        LogFilter *log = (LogFilter *) handle;
        log->ingest(readingSet);
    }

    /**
     * Plugin reconfiguration method
     *
     * @param handle      The plugin handle
     * @param newConfig   The updated configuration
     */
    void plugin_reconfigure(PLUGIN_HANDLE *handle, const std::string& newConfig)
    {
        LogFilter *log = (LogFilter *)handle;
        log->reconfigure(newConfig);
    }

    /**
     * Call the shutdown method in the plugin
     */
    void plugin_shutdown(PLUGIN_HANDLE *handle)
    {
        LogFilter *log = (LogFilter *) handle;
        delete log;
    }

    // End of extern "C"
};

```

## Filter Class

Although it is not mandatory it is good practice to encapsulate the filter logic in a class, these classes are derived from the FledgeFilter class

```

#ifndef _LOG_FILTER_H
#define _LOG_FILTER_H
/*
 * Fledge "Log" filter plugin.
 *
 * Copyright (c) 2020 Dianomic Systems
 *
 * Released under the Apache 2.0 Licence
 *
 * Author: Mark Riddoch
 */
#include <filter.h>
#include <reading_set.h>
#include <config_category.h>
#include <string>
#include <logger.h>
#include <mutex>
#include <regex>
#include <math.h>

/**

```

(continues on next page)



(continued from previous page)

```

    * Convert the incoming data to use a logarithmic scale
    */
class LogFilter : public FledgeFilter {
public:
    LogFilter(const std::string& filterName,
              ConfigCategory& filterConfig,
              OUTPUT_HANDLE *outHandle,
              OUTPUT_STREAM output);
    ~LogFilter();
    void ingest(READINGSET *readingSet);
    void reconfigure(const std::string& newConfig);
private:
    void handleConfig(ConfigCategory& config);
    std::string m_match;
    std::regex *m_regex;
    std::mutex m_configMutex;
};

#endif

```

## Filter Class Implementation

The following is the code that implements the filter logic

```

/*
 * Fledge "Log" filter plugin.
 *
 * Copyright (c) 2020 Dianomic Systems
 *
 * Released under the Apache 2.0 Licence
 *
 * Author: Mark Riddoch
 */
#include <logFilter.h>

using namespace std;

/**
 * Constructor for the LogFilter.
 *
 * We call the constructor of the base class and handle the initial
 * configuration of the filter.
 *
 * @param filterName The name of the filter
 * @param filterConfig The configuration category for this filter
 * @param outHandle The handle of the next filter in the chain
 * @param output A function pointer to call to output data to the next_
 * filter
 */
LogFilter::LogFilter(const std::string& filterName,
                     ConfigCategory& filterConfig,
                     OUTPUT_HANDLE *outHandle,
                     OUTPUT_STREAM output) : m_regex(NULL),
                                             FledgeFilter(filterName, filterConfig, outHandle,
->output)

```

(continues on next page)



(continued from previous page)

```

{
    handleConfig(filterConfig);
}

/**
 * Destructor for this filter class
 */
LogFilter::~LogFilter()
{
    if (m_regex)
        delete m_regex;
}

/**
 * The actual filtering code
 *
 * @param readingSet The reading data to filter
 */
void
LogFilter::ingest(READINGSET *readingSet)
{
    lock_guard<mutex> guard(m_configMutex);

    if (isEnabled()) // Filter enable, process the readings
    {
        const vector<Reading *>& readings = ((ReadingSet *)readingSet)->
↳getAllReadings();
        for (vector<Reading *>::const_iterator elem = readings.begin();
            elem != readings.end(); ++elem)
        {
            // If we set a matching regex then compare to the name of
↳this asset

            if (!m_match.empty())
            {
                string asset = (*elem)->getAssetName();
                if (!regex_match(asset, *m_regex))
                {
                    continue;
                }
            }

            // We are modifying this asset so put an entry in the asset
↳tracker

            AssetTracker::getAssetTracker()->
↳addAssetTrackingTuple(getName(), (*elem)->getAssetName(), string("Filter"));

            // Get a reading DataPoints
            const vector<Datapoint *>& dataPoints = (*elem)->
↳getReadingData();

            // Iterate over the datapoints
            for (vector<Datapoint *>::const_iterator it = dataPoints.
↳begin(); it != dataPoints.end(); ++it)
            {
                // Get the reference to a DataPointValue
                DatapointValue& value = (*it)->getData();
            }
        }
    }
}

```

(continues on next page)



(continued from previous page)

```

        /*
         * Deal with the T_INTEGER and T_FLOAT types.
         * Try to preserve the type if possible but
         * if a floating point log function is applied
         * then T_INTEGER values will turn into T_FLOAT.
         * If the value is zero we do not apply the log_
↪function
        */
        if (value.getType() == DatapointValue::T_INTEGER)
        {
            long ival = value.toInt();
            if (ival != 0)
            {
                double newValue = log((double)ival);
                value.setValue(newValue);
            }
        }
        else if (value.getType() == DatapointValue::T_FLOAT)
        {
            double dval = value.toDouble();
            if (dval != 0.0)
            {
                value.setValue(log(dval));
            }
        }
        else
        {
            // do nothing for other types
        }
    }
}

// Pass on all readings in this case
(*m_func)(m_data, readingSet);
}

/**
 * Reconfiguration entry point to the filter.
 *
 * This method runs holding the configMutex to prevent
 * ingest using the regex class that may be destroyed by this
 * call.
 *
 * Pass the configuration to the base FilterPlugin class and
 * then call the private method to handle the filter specific
 * configuration.
 *
 * @param newConfig The JSON of the new configuration
 */
void
LogFilter::reconfigure(const std::string& newConfig)
{
    lock_guard<mutex> guard(m_configMutex);
    setConfig(newConfig);
    // Pass the configuration to the base_
↪class
    handleConfig(m_config);

```

(continues on next page)



(continued from previous page)

```

}

/**
 * Handle the filter specific configuration. In this case
 * it is just the single item "match" that is a regex
 * expression
 *
 * @param config      The configuration category
 */
void
LogFilter::handleConfig(ConfigCategory& config)
{
    if (config.itemExists("match"))
    {
        m_match = config.getValue("match");
        if (m_regex)
            delete m_regex;
        m_regex = new regex(m_match);
    }
}

```

### 13.11.4 Python Filter API

Filters may also be written in Python, the API is very similar to that of a C++ filter and consists of the same set of entry points.

#### Plugin Information

As with C++ filters this is the first entry point called, it returns a Python dictionary that describes the filter.

```

def plugin_info():
    """ Returns information about the plugin
    Args:
    Returns:
        dict: plugin information
    Raises:
    """

```

#### Plugin Initialisation

The *plugin\_init* call is used to pass the resolved configuration to the plugin and also pass in the handle of the next filter in the pipeline and a callback that should be called with the output data of the filter.

```

def plugin_init(config, ingest_ref, callback):
    """ Initialise the plugin
    Args:
        config: JSON configuration document for the Filter plugin configuration_
        ↪category
        ingest_ref: filter ingest reference
        callback: filter callback
    Returns:
        data: JSON object to be used in future calls to the plugin

```

(continues on next page)



(continued from previous page)

```
Raises:
"""
```

## Plugin Ingestion

The *plugin\_ingest* method is used to pass data into the plugin, the plugin will then process that data and call the callback that was passed into the *plugin\_init* entry point with the *ingest\_ref* handle and the data to send along the filter pipeline.

```
def plugin_ingest(handle, data):
    """ Modify readings data and pass it onward

    Args:
        handle: handle returned by the plugin initialisation call
        data: readings data
    """
```

The *data* is arranged as an array of Python dictionaries, each of which is a *Reading*. Typically the data can be processed by traversing the array

```
for elem in data:
    process(elem)
```

## Plugin Reconfigure

The *plugin\_reconfigure* entry point is called whenever a configuration change occurs for the filters configuration category.

```
def plugin_reconfigure(handle, new_config):
    """ Reconfigures the plugin

    Args:
        handle: handle returned by the plugin initialisation call
        new_config: JSON object representing the new configuration category for the
        ↪category
    Returns:
        new_handle: new handle to be used in the future calls
    """
```

## Plugin Shutdown

Called when the plugin is to be shutdown to allow it to perform any cleanup operations.

```
def plugin_shutdown(handle):
    """ Shutdowns the plugin doing required cleanup.

    Args:
        handle: handle returned by the plugin initialisation call
    Returns:
        plugin shutdown
    """
```



### 13.11.5 Python Filter Example

The following is an example of a Python filter that calculates an exponential moving average.

```
# -*- coding: utf-8 -*-

# Fledge_BEGIN
# See: http://fledge-iot.readthedocs.io/
# Fledge_END

""" Module for EMA filter plugin

Generate Exponential Moving Average
The rate value (x) allows to include x% of current value
and (100-x)% of history
A datapoint called 'ema' is added to each reading being filtered
"""

import time
import copy
import logging

from fledge.common import logger
import filter_ingest

__author__ = "Massimiliano Pinto"
__copyright__ = "Copyright (c) 2022 Dianomic Systems Inc."
__license__ = "Apache 2.0"
__version__ = "${VERSION}"

_LOGGER = logger.setup(__name__, level = logging.INFO)

PLUGIN_NAME = 'ema'

_DEFAULT_CONFIG = {
    'plugin': {
        'description': 'Exponential Moving Average filter plugin',
        'type': 'string',
        'default': PLUGIN_NAME,
        'readonly': 'true'
    },
    'enable': {
        'description': 'Enable ema plugin',
        'type': 'boolean',
        'default': 'false',
        'displayName': 'Enabled',
        'order': "3"
    },
    'rate': {
        'description': 'Rate value: include % of current value',
        'type': 'float',
        'default': '0.07',
        'displayName': 'Rate',
        'order': "2"
    },
    'datapoint': {
        'description': 'Datapoint name for calculated ema value',
        'type': 'string',

```

(continues on next page)



(continued from previous page)

```

        'default': PLUGIN_NAME,
        'displayName': 'EMA datapoint',
        'order': "1"
    }
}

def compute_ema(handle, reading):
    """ Compute EMA

    Args:
        A reading data
    """
    rate = float(handle['rate']['value'])
    for attribute in list(reading):
        if not handle['latest']:
            handle['latest'] = reading[attribute]
        handle['latest'] = reading[attribute] * rate + handle['latest'] * (1 - rate)
    reading[handle['datapoint']['value']] = handle['latest']

def plugin_info():
    """ Returns information about the plugin

    Args:
    Returns:
        dict: plugin information
    Raises:
    """
    return {
        'name': PLUGIN_NAME,
        'version': '1.9.2',
        'mode': 'none',
        'type': 'filter',
        'interface': '1.0',
        'config': _DEFAULT_CONFIG
    }

def plugin_init(config, ingest_ref, callback):
    """ Initialise the plugin

    Args:
        config: JSON configuration document for the Filter plugin configuration_
        ↪category
        ingest_ref: filter ingest reference
        callback: filter callback
    Returns:
        data: JSON object to be used in future calls to the plugin
    Raises:
    """
    _config = copy.deepcopy(config)
    _config['ingestRef'] = ingest_ref
    _config['callback'] = callback
    _config['latest'] = None
    _config['shutdownInProgress'] = False
    return _config

```

(continues on next page)



(continued from previous page)

```

def plugin_reconfigure(handle, new_config):
    """ Reconfigures the plugin

    Args:
        handle: handle returned by the plugin initialisation call
        new_config: JSON object representing the new configuration category for the
    category
    Returns:
        new_handle: new handle to be used in the future calls
    """
    _LOGGER.info("Old config for ema plugin {} \n new config {}".format(handle, new_
    config))

    new_handle = copy.deepcopy(new_config)
    new_handle['shutdownInProgress'] = False
    new_handle['latest'] = None
    new_handle['ingestRef'] = handle['ingestRef']
    new_handle['callback'] = handle['callback']
    return new_handle

def plugin_shutdown(handle):
    """ Shutdowns the plugin doing required cleanup.

    Args:
        handle: handle returned by the plugin initialisation call
    Returns:
        plugin shutdown
    """
    handle['shutdownInProgress'] = True
    time.sleep(1)
    handle['callback'] = None
    handle['ingestRef'] = None
    handle['latest'] = None

    _LOGGER.info('{} filter plugin shutdown.'.format(PLUGIN_NAME))

def plugin_ingest(handle, data):
    """ Modify readings data and pass it onward

    Args:
        handle: handle returned by the plugin initialisation call
        data: readings data
    """
    if handle['shutdownInProgress']:
        return

    if handle['enable']['value'] == 'false':
        # Filter not enabled, just pass data onwards
        filter_ingest.filter_ingest_callback(handle['callback'], handle['ingestRef'],
    data)
        return

    # Filter is enabled: compute EMA for each reading
    for elem in data:
        compute_ema(handle, elem['readings'])

```

(continues on next page)



(continued from previous page)

```
# Pass data onwards
filter_ingest.filter_ingest_callback(handle['callback'], handle['ingestRef'],
↳data)

_LOGGER.debug("{} filter_ingest done.".format(PLUGIN_NAME))
```

## 13.12 Notification Delivery Plugins

Notification delivery plugins are used by the notification system to send a notification to some other system or device. They are the transport that allows the event to be notified to that other system or device.

Notification delivery plugins may be written in C or C++ and have a very simple interface. The plugin mechanism and a subset of the API is common between all types of plugins including filters. This documentation is based on the . The sends MQTT messages to a configurable MQTT topic when a notification is triggered and cleared.

### 13.12.1 Configuration

Notification Delivery plugins use the same configuration mechanism as the rest of Fledge, using a JSON document to describe the configuration parameters. As with any other plugin the structure is defined by the plugin and retrieve by the *plugin\_info* entry point. This is then matched with the database content to pass the configured values to the *plugin\_init* entry point.

### 13.12.2 Notification Delivery Plugin API

The notification delivery plugin API consists of a small number of C function entry points, these are called in a strict order and based on the same set of common API entry points for all Fledge plugins.

#### Plugin Information

The *plugin\_info* entry point is the first entry point that is called in a notification delivery plugin and returns the plugin information structure. This is the exact same call that every Fledge plugin must support and is used to determine the type of the plugin and the configuration category defaults for the plugin.

A typical implementation of *plugin\_info* would merely return a pointer to a static PLUGIN\_INFORMATION structure.

```
PLUGIN_INFORMATION *plugin_info()
{
    return &info;
}
```



## Plugin Initialise

The second call that is made to the plugin is the *plugin\_init* call, that is used to retrieve a handle on the plugin instance and to configure the plugin.

```
PLUGIN_HANDLE plugin_init(ConfigCategory* config)
{
    MQTT *mqtt = new MQTT(config);
    return (PLUGIN_HANDLE)mqtt;
}
```

The *config* parameter is the configuration category with the user supplied values inserted, these values are used to configure the behavior of the plugin. In the case of our MQTT example we use this to call the constructor of our MQTT class.

```
/**
 * Construct a MQTT notification plugin
 *
 * @param category    The configuration of the plugin
 */
MQTT::MQTT(ConfigCategory *category)
{
    if (category->itemExists("broker"))
        m_broker = category->getValue("broker");
    if (category->itemExists("topic"))
        m_topic = category->getValue("topic");
    if (category->itemExists("trigger_payload"))
        m_trigger = category->getValue("trigger_payload");
    if (category->itemExists("clear_payload"))
        m_clear = category->getValue("clear_payload");
}
```

This constructor merely stores values out of the configuration category as private member variables of the MQTT class.

We return the pointer to our MQTT class as the handle for the plugin. This allows subsequent calls to the plugin to reference the instance created by the *plugin\_init* call.

## Plugin Delivery

This is the API call made whenever the plugin needs to send a triggered or cleared notification state. It may be called multiple times within the lifetime of a plugin.

```
bool plugin_deliver(PLUGIN_HANDLE handle,
                    const std::string& deliveryName,
                    const std::string& notificationName,
                    const std::string& triggerReason,
                    const std::string& message)
{
    MQTT *mqtt = (MQTT *)handle;
    return mqtt->notify(notificationName, triggerReason, message);
}
```

The delivery call is passed the handle, which gives us the MQTT class instance on this case, the name of the notification, a trigger reason, which is a JSON document and a message. The trigger reason JSON document contains information about why the delivery call was made, including the triggered or cleared status, the timestamp of the



reading that caused the notification to trigger and the name of the asset or assets involved in the notification rule that triggered this delivery event.

```
{
  "reason": "triggered",
  "asset": ["sinusoid"],
  "timestamp": "2020-11-18 11:52:33.960530+00:00"
}
```

The return from the *plugin\_deliver* entry point is a boolean that indicates if the delivery succeeded or not.

In the case of our MQTT example we call the notify method of the class, this then interacts with the MQTT broker.

```
/**
 * Send a notification via MQTT broker
 *
 * @param notificationName The name of this notification
 * @param triggerReason    Why the notification is being sent
 * @param message          The message to send
 */
bool MQTT::notify(const string& notificationName, const string& triggerReason, const_
↳string& message)
{
    string          payload = m_trigger;
    MQTTClient      client;

    lock_guard<mutex> guard(m_mutex);

    // Parse the JSON that represents the reason data
    Document doc;
    doc.Parse(triggerReason.c_str());
    if (!doc.HasParseError() && doc.HasMember("reason"))
    {
        if (!strcmp(doc["reason"].GetString(), "cleared"))
            payload = m_clear;
    }

    // Connect to the MQTT broker
    MQTTClient_connectOptions conn_opts = MQTTClient_connectOptions_initializer;
    MQTTClient_message pubmsg = MQTTClient_message_initializer;
    MQTTClient_deliveryToken token;
    int rc;

    if ((rc = MQTTClient_create(&client, m_broker.c_str(), CLIENTID,
MQTTCLIENT_PERSISTENCE_NONE, NULL)) != MQTTCLIENT_SUCCESS)
    {
        Logger::getLogger()->error("Failed to create client, return code %d\n
↳", rc);
        return false;
    }

    conn_opts.keepAliveInterval = 20;
    conn_opts.cleansession = 1;
    if ((rc = MQTTClient_connect(client, &conn_opts)) != MQTTCLIENT_SUCCESS)
    {
        Logger::getLogger()->error("Failed to connect, return code %d\n", rc);
        return false;
    }
}
```

(continues on next page)



(continued from previous page)

```

    // Construct the payload
    pubmsg.payload = (void *)payload.c_str();
    pubmsg.payloadlen = payload.length();
    pubmsg.qos = 1;
    pubmsg.retained = 0;

    // Publish the message
    if ((rc = MQTTClient_publishMessage(client, m_topic.c_str(), &pubmsg, &
    token)) != MQTTCLIENT_SUCCESS)
    {
        Logger::getLogger()->error("Failed to publish message, return code %d\
    ", rc);
        return false;
    }

    // Wait for completion and disconnect
    rc = MQTTClient_waitForCompletion(client, token, TIMEOUT);
    if ((rc = MQTTClient_disconnect(client, 10000)) != MQTTCLIENT_SUCCESS)
        Logger::getLogger()->error("Failed to disconnect, return code %d\n",
    rc);
    MQTTClient_destroy(&client);
    return true;
}

```

## Plugin Reconfigure

As with other plugin types the notification delivery plugin may be reconfigured during its lifetime. When a reconfiguration operation occurs the *plugin\_reconfigure* method will be called with the new configuration for the plugin.

```

void plugin_reconfigure(PUGIN_HANDLE *handle, const std::string& newConfig)
{
    MQTT *mqtt = (MQTT *)handle;
    mqtt->reconfigure(newConfig);
    return;
}

```

In the case of our MQTT example we call the reconfigure method of our MQTT class. In this method the new values are copied into the local member variables of the instance.

```

/**
 * Reconfigure the MQTT delivery plugin
 *
 * @param newConfig The new configuration
 */
void MQTT::reconfigure(const string& newConfig)
{
    ConfigCategory category("new", newConfig);
    lock_guard<mutex> guard(m_mutex);
    m_broker = category.getValue("broker");
    m_topic = category.getValue("topic");
    m_trigger = category.getValue("trigger_payload");
    m_clear = category.getValue("clear_payload");
}

```



The mutex is used here to prevent the plugin reconfiguration occurring when we are delivering a notification. The same mutex is held in the notify method of the MQTT class.

## Plugin Shutdown

As with other plugins a shutdown call exists which may be used by the plugin to perform any cleanup that is required when the plugin is shut down.

```
void plugin_shutdown(PLUGIN_HANDLE *handle)
{
    MQTT *mqtt = (MQTT *)handle;
    delete mqtt;
}
```

In the case of our MQTT example we merely destroy the instance of the MQTT class and allow the destructor of that class to do any cleanup that is required. In the case of this example there is no cleanup required.

## 13.13 Plugin Packaging

There are a set of files that must exist within the repository of a plugin that are used to create the package for that plugin on the various supported platforms. The following documents what those files are and what they should contain.

### 13.13.1 Common files

- **Description** - It should contain a brief description of the plugin and will be used as the description for the package that is created. Also make sure description of plugin must be in a single line as of now we do not have support multi lines yet.
- **Package** - This is the main file where we define set of variables.
  - **plugin\_name** - Name of the Plugin.
  - **plugin\_type** - Type of the Plugin.
  - **plugin\_install\_dirname** - Installed Directory name.
  - **plugin\_package\_name (Optional)** - Name of the Package. If it is not given then the package name should be same as plugin name.
  - **requirements** - Runtime Architecture specific packages list and should have comma separated values without any space.

---

**Note:** For C-based plugins if a plugin requires some additional libraries to install with then set `additional_libs` variable inside Package file. And the value must be with following contract:

`additional_libs="DIRECTORY_NAME:FILE_NAME"` - in case of single additional  
`additional_libs="DIRECTORY_NAME:FILE_NAME1,DIRECTORY_NAME:FILE_NAME2"` - in case of multiple use comma separated with both directory & file name

---

- **service\_notification.version** - It is only required if the plugin is a notification rule or notification delivery plugin. It contains the minimum version of the notification service which the plugin requires.



### 13.13.2 C based Plugins

- **VERSION** - It contains the version number of the plugin and is used by the build process to include the version number within the code and also within the name of the package file created.
- **fledge.version** - It contains the minimum version number of Fledge required by the plugin.
- **requirements.sh (Optional)** - It is used to install any additional libraries or other artifacts that are need to build the plugin. It takes the form of a shell script. This script, if it exists, will be run as a part of the process of building the plugin before the cmake command is issued in the build process.
- **extras\_install.sh (Optional)** - It is a shell script that is added to the package to allow for extra commands to be executed as part of the package installation. Not all plugins will require this file to be present and it can be omitted if there are no extra steps required on the installation.

#### Examples of filename along with content

##### 1. VERSION

```
$ cat VERSION
1.9.2
```

##### 2. fledge.version

```
$ cat fledge.version
fledge_version>=1.9
```

##### 3. requirements.sh

```
$ cat requirements.sh
#!/usr/bin/env bash
which apt >/dev/null 2>&1
if [ $? -eq 0 ]; then
    sudo apt install -y libmodbus-dev
else
    which yum >/dev/null 2>&1
    if [ $? -eq 0 ]; then
        sudo yum -y install epel-release libmodbus libmodbus-devel
    fi
fi
```

##### 4. Description

```
$ cat Description
Fledge modbus plugin. Supports modbus RTU and modbus TCP.
```

##### 5. Package

```
$ cat Package
# A set of variables that define how we package this repository
#
plugin_name=modbus
plugin_type=south
plugin_install_dirname=ModbusC
plugin_package_name=fledge-south-modbus
additional_libs="usr/local/lib:usr/local/lib/libsmod.so*"
```

(continues on next page)



(continued from previous page)

```
# Now build up the runtime requirements list. This has 3 components
# 1. Generic packages we depend on in all architectures and package managers
# 2. Architecture specific packages we depend on
# 3. Package manager specific packages we depend on
requirements="fledge"

case "$arch" in
    x84_64)
        ;;
    armv7l)
        ;;
    aarch64)
        ;;
esac
case "$package_manager" in
    deb)
        requirements="${requirements}, libmodbus-dev"
        ;;
esac
```

**Note:** If your package is not supported for a specific platform then you must exit with exitcode 1.

#### 6. service\_notification.version

```
$ cat service_notification.version
service_notification_version>=1.9.2
```

## Common Additional Libraries Package

Below are the packages which created a part of the process of building Fledge that are commonly used in plugins.

- **fledge-mqtt** which is a packaged version of the libpaho-mqtt library.
- **fledge-gcp** which is a packaged version of the libjwt and libjansson libraries.
- **fledge-iec** which is a packaged version of the IEC 60870 and IEC 61850 libraries.
- **fledge-s2opcua** which is a packaged version of libexpat and libs2opcua libraries.

If your plugin depends on any of these libraries they should be added to the *requirements* variable in the **Package** file rather than adding them as *additional\_libs* since the version of these is managed by the Fledge build and packaging process. Below is the example

```
requirements="fledge, fledge-s2opcua"
```



### 13.13.3 Python based Plugins

- **VERSION.{PLUGIN\_TYPE}.{PLUGIN\_NAME}** - It contains the packaged version of the plugin and also the minimum fledge version that the plugin requires.
- **install\_notes.txt (Optional)** - It is a simple text file that can be included if there are specific instructions required to be given during the installation of the plugin. These notes will be displayed at the end of the installation process for the package.
- **extras\_install.sh (Optional)** - It is a shell script that is added to the package to allow for extra commands to be executed as part of the package installation. Not all plugins will require this file to be present and it can be omitted if there are no extra steps required on the installation.
- **requirements-{PLUGIN\_NAME}.txt (Optional)** - It is a simple text file that can be included if there are pip dependencies required to be given during the installation of the plugin. Also make sure file should be placed inside *python* directory.

#### Examples of filename along with content

##### 1. Description

```
$ cat Description
Fledge South Sinusoid plugin
```

##### 2. Package

```
$ cat Package
# A set of variables that define how we package this repository
#
plugin_name=sinusoid
plugin_type=south
plugin_install_dirname=sinusoid

# Now build up the runtime requirements list. This has 3 components
# 1. Generic packages we depend on in all architectures and package managers
# 2. Architecture specific packages we depend on
# 3. Package manager specific packages we depend on
requirements="fledge"

case "$arch" in
    x86_64)
        ;;
    armv7l)
        ;;
    aarch64)
        ;;
esac
case "$package_manager" in
    deb)
        ;;
esac
```

---

**Note:** If your package is not supported for a specific platform then you must exit with exitcode 1.

---

##### 3. VERSION.{PLUGIN\_TYPE}.{PLUGIN\_NAME}



```
$ cat VERSION.south.sinusoid
fledge_south_sinusoid_version=1.9.2
fledge_version>=1.9
```

#### 4. install\_notes.txt

```
$ cat install_notes.txt
It is required to reboot the RPi, please do the following steps:
1) sudo reboot
```

#### 5. extras\_install.sh

```
#!/usr/bin/env bash

os_name=$(grep -o '^NAME=.*' /etc/os-release | cut -f2 -d\" | sed 's/"/g')
os_version=$(grep -o '^VERSION_ID=.*' /etc/os-release | cut -f2 -d\" | sed 's/"/g')
echo "Platform is ${os_name}, Version: ${os_version}"
arch=`arch`
ID=$(cat /etc/os-release | grep -w ID | cut -f2 -d"=)
if [ ${ID} != "mendel" ]; then
case $os_name in
  *Ubuntu*)
    if [ ${arch} = "aarch64" ]; then
      python3 -m pip install --upgrade pip
    fi
    ;;
  esac
fi
```

#### 6. requirements-{PLUGIN\_NAME}.txt

```
$ cat python/requirements-modbuscp.txt
pymodbus3==1.0.0
```

## 13.13.4 Building A Package

Firstly you need to clone the repository [fledge-pkg](#). Now do the following steps

```
$ cd plugins
$ ./make_deb -b <BRANCH_NAME> <REPOSITORY_NAME>

if everything goes well with above command then you can find your package inside
↳ archive directory.

$ ls archive
```



## 13.14 Testing Your Plugin

The first step in testing your new plugin is to put the plugin in the location in which your Fledge system will be loading it from. The exact location depends on the way you installed your Fledge system and the type of plugin.

If your Fledge system was installed from a package and you used the default installation path, then your plugin must be stored under the directory `/usr/local/fledge`. If you installed Fledge in a nonstandard location or you have built it from the source code, then the plugin should be stored under the directory `$FLEDGE_ROOT`.

A C/C++ plugin or a hybrid plugin should be placed in the directory `plugins/<type>/<plugin name>` under the installed directory described above. Where `<type>` is one of *south*, *filter*, *north*, *notificationRule* or *notificationDelivery*. And `<plugin name>` is the name you gave your plugin.

A south plugin written in C/C++ and called DHT11, for a system installed from a package, would be installed in a directory called `/usr/local/fledge/plugins/south/DHT11`. Within that directory Fledge would expect to find a file called `libDHT11.so`.

A south hybrid plugin called MD1421, for a development system built from source would be installed in `${FLEDGE_ROOT}/plugins/south/MD1421`. In this directory a JSON file called `MD1421.json` should exist, this is what the system will read to create the plugin.

A Python plugin should be installed in the directory `python/fledge/plugins/<plugin type>/<plugin name>` under the installed directory described above. Where `<type>` is one of *south*, *filter*, *north*, *notificationRule* or *notificationDelivery*. And `<plugin name>` is the name you gave your plugin.

A Python filter plugin called normalise, on a system installed from a package in the default location should be copied into a directory `/usr/local/fledge/python/fledge/plugins/filter/normalise`. Within this directory should be a file called `normalise.py` and an empty file called `__init__.py`.

### 13.14.1 Initial Testing

After you have copied your plugin into the correct location you can test if Fledge is able to see it by running the API call `/fledge/plugins/installed`. This will list all the installed plugins and their versions.

```
$ curl http://localhost:8081/fledge/plugins/installed | jq
{
  "plugins": [
    {
      "name": "http_north",
      "type": "north",
      "description": "HTTP North Plugin",
      "version": "1.8.1",
      "installedDirectory": "north/http_north",
      "packageName": "fledge-north-http-north"
    },
    {
      "name": "GCP",
      "type": "north",
      "description": "Google Cloud Platform IoT-Core",
      "version": "1.8.1",
      "installedDirectory": "north/GCP",
      "packageName": "fledge-north-gcp"
    },
    ...
  ]
}
```



Note, in the above example the *jq* program has been used to format the returned JSON and the output has been truncated for brevity.

If your plugin does not appear it may be because there was a problem loading it or because the *plugin\_info* call returned a bad value. Examine the syslog file to see if there are any errors recorded during the above API call.

### 13.14.2 C/C++ Common Faults

Common faults for C/C++ plugins are that a symbol could not be resolved when the plugin was loaded or the JSON for the default configuration is malformed.

There is a utility called *get\_plugin\_info* that is used by Python code to call the *C plugin\_info* call, this can be used to ascertain the cause of some problems. It should return the default configuration of your plugin and will verify that your plugin has no undefined symbols.

The location of *get\_plugin\_info* will depend on the type of installation you have. If you have built from source then it can be found in *./make\_build/C/plugins/utis/get\_plugin\_info*. If you have installed a package, or run *make install*, you can find it in */usr/local/fledge/extras/C/get\_plugin\_info*.

The utility is passed the library file of your plugin as its first argument and the function to call, usually *plugin\_info*.

```
$ get_plugin_info plugins/north/GCP/libGCP.so plugin_info
{"name": "GCP", "version": "1.8.1", "type": "north", "interface": "1.0.0", "flag": 0,
  "config": { "plugin": { "description": "Google Cloud Platform IoT-Core", "type":
    "string", "default": "GCP", "readonly": "true" }, "project_id": { "description":
    "The GCP IoT Core Project ID", "type": "string", "default": "", "order": "1",
    "displayName": "Project ID" }, "region": { "description": "The GCP Region", "type":
    "enumeration", "options": [ "us-centrall", "europe-west1", "asia-east1" ],
    "default": "us-centrall", "order": "2", "displayName": "The GCP Region" },
    "registry_id": { "description": "The Registry ID of the GCP Project", "type":
    "string", "default": "", "order": "3", "displayName": "Registry ID" }, "device_id":
    { "description": "Device ID within GCP IoT Core", "type": "string", "default":
    "", "order": "4", "displayName": "Device ID" }, "key": { "description": "Name
    of the key file to use", "type": "string", "default": "", "order": "5",
    "displayName": "Key Name" }, "algorithm": { "description": "JWT algorithm", "type":
    "enumeration", "options": [ "ES256", "RS256" ], "default": "RS256", "order":
    "6", "displayName": "JWT Algorithm" }, "source": { "description": "The source of
    data to send", "type": "enumeration", "default": "readings", "order": "8",
    "displayName": "Data Source", "options": [ "readings", "statistics" ] } } }
```

If there is an undefined symbol you will get an error from this utility. You can also check the validity of your JSON configuration by piping the output to a program such as *jq*.

```
$ get_plugin_info plugins/south/Random/libRandom.so plugin_info | jq
{
  "name": "Random",
  "version": "1.9.2",
  "type": "south",
  "interface": "1.0.0",
  "flag": 4096,
  "config": {
    "plugin": {
      "description": "Random data generation plugin",
      "type": "string",
      "default": "Random",
      "readonly": "true"
    }
  },
}
```

(continues on next page)



(continued from previous page)

```

    "asset": {
      "description": "Asset name",
      "type": "string",
      "default": "Random",
      "displayName": "Asset Name",
      "mandatory": "true"
    }
  }
}

```

### 13.14.3 Running Under a Debugger

If you have a C/C++ plugin that crashes you may want to run the plugin under a debugger. To build with debug symbols use the CMake option `-DCMAKE_BUILD_TYPE=Debug` when you create the *Makefile*.

#### Running a Service Under the Debugger

```
$ cmake -DCMAKE_BUILD_TYPE=Debug ..
```

The easiest approach to run under a debugger is

- Create the service that uses your plugin, say a south service and name that service as you normally would.
- Disable that service from being started by Fledge
- Use the fledge status script to find the arguments to pass the service

```

$ scripts/fledge status
Fledge v1.8.2 running.
Fledge Uptime: 1451 seconds.
Fledge records: 200889 read, 200740 sent, 120962 purged.
Fledge does not require authentication.
=== Fledge services:
fledge.services.core
fledge.services.storage --address=0.0.0.0 --port=39821
fledge.services.south --port=39821 --address=127.0.0.1 --name=AX8
fledge.services.south --port=39821 --address=127.0.0.1 --name=Sine
=== Fledge tasks:

```

- Note the `--port=` and `--address=` arguments
- Set your `LD_LIBRARY_PATH`. This is normally done in the script that launches Fledge but will need to be run as a manual step when running under the debugger.

```
export LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:/usr/local/fledge/lib
```

If you built from source rather than installing a package you will need to include the libraries you built

```
export LD_LIBRARY_PATH=${LD_LIBRARY_PATH}:${FLEDGE_ROOT}/cmake_
↳ build/C/lib
```

- Get a startup token by calling the Fledge API endpoint



*Note:* the caller must be authenticated as the *admin* user using either the username and password authentication or the certificate authentication mechanism in order to call the API endpoint. You must first set Fledge to require authentication. To do this, launch the Fledge GUI, navigate to Configuration and then Admin API. Set Authentication to *mandatory*. Authentication Method can be left as *any*.

In order to authenticate as the *admin* user one of the two following methods should be used, the choice of which is dependant on the authentication mechanism configured in your Fledge installation.

– User and password login

```
curl -d '{"username": "admin", "some_pass": "fledge"}' -X_
↪POST http://localhost:8081/fledge/login
```

Successful authentication will produce a response as shown below.

```
{ "message": "Logged in successfully", "uid": 1, "token":
↪"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.
↪eyJ1aWQiOiJEsImV4cCI6MTY1NDU5NTIyMn0.1lhIgQ93LbCP-
↪ztGlIuJVd6AJrBlbNBNvCv7SeuMfAs", "admin": true}
```

– Certificate login

```
curl -T /some_path/admin.cert -X POST http://
↪localhost:8081/fledge/login
```

Successful authentication will produce a response as shown below.

```
{ "message": "Logged in successfully", "uid": 1, "token":
↪"eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.
↪eyJ1aWQiOiJEsImV4cCI6MTY1NDU5NTkzN30.6VVD_
↪5RwmpLga2A7ri2bXhlo3x_CLqOYiefAAmLP63Y", "admin": true}
```

It is now possible to call the API endpoint to retrieve a startup token by passing the authentication token given in the authentication request.

```
curl -X POST 127.0.0.1:8081/fledge/service/ServiceName/otp -H
↪'authorization: Token'
```

Where *\*ServiceName\** is the name you gave your service when you\_  
↪created it and *\*Token\** received by the authentication request\_  
↪above.

This call will respond with a startup token that can be used to\_  
↪start the service you are debugging. An example response is shown\_  
↪below.

```
.. code-block:: console

{"startupToken": "WvFTYeGUvSEFMndePGbyvOsVYUzbnJdi"}
```

*\*startupToken\** will be passed as service start argument: --  
↪token=*\*startupToken\**

- Load the service you wish to use to run your plugin, e.g. a south service, under the debugger. This should be run from the FLEDGE\_ROOT directory



```
$ cd $FLEDGE_ROOT
$ gdb services/fledge.services.south
```

- Run the service passing the `--port=` and `--address=` arguments you noted above and add `-d` and `--name=` with the name of your service and `--token=startupToken`

```
(gdb) run --port=39821 --address=127.0.0.1 --name=ServiceName -d -
↳ --token=startupToken
```

Where *ServiceName* is the name you gave your service when you created it and *startupToken* is the token issued using the method described above. Note, this token may only be used once, each time the service is restarted using the debugger a new startup token must be obtained.

- You can now use the debugger in the way you normally would to find any issues.

---

**Note:** At this stage the plugins have not been loaded into the address space. If you try to set a break point in the plugin code you will get a warning that the break point can not currently be set. However when the plugin is later loaded the break point will be set and behave as expected.

---

Only the plugin has been built with debug information, if you wish to be able to single step into the library code that supports the plugin, and the services you must rebuild Fledge itself with debug symbols. There are multiple ways this can be done, but perhaps the simplest approach is to modify the *Makefile* in the route of the Fledge source.

When building Fledge the *cmake* command is executed by the make process, hence rather than manually running *cmake* and rebuilding you can simple alter the line

```
CMAKE := cmake
```

in the *Makefile* to read

```
CMAKE := cmake -DCMAKE_BUILD_TYPE=Debug
```

After making this change you should run a *make clean* followed by a *make* command

```
$ make clean
$ make
```

One side effect of this, caused by running *make clean* is that the plugins you have previously built have been removed from the `$FLEDGE_ROOT/plugins` directory and this must be rebuilt.

Alternatively you can manually build a debug version by running the following commands

```
$ cd $FLEDGE_ROOT/cmake_build
$ cmake -DCMAKE_BUILD_TYPE=Debug ..
$ make
```

This has the advantage that *make clean* is not run so your plugins will be preserved.



## Running a Task Under the Debugger

Running a task under the debugger is much the same as running a service, you will first need to find the management port and address of the core management service. Create the task, e.g. a north sending process in the same way as you normally would and disable it. You will also need to set your `LD_LIBRARY_PATH` as with running a service under the debugger.

If you are using a plugin with a task, such as the north sending process task, then the command to use to start the debugger is

```
$ gdb tasks/sending_process
```

## Running the Storage Service Under the Debugger

Running the storage service under the debugger is more difficult as you can not start the storage service after Fledge has started, the startup of the storage service is coordinated by the core due to the nature of how configuration is stored. It is possible however to attach a debugger to a running storage service.

- Run a command to find the process ID of the storage service

```
$ ps aux | grep fledge.services.storage
fledge 23318 0.0 0.3 270848 12388 ?        Ssl  10:00   0:01 /usr/
↳local/fledge/services/fledge.services.storage --address=0.0.0.0 --
↳port=33761
fledge 31033 0.0 0.0 13136 1084 pts/1    S+   10:37   0:00 grep --
↳color=auto fledge.services.storage
```

- Use the process ID of the fledge service as an argument to gdb. Note you will need to run gdb as root on some systems

```
$ sudo gdb /usr/local/fledge/services/fledge.services.storage 23318
GNU gdb (Ubuntu 8.1-0ubuntu3) 8.1.0.20180409-git
Copyright (C) 2018 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/
↳gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law.  Type "show
↳copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from services/fledge.services.storage...done.
Attaching to program: /usr/local/fledge/services/fledge.services.
↳storage, process 23318
[New LWP 23320]
[New LWP 23321]
[New LWP 23322]
[New LWP 23330]
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/x86_64-linux-gnu/libthread_db.
↳so.1".
```

(continues on next page)



(continued from previous page)

```
0x00007f47a3e05d2d in __GI___pthread_timedjoin_ex_
↳ (threadid=139945627997952, thread_return=0x0, abstime=0x0,
    block=<optimized out>) at pthread_join_common.c:89
89  pthread_join_common.c: No such file or directory.
(gdb)
```

- You can now use gdb to set break points etc and debug the storage service and plugins.

If you are debugging a plugin that crashes the system when readings are processed you should disable the south services until you have connected the debugger to the storage system. If you have a system that is setup and crashes, use the `--safe-mode` flag to the startup of Fledge in order to disable all processes and services. This will allow you to disable services or to run a particular service manually.

### 13.14.4 Using strace

You can also use a similar approach to that of running gdb to use the *strace* command to trace system calls and signals

- Create the service that uses your plugin, say a south service and name that service as you normally would.
- Disable that service from being started by Fledge
- Use the fledge status script to find the arguments to pass the service

```
$ scripts/fledge status
Fledge v1.8.2 running.
Fledge Uptime: 1451 seconds.
Fledge records: 200889 read, 200740 sent, 120962 purged.
Fledge does not require authentication.
=== Fledge services:
fledge.services.core
fledge.services.storage --address=0.0.0.0 --port=39821
fledge.services.south --port=39821 --address=127.0.0.1 --name=AX8
fledge.services.south --port=39821 --address=127.0.0.1 --name=Sine
=== Fledge tasks:
```

- Note the `--port=` and `--address=` arguments
- Run *strace* with the service adding the same set of arguments you used in gdb when running the service

```
$ strace services/fledge.services.south --port=39821 --address=127.0.0.1_
↳ --name=ServiceName --token=StartupToken -d
```

Where *ServiceName* is the name you gave your service and *startupToken* as issued following above steps.



### 13.14.5 Memory Leaks and Corruptions

The same approach can be used to make use of the *valgrind* command to find memory corruption and leak issues in your plugin

- Create the service that uses your plugin, say a south service and name that service as you normally would.
- Disable that service from being started by Fledge
- Use the fledge status script to find the arguments to pass the service

```
$ scripts/fledge status
Fledge v1.8.2 running.
Fledge Uptime: 1451 seconds.
Fledge records: 200889 read, 200740 sent, 120962 purged.
Fledge does not require authentication.
=== Fledge services:
fledge.services.core
fledge.services.storage --address=0.0.0.0 --port=39821
fledge.services.south --port=39821 --address=127.0.0.1 --name=AX8
fledge.services.south --port=39821 --address=127.0.0.1 --name=Sine
=== Fledge tasks:
```

- Note the *--port=* and *--address=* arguments
- Run *valgrind* with the service adding the same set of arguments you used in *gdb* when running the service.

Add any arguments you wish to pass to *valgrind* itself before the service executable name, in this case we are passing *--leak-check=full*.

```
$ valgrind --leak-check=full services/fledge.services.south --
↪port=39821 --address=127.0.0.1 --name=ServiceName --token=StartupToken_
↪-d
```

Where *ServiceName* is the name you gave your service and *startupToken* is a one time use token obtained following the steps shown above.

- Once the service has run for a while shut it down to trigger *valgrind* to print a summary of memory leaks found during the execution.

### 13.14.6 Python Plugin Info

It is also possible to test the loading and validity of the *plugin\_info* call in a Python plugin.

- From the */usr/include/fledge* or *\${FLEDGE\_ROOT}* directory run the command

```
python3 -c 'from fledge.plugins.south.<name>.<name> import plugin_info;
↪print(plugin_info())'
```

Where *<name>* is the name of your plugin.

```
python3 -c 'from fledge.plugins.south.sinusoid.sinusoid import plugin_info;
↪print(plugin_info())'
{'name': 'Sinusoid Poll plugin', 'version': '1.8.1', 'mode': 'poll', 'type':
↪'south', 'interface': '1.0', 'config': {'plugin': {'description': 'Sinusoid_
↪Poll Plugin which implements sine wave with data points', 'type': 'string',
↪'default': 'sinusoid', 'readonly': 'true'}, 'assetName': {'description': 'Name_
↪of Asset', 'type': 'string', 'default': 'sinusoid', 'displayName': '(continues on next page)
↪'mandatory': 'true'}}}
```



(continued from previous page)

This allows you to confirm the plugin can be loaded and the *plugin\_info* entry point can be called.

You can also check your default configuration. Although in Python this is usually harder to get wrong.

```
$ python3 -c 'from fledge.plugins.south.sinusoid.sinusoid import plugin_info;
↳ print(plugin_info()["config"])'
{'plugin': {'description': 'Sinusoid Poll Plugin which implements sine wave with data,
↳ points', 'type': 'string', 'default': 'sinusoid', 'readonly': 'true'}, 'assetName':
↳ {'description': 'Name of Asset', 'type': 'string', 'default': 'sinusoid',
↳ 'displayName': 'Asset name', 'mandatory': 'true'}}
```

## 13.15 Developing with Windows Subsystem for Linux (WSL2)

Windows Subsystem for Linux (WSL2) allows you to run a Linux environment directly on Windows without the overhead of Hyper-V on Windows 10 or a dual-boot setup. You can run many Linux command-line tools, utilities, and applications on a special lightweight virtual machine running on Windows. It is possible to run a complete Fledge system on WSL2. This includes the [Fledge GUI](#) which can be accessed from a browser running on the host Windows environment.

Microsoft's [Visual Studio Code](#) is a cross-platform editor that supports extensions for building and debugging software in a variety of languages and environments. This article describes how to configure Visual Studio Code to edit, build and debug Fledge plugins written in C++ running in Linux under WSL2.

---

**Note:** It is possible to configure Visual Studio Code to build and test Python code in WSL2 but this is not covered in this article.

---

### 13.15.1 Preparing the Development Environment

This section outlines the steps to configure WSL2 and the Linux environment.

#### Installing Windows Subsystem for Linux (WSL2)

You must be running Windows 10 version 2004 and higher (Build 19041 and higher) or Windows 11 to install WSL2. The easiest way to install is to open a Windows Command Prompt as Administrator and run this command:

```
wsl --install
```

Windows will perform all the necessary steps for you. It will install the default Linux distribution which is the latest version of Ubuntu. If you wish to perform the steps manually, or install a Linux distribution other than the default, see the Microsoft documentation on [Installing WSL](#).

When the installation completes, the Linux distribution will launch in a new window. It will prompt you for a username to serve as the root account and password. This username has nothing to do with your Windows environment so it can be any name you choose.

You can start the Linux distribution at any time by finding it in the Windows Start Menu. If you hit the Windows key and type the name of your Linux distribution (default: "Ubuntu"), you should see it immediately.



## Some Useful Features of WSL2

A Linux distribution running in WSL2 is a lightweight virtual machine but is well integrated with the Windows environment. Here are some useful features:

- *Cut and paste text into and out of the Linux window:* The Linux window behaves just like a Command Prompt window or a Powershell window. You can copy text from any window and paste it into any other.
- *Access the Linux file system from Windows:* The Linux file system appears as a Network drive in Windows. Open the Windows File Explorer and navigate to “\\wsl\$.” You will see your Linux distributions appear as network folders.
- *Access the Windows file system from Linux:* From the *bash* command line, navigate to the mount point “/mnt.” You will see your Windows drive letters in this directory.
- *Access the Linux environment from the Windows host through the network:* From the *bash* command line, run the command *hostname -I*. The external IP address returned by this command can be used in the Windows host to reach Linux.
- *Access the Windows host from the Linux environment through the network:* From the *bash* command line, run the command *cat /etc/resolv.conf*. The IP address after the label *nameserver* can be used in the Linux environment to reach the Windows host.

## Preparing the Linux Distribution for Fledge

The *systemd* service manager is not configured by default in an Ubuntu distribution running in WSL2. Since Fledge relies on *systemd*, you must run a script to enable it. From your home directory in the Ubuntu window, enter the commands:

```
git clone https://github.com/DamionGans/ubuntu-wsl2-systemd-script.git
cd ubuntu-wsl2-systemd-script
bash ubuntu-wsl2-systemd-script.sh
```

Restart the Ubuntu distribution using *sudo reboot* or *sudo systemctl reboot*. When the distribution has restarted, run the command *systemctl*. You should see no error and a list of units. The script must be run *one time only*. Whenever you start up your Ubuntu distribution, *systemd* should be ready.

## Installing Fledge

Following the normal instructions for [Installing Fledge on Ubuntu](#). Make sure the package repository matches your version of Ubuntu. You can check the operating system version in your distribution with the command *hostnamectl* or *cat /etc/os-release*.

## Installing Visual Studio Code

Navigate to the [Visual Studio Code](#) webpage in your Windows browser. Click the *Download for Windows* button. Run the installer to install Visual Studio Code.

Visual Studio Code is available for Microsoft Windows, Apple MacOS, and several Linux distributions. **Do not install the Linux build of Visual Studio Code in your Linux distribution in WSL2.** You will actually be launching Visual Studio Code for Windows from your Linux distribution!



## 13.15.2 Starting the Linux Distribution

Perform these steps every time you start your Linux distribution if you plan to run Fledge:

### Starting syslog

The system log `/var/log/syslog` is not configured to run automatically in a Linux distribution in WSL2. Start *syslog* with the command:

```
sudo service rsyslog start
```

You must do this at every startup.

### Starting Nginx

Fledge uses [Nginx](#) as a web server to host the Fledge GUI. If you plan to run Fledge GUI during your Linux distribution session, enter the command:

```
sudo service nginx start
```

You must do this at every startup if you plan to run the Fledge GUI.

### Starting Fledge

Start Fledge normally. You can start it from the normal run directory, or from your build directory by following the directions on the webpage [Testing Your Plugin](#).

### Starting Fledge GUI

If *Nginx* is running, you can run the Fledge GUI in a browser in your host Windows environment. Find the external IP address for your Linux distribution using the command:

```
hostname -I
```

This address is reachable from your Windows environment. Copy the IP address to a new tab in your browser and hit Enter. You should see the Fledge GUI Dashboard page.

---

**Note:** The Linux distribution's external IP address is (usually) different every time you start it. You will need to run the *hostname -I* command every time to obtain the current IP address.

---

## 13.15.3 Configuring Visual Studio Code

This section describes how to configure Visual Studio Code to edit, build and debug your C++ Linux projects. These instructions are summarized from the Visual Studio Code tutorial [Using C++ and WSL in VS Code](#).



## Installing Extensions

Navigate to a directory containing your C++ source code files and issue the command:

```
code .
```

This will launch Visual Studio Code in your Windows environment but it will be looking at the current directory in your Linux distribution. Since you are launching Visual Studio Code from your Linux distribution, Code should prompt you to install two Extensions:

- [Remote-WSL](#)
- [C/C++](#)

If you are not prompted, follow these links to install the extensions and restart Visual Studio Code. If the extensions are installed and working, you should see a green label in the lower left-hand corner of the Visual Studio Code window with the text *WSL:* followed by the name of your Linux distribution.

## Configuring your Workspace

Visual Studio Code refers to your directory of source code files as the *Workspace*. In order to edit, build and debug your code, you must create 3 Json files in a Workspace subdirectory called *.vscode*:

- **c\_cpp\_properties.json**: compiler path, IntelliSense settings, and include file paths
- **tasks.json**: build instructions
- **launch.json**: debugger settings

You can create these files manually or use Visual Studio Code's configuration wizards. These subsections describe creation and required contents of each of these three files.

### Code Editor Configuration: c\_cpp\_properties.json

- Open the Command Palette using the key sequence *Ctrl+Shift+P*.
- Choose the command *C/C++: Edit Configurations (JSON)*.
- This will create the *.vscode* subdirectory (if it doesn't already exist) and the *c\_cpp\_properties.json* file.
- This Json file will be opened for editing.
- You will see a new array called *configurations* with a single configuration object defined.
- This configuration will have a string array called *includePath*.
- Add the paths to your own include files, and those required by the Fledge API to the *includePath* array.
- You can use Linux environment variables in your paths. For example:

```
"${FLEDGE_ROOT}/C/common/include"
```

- You can find the list of include files by running your *make* command:

```
make --just-print
```

which will list all commands defined by *make* without executing them. You will see the include file list in every instance of the *gcc* compiler command.



### Build Configuration: tasks.json

- From the Visual Studio Code main menu, choose *Terminal -> Configure Default Build Task*.
- A dropdown will display of available tasks for C++ projects.
- Choose *g++ build active file*.
- This will create the `.vscode` subdirectory (if it doesn't already exist) and the `tasks.json` file.
- Open the Json file for editing.

Building the project will be done using the `make` file rather than the `gcc` compiler. To make this change, edit the `command` and `args` entries as follows:

```
"command": "make",
"args": [
  "-C",
  "${workspaceFolder}/build"
],
```

The “-C” argument for `make` will move into the specified directory before doing anything.

You can invoke a build from Visual Studio Code at any time with the key sequence `Ctrl+Shift+B`.

### Debugger Configuration: launch.json

- From the Visual Studio Code main menu, choose *Run -> Add Configuration...*
- Choose *C++ (GDB/LLDB)*.
- This will create the `.vscode` subdirectory (if it doesn't already exist) and the `launch.json` file.
- Edit the `launch.json` file so it looks like this:

```
{
  "version": "0.2.0",
  "configurations": [
    {
      "name": "Debug Plugin",
      "type": "cppdbg",
      "request": "launch",
      "targetArchitecture": "x86_64",
      "cwd": "${fileDirname}",
      "program": "/full/path/to/fledge.services.north",
      "externalConsole": false,
      "stopAtEntry": true,
      "MIMode": "gdb",
      "avoidWindowsConsoleRedirection": false,
      "args": [
        "--port=42467",
        "--address=0.0.0.0",
        "--name=MyPluginInstance",
        "-d"
      ]
    }
  ]
}
```



**Note:**

- The *program* attribute holds the program that the *gdb* debugger should launch. For Fledge plugin development, this is either *fledge.services.north* or *fledge.services.south* depending on which one you are building. These service executables will dynamically load your plugin library when they run.
  - The *args* attribute has the arguments normally passed to the service executable. Since the TCP/IP *port* changes every time Fledge starts up, you must edit this file to update the *port* number before starting your debug session.
- 

Start your debug session from the Visual Studio Code main menu. Choose *Run -> Start Debugging* or by hitting the F5 key.

### 13.15.4 Known Problems

- *Environment variables in launch.json*: Support for environment variables in the *program* attribute is inconsistent. Variables created by Visual Studio Code itself will work but user-defined environment variables like `FLEDGE_ROOT` will not.
- *gdb startup errors*: It can occur that *gdb* stops with error 42 and exits immediately when you start a debugging session. To fix this, shut down your Linux distributions and reinstall Visual Studio Code in Windows. You will not lose your configuration settings or your installed extensions.
- *Inconsistent breakpoint lists*: Visual Studio Code shows a list of breakpoints in the lower left corner of the window. The *gdb* debugger maintains its own list of breakpoints. It can occur that the two lists fall out of sync. You can still create, view and delete breakpoints from the *Debug Console* tab at the bottom of the screen which gives you access to the *gdb* command line. When using the *Debug Console*, you must precede all *gdb* commands with “*-exec*.”

**To manipulate breakpoints:**

- Set a breakpoint: *-exec b functionName*.
- View breakpoints: *-exec info b*. This will display an ordinal number for each breakpoint.
- Delete breakpoints: *-exec del ##*. Use the original number returned by *-exec info b* as “*##*.”

### 13.15.5 References

- [Visual Studio Code](#)
- [Using C++ and WSL in VS Code](#)
- [Remote development in WSL](#)
- [Debug C++ in Visual Studio Code](#)
- [Predefined Variables Reference](#)
- [C\\_cpp\\_properties.json reference](#)
- [Schema for tasks.json](#)
- [Configuring C/C++ Debugging \(launch.json\)](#)







## REST API DEVELOPERS GUIDE

### 14.1 The Fledge REST API

Users, administrators and applications interact with Fledge via a REST API. This section presents a full reference of the API.

---

**Note:** The Fledge REST API should not be confused with the internal REST API used by Fledge tasks and microservices to communicate with each other.

---

#### 14.1.1 Introducing the Fledge REST API

The REST API is the route into the Fledge appliance, it provides all user and program interaction to configure, monitor and manage the Fledge system. A separate specification will define the contents of the API, in summary however it is designed to allow for:

- The complete configuration of the Fledge appliance
- Access to monitoring statistics for the Fledge appliance
- User and role management for access to the API
- Access to the data buffer contents

#### Port Usage

In general Fledge components use dynamic port allocation to determine which port to use, the admin API is however an exception to this rule. The Admin API port has to be known to end-users and any user interface or management system that uses it, therefore the port on which the admin API listens must be consistent and fixed between invocations. This does not mean however that it can not be changed by the user. The user must have the option to define the port to use by the admin API to listen on. To achieve this the port will be stored in the configuration data for the admin API, using the configuration category *AdminAPI*, see Configuration. Administrators who have access to the appliance can find information regarding the port and the protocol to used (i.e. HTTP or HTTPS) in the *pid* file stored in *\$FLEDGE\_DATA/var/run/*:

```
$ cat data/var/run/fledge.core.pid
{ "adminAPI" : { "protocol" : "HTTP",
                 "port"      : 8081,
                 "addresses" : [ "0.0.0.0" ] },
  "processID" : 3585 }
$
```



Fledge is shipped with a default port for the admin API to use, however the user is free to change this after installation. This can be done by first connecting to the port defined as the default and then modifying the port using the admin API. Fledge should then be restarted to make use of this new port.

### Infrastructure

There are two REST API's that allow external access to Fledge, the **Administration API** and the **User API**. The User API is intended to allow access to the data in the Fledge storage layer which buffers sensor readings, and it is not part of this current version.

The Administration API is the first API is concerned with all aspects of managing and monitoring the Fledge appliance. This API is used for all configuration operations that occur beyond basic installation.

## 14.2 Administration API Reference

This section presents the list of administrative API methods in alphabetical order.

### 14.2.1 Audit Trail

The audit trail API is used to interact with the audit trail log tables in the storage microservice. In Fledge, log information is stored in the system log where the microservice is hosted. All the relevant information used for auditing are instead stored inside Fledge and they are accessible through the Admin REST API. The API allows the reading but also the addition of extra audit logs, as if such logs are created within the system.

#### audit

The *audit* methods implement the audit trail, they are used to create and retrieve audit logs.

#### GET Audit Entries

GET /fledge/audit - return a list of audit trail entries sorted with most recent first.

#### Request Parameters

- **limit** - limit the number of audit entries returned to the number specified
- **skip** - skip the first n entries in the audit table, used with limit to implement paged interfaces
- **source** - filter the audit entries to be only those from the specified source
- **severity** - filter the audit entries to only those of the specified severity

#### Response Payload

The response payload is an array of JSON objects with the audit trail entries.



Name	Type	Description	Example
timestamp	timestamp	The timestamp when the audit trail item was written.	2018-04-16 14:33:18.215
source	string	The source of the audit trail entry.	CoAP
severity	string	The severity of the event that triggered the audit trail entry to be written. This will be one of SUCCESS, FAILURE, WARNING or INFORMATION.	FAILURE
details	object	A JSON object that describes the detail of the audit trail event.	{ "message": "Sensor readings discarded due to malformed payload" }

### Example

```
$ curl -s http://localhost:8081/fledge/audit?limit=2
{ "totalCount" : 24,
  "audit"      : [ { "timestamp" : "2018-02-25 18:58:07.748",
                    "source"     : "SRVRG",
                    "details"    : { "name" : "COAP" },
                    "severity"   : "INFORMATION" },
                  { "timestamp" : "2018-02-25 18:58:07.742",
                    "source"     : "SRVRG",
                    "details"    : { "name" : "HTTP_SOUTH" },
                    "severity"   : "INFORMATION" },
                  { "timestamp" : "2018-02-25 18:58:07.390",
                    "source"     : "START",
                    "details"    : {},
                    "severity"   : "INFORMATION" }
                ]
}
$ curl -s http://localhost:8081/fledge/audit?source=SRVUN&limit=1
{ "totalCount" : 4,
  "audit"      : [ { "timestamp" : "2018-02-25 05:22:11.053",
                    "source"     : "SRVUN",
                    "details"    : { "name": "COAP" },
                    "severity"   : "INFORMATION" }
                ]
}
$
```

## POST Audit Entries

POST /fledge/audit - create a new audit trail entry.

The purpose of the create method on an audit trail entry is to allow a user interface or an application that is using the Fledge API to utilize the Fledge audit trail and notification mechanism to raise user defined audit trail entries.

### Request Payload

The request payload is a JSON object with the audit trail entry minus the timestamp.



Name	Type	Description	Example
source	string	The source of the audit trail entry.	LOGGN
severity	string	The severity of the event that triggered the audit trail entry to be written. This will be one of SUCCESS, FAILURE, WARNING or INFORMATION.	FAILURE
details	object	A JSON object that describes the detail of the audit trail event.	{ "message" : "Internal System Error" }

### Response Payload

The response payload is the newly created audit trail entry.

Name	Type	Description	Example
timestamp	timestamp	The timestamp when the audit trail item was written.	2018-04-16 14:33:18.215
source	string	The source of the audit trail entry.	LOGGN
severity	string	The severity of the event that triggered the audit trail entry to be written. This will be one of SUCCESS, FAILURE, WARNING or INFORMATION.	FAILURE
details	object	A JSON object that describes the detail of the audit trail event.	{ "message" : "Internal System Error" }

### Example

```
$ curl -X POST http://localhost:8081/fledge/audit \
-d '{ "severity": "FAILURE", "details": { "message": "Internal System Error" },
↪ "source": "LOGGN" }'
{ "source": "LOGGN",
  "timestamp": "2018-04-17 11:49:55.480",
  "severity": "FAILURE",
  "details": { "message": "Internal System Error" }
}
$
$ curl -X GET http://localhost:8081/fledge/audit?severity=FAILURE
{ "totalCount": 1,
  "audit": [ { "timestamp": "2018-04-16 18:32:28.427",
               "source" : "LOGGN",
               "details" : { "message": "Internal System Error" },
               "severity" : "FAILURE" }
            ]
}
$
```



## 14.2.2 Configuration Management

Configuration management is an important aspect of the REST API, however due to the discoverable form of the configuration of Fledge the API itself is fairly small.

The configuration REST API interacts with the configuration manager to create, retrieve, update and delete the configuration categories and values. Specifically all updates must go via the management layer as this is used to trigger the notifications to the components that have registered interest in configuration categories. This is the means by which the dynamic reconfiguration of Fledge is achieved.

### category

The *category* interface is part of the Configuration Management for Fledge and it is used to create, retrieve, update and delete configuration categories and items.

### GET categor(ies)

GET /fledge/category - return the list of known categories in the configuration database

#### Response Payload

The response payload is a JSON object with an array of JSON objects, one per valid category.

Name	Type	Description	Example
key	string	The category key, each category has a unique textual key that defines it.	network
description	string	A description of the category that may be used for display purposes.	Network Settings
display-Name	string	Name of the category that may be used for display purposes.	Network Settings

#### Example

```
$ curl -X GET http://localhost:8081/fledge/category
{
  "categories":
  [
    {
      "key": "SCHEDULER",
      "description": "Scheduler configuration",
      "displayName": "Scheduler"
    },
    {
      "key": "SMNTR",
      "description": "Service Monitor",
      "displayName": "Service Monitor"
    },
    {
      "key": "rest_api",
      "description": "Fledge Admin and User REST API",
      "displayName": "Admin API"
    },
    {
      "key": "service",
```

(continues on next page)



(continued from previous page)

```

    "description": "Fledge Service",
    "displayName": "Fledge Service"
  },
  {
    "key": "Installation",
    "description": "Installation",
    "displayName": "Installation"
  },
  {
    "key": "General",
    "description": "General",
    "displayName": "General"
  },
  {
    "key": "Advanced",
    "description": "Advanced",
    "displayName": "Advanced"
  },
  {
    "key": "Utilities",
    "description": "Utilities",
    "displayName": "Utilities"
  }
]
}
$

```

## GET category

GET /fledge/category/{name} - return the configuration items in the given category.

### Path Parameters

- **name** is the name of one of the categories returned from the GET /fledge/category call.

### Response Payload

The response payload is a set of configuration items within the category, each item is a JSON object with the following set of properties.

Name	Type	Description	Example
description	string	A description of the configuration item that may be used in a user interface.	The IPv4 network address of the Fledge server
type	string	A type that may be used by a user interface to know how to display an item.	IPv4
default	string	An optional default value for the configuration item.	127.0.0.1
displayName	string	Name of the category that may be used for display purposes.	IPv4 address
order	integer	Order at which category name will be displayed.	1
value	string	The current configured value of the configuration item. This may be empty if no value has been set.	192.168.0.27



**Example**

```
$ curl -X GET http://localhost:8081/fledge/category/rest_api
{
  "enableHttp": {
    "description": "Enable HTTP (disable to use HTTPS)",
    "type": "boolean",
    "default": "true",
    "displayName": "Enable HTTP",
    "order": "1",
    "value": "true"
  },
  "httpPort": {
    "description": "Port to accept HTTP connections on",
    "type": "integer",
    "default": "8081",
    "displayName": "HTTP Port",
    "order": "2",
    "value": "8081"
  },
  "httpsPort": {
    "description": "Port to accept HTTPS connections on",
    "type": "integer",
    "default": "1995",
    "displayName": "HTTPS Port",
    "order": "3",
    "validity": "enableHttp==\"false\"",
    "value": "1995"
  },
  "certificateName": {
    "description": "Certificate file name",
    "type": "string",
    "default": "fledge",
    "displayName": "Certificate Name",
    "order": "4",
    "validity": "enableHttp==\"false\"",
    "value": "fledge"
  },
  "authentication": {
    "description": "API Call Authentication",
    "type": "enumeration",
    "options": [
      "mandatory",
      "optional"
    ],
    "default": "optional",
    "displayName": "Authentication",
    "order": "5",
    "value": "optional"
  },
  "authMethod": {
    "description": "Authentication method",
    "type": "enumeration",
    "options": [
      "any",
      "password",
      "certificate"
    ],
    "value": "any"
  }
}
```

(continues on next page)



(continued from previous page)

```

    "default": "any",
    "displayName": "Authentication method",
    "order": "6",
    "value": "any"
  },
  "authCertificateName": {
    "description": "Auth Certificate name",
    "type": "string",
    "default": "ca",
    "displayName": "Auth Certificate",
    "order": "7",
    "value": "ca"
  },
  "allowPing": {
    "description": "Allow access to ping, regardless of the authentication required_
↪and authentication header",
    "type": "boolean",
    "default": "true",
    "displayName": "Allow Ping",
    "order": "8",
    "value": "true"
  },
  "passwordChange": {
    "description": "Number of days after which passwords must be changed",
    "type": "integer",
    "default": "0",
    "displayName": "Password Expiry Days",
    "order": "9",
    "value": "0"
  },
  "authProviders": {
    "description": "Authentication providers to use for the interface (JSON array_
↪object)",
    "type": "JSON",
    "default": "{\"providers\": [\"username\", \"ldap\"] }",
    "displayName": "Auth Providers",
    "order": "10",
    "value": "{\"providers\": [\"username\", \"ldap\"] }"
  }
}
$

```

## GET category item

GET /fledge/category/{name}/{item} - return the configuration item in the given category.

### Path Parameters

- **name** - the name of one of the categories returned from the GET /fledge/category call.
- **item** - the item within the category to return.

### Response Payload

The response payload is a configuration item within the category, each item is a JSON object with the following set of properties.



Name	Type	Description	Example
description	string	A description of the configuration item that may be used in a user interface.	The IPv4 network address of the Fledge server
type	string	A type that may be used by a user interface to know how to display an item.	IPv4
default	string	An optional default value for the configuration item.	127.0.0.1
displayName	string	Name of the category that may be used for display purposes.	IPv4 address
order	integer	Order at which category name will be displayed.	1
value	string	The current configured value of the configuration item. This may be empty if no value has been set.	192.168.0.27

### Example

```
$ curl -X GET http://localhost:8081/fledge/category/rest_api/httpsPort
{
  "description": "Port to accept HTTPS connections on",
  "type": "integer",
  "default": "1995",
  "displayName": "HTTPS Port",
  "order": "3",
  "validity": "enableHttp==\"false\"",
  "value": "1995"
}
$
```

### PUT category item

PUT /fledge/category/{name}/{item} - set the configuration item value in the given category.

#### Path Parameters

- **name** - the name of one of the categories returned from the GET /fledge/category call.
- **item** - the item within the category to set.

#### Request Payload

A JSON object with the new value to assign to the configuration item.

Name	Type	Description	Example
value	string	The new value of the configuration item.	192.168.0.27

#### Response Payload

The response payload is the newly updated configuration item within the category, the item is a JSON object with the following set of properties.



Name	Type	Description	Example
description	string	A description of the configuration item that may be used in a user interface.	The IPv4 network address of the Fledge server
type	string	A type that may be used by a user interface to know how to display an item.	IPv4
default	string	An optional default value for the configuration item.	127.0.0.1
displayName	string	Name of the category that may be used for display purposes.	IPv4 address
order	integer	Order at which category name will be displayed.	1
value	string	The current configured value of the configuration item. This may be empty if no value has been set.	192.168.0.27

### Example

```
$ curl -X PUT http://localhost:8081/fledge/category/rest_api/httpsPort \
-d '{ "value" : "1996" }'
{
  "description": "Port to accept HTTPS connections on",
  "type": "integer",
  "default": "1995",
  "displayName": "HTTPS Port",
  "order": "3",
  "validity": "enableHttp=="false"",
  "value": "1996"
}
$
```

### DELETE category item

DELETE /fledge/category/{name}/{item}/value - unset the value of the configuration item in the given category.

This will result in the value being returned to the default value if one is defined. If not the value will be blank, i.e. the value property of the JSON object will exist with an empty value.

#### Path Parameters

- **name** - the name of one of the categories returned from the GET /fledge/category call.
- **item** - the the item within the category to return.

#### Response Payload

The response payload is the newly updated configuration item within the category, the item is a JSON object object with the following set of properties.



Name	Type	Description	Example
description	string	A description of the configuration item that may be used in a user interface.	The IPv4 network address of the Fledge server
type	string	A type that may be used by a user interface to know how to display an item.	IPv4
default	string	An optional default value for the configuration item.	127.0.0.1
displayName	string	Name of the category that may be used for display purposes.	IPv4 address
order	integer	Order at which category name will be displayed.	1
value	string	The current configured value of the configuration item. This may be empty if no value has been set.	127.0.0.1

### Example

```
$ curl -X DELETE http://localhost:8081/fledge/category/rest_api/httpsPort/value
{
  "description": "Port to accept HTTPS connections on",
  "type": "integer",
  "default": "1995",
  "displayName": "HTTPS Port",
  "order": "3",
  "validity": "enableHttp==\"false\"",
  "value": "1995"
}
$
```

## POST category

POST /fledge/category - creates a new category

### Request Payload

A JSON object that defines the category.

Name	Type	Description	Example
key	string	The key that identifies the category. If the key already exists as a category then the contents of this request is merged with the data stored.	backup
description	string	A description of the configuration category	Backup configuration
items	list	An optional list of items to create in this category	
name	string	The name of a configuration item	destination
description	string	A description of the configuration item	The destination to which the backup will be written
type	string	The type of the configuration item	string
default	string	An optional default value for the configuration item	/backup



**NOTE:** with list we mean a list of JSON objects in the form of { obj1,obj2,etc. }, to differ from the concept of *array*, i.e. [ obj1,obj2,etc. ]

### Example

```
$ curl -X POST http://localhost:8081/fledge/category
-d '{ "key": "My Configuration", "description": "This is my new configuration",
      "value": { "item one": { "description": "The first item", "type": "string",
↪ "default": "one" },
                  "item two": { "description": "The second item", "type": "string",
↪ "default": "two" },
                  "item three": { "description": "The third item", "type": "string",
↪ "default": "three" } } }'
{ "description": "This is my new configuration", "key": "My Configuration", "value": {
  "item one": { "default": "one", "type": "string", "description": "The first item
↪ ", "value": "one" },
  "item two": { "default": "two", "type": "string", "description": "The second
↪ item", "value": "two" },
  "item three": { "default": "three", "type": "string", "description": "The third
↪ item", "value": "three" } }
}
$
```

## 14.2.3 Task Management

The task management API's allow an administrative user to monitor and control the tasks that are started by the task scheduler either from a schedule or as a result of an API request.

### task

The *task* interface allows an administrative user to monitor and control Fledge tasks.

### GET task

GET /fledge/task - return the list of all known task running or completed

#### Request Parameters

- **name** - an optional task name to filter on, only executions of the particular task will be reported.
- **state** - an optional query parameter that will return only those tasks in the given state.

#### Response Payload

The response payload is a JSON object with an array of task objects.



Name	Type	Description	Example
id	string	A unique identifier for the task. This takes the form of a uuid and not a Linux process id as the ID's must survive restarts and failovers	0a787bf3-4f48-4235-ae9a-2816f8ac76cc
name	string	The name of the task	purge
state	string	The current state of the task	Running
start-Time	times-tamp	The date and time the task started	2018-04-17 08:32:15.071
end-Time	times-tamp	The date and time the task ended This may not exist if the task is not completed.	2018-04-17 08:32:14.872
exit-Code	integer	Exit Code of the task.	0
reason	string	An optional reason string that describes why the task failed.	No destination available to write backup

### Example

```
$ curl -X GET http://localhost:8081/fledge/task
{
  "tasks": [
    {
      "id": "a9967d61-8bec-4d0b-8aa1-8b4dfb1d9855",
      "name": "stats collection",
      "processName": "stats collector",
      "state": "Complete",
      "startTime": "2020-05-28 09:21:58.650",
      "endTime": "2020-05-28 09:21:59.155",
      "exitCode": 0,
      "reason": ""
    },
    {
      "id": "7706b23c-71a4-410a-a03a-9b517dcd8c93",
      "name": "stats collection",
      "processName": "stats collector",
      "state": "Complete",
      "startTime": "2020-05-28 09:22:13.654",
      "endTime": "2020-05-28 09:22:14.160",
      "exitCode": 0,
      "reason": ""
    },
    ... ] }

$ curl -X GET http://localhost:8081/fledge/task?name=purge
{
  "tasks": [
    {
      "id": "c24e006d-22f2-4c52-9f3a-391a9b17b6d6",
      "name": "purge",
      "processName": "purge",
      "state": "Complete",
      "startTime": "2020-05-28 09:44:00.175",
      "endTime": "2020-05-28 09:44:13.915",
      "exitCode": 0,
      "reason": ""
    },
    {
```

(continues on next page)



(continued from previous page)

```

    "id": "609f35e6-4e89-4749-ac17-841ae3ee2b31",
    "name": "purge",
    "processName": "purge",
    "state": "Complete",
    "startTime": "2020-05-28 09:44:15.165",
    "endTime": "2020-05-28 09:44:28.154",
    "exitCode": 0,
    "reason": ""
  },
  ... ] }
$
$ curl -X GET http://localhost:8081/fledge/task?state=complete
{
"tasks": [
  {
    "id": "a9967d61-8bec-4d0b-8aa1-8b4dfb1d9855",
    "name": "stats collection",
    "processName": "stats collector",
    "state": "Complete",
    "startTime": "2020-05-28 09:21:58.650",
    "endTime": "2020-05-28 09:21:59.155",
    "exitCode": 0,
    "reason": ""
  },
  {
    "id": "7706b23c-71a4-410a-a03a-9b517dcd8c93",
    "name": "stats collection",
    "processName": "stats collector",
    "state": "Complete",
    "startTime": "2020-05-28 09:22:13.654",
    "endTime": "2020-05-28 09:22:14.160",
    "exitCode": 0,
    "reason": ""
  },
  ... ] }
$

```

## GET task latest

GET /fledge/task/latest - return the list of most recent task execution for each name.

This call is designed to allow a monitoring interface to show when each task was last run and what the status of that task was.

### Request Parameters

- **name** - an optional task name to filter on, only executions of the particular task will be reported.
- **state** - an optional query parameter that will return only those tasks in the given state.

### Response Payload

The response payload is a JSON object with an array of task objects.



Name	Type	Description	Example
id	string	A unique identifier for the task. This takes the form of a uuid and not a Linux process id as the ID's must survive restarts and failovers	0a787bf3-4f48-4235-ae9a-2816f8ac76cc
name	string	The name of the task	purge
state	string	The current state of the task	Running
start-Time	times-tamp	The date and time the task started	2018-04-17 08:32:15.071
end-Time	times-tamp	The date and time the task ended This may not exist if the task is not completed.	2018-04-17 08:32:14.872
exit-Code	integer	Exit Code of the task.	0
reason	string	An optional reason string that describes why the task failed.	No destination available to write backup
pid	integer	Process ID of the task.	17481

### Example

```
$ curl -X GET http://localhost:8081/fledge/task/latest
{
  "tasks": [
    {
      "id": "ea334d3b-8a33-4a29-845c-8be50efd44a4",
      "name": "certificate checker",
      "processName": "certificate checker",
      "state": "Complete",
      "startTime": "2020-05-28 09:35:00.009",
      "endTime": "2020-05-28 09:35:00.057",
      "exitCode": 0,
      "reason": "",
      "pid": 17481
    },
    {
      "id": "794707da-dd32-471e-8537-5d20dc0f401a",
      "name": "stats collection",
      "processName": "stats collector",
      "state": "Complete",
      "startTime": "2020-05-28 09:37:28.650",
      "endTime": "2020-05-28 09:37:29.138",
      "exitCode": 0,
      "reason": "",
      "pid": 17926
    }
  ]
}

$ curl -X GET http://localhost:8081/fledge/task/latest?name=purge
{
  "tasks": [
    {
      "id": "609f35e6-4e89-4749-ac17-841ae3ee2b31",
      "name": "purge",
      "processName": "purge",
      "state": "Complete",
      "startTime": "2020-05-28 09:44:15.165",
      "endTime": "2020-05-28 09:44:28.154",
```

(continues on next page)



(continued from previous page)

```

    "exitCode": 0,
    "reason": "",
    "pid": 20914
  }
]
}
$

```

## GET task by ID

GET /fledge/task/{id} - return the task information for the given task

### Path Parameters

- **id** - the uuid of the task whose data should be returned.

### Response Payload

The response payload is a JSON object containing the task details.

Name	Type	Description	Example
id	string	A unique identifier for the task. This takes the form of a uuid and not a Linux process id as the ID's must survive restarts and failovers	0a787bf3-4f48-4235-ae9a-2816f8ac76cc
name	string	The name of the task	purge
state	string	The current state of the task	Running
start-Time	times-tamp	The date and time the task started	2018-04-17 08:32:15.071
end-Time	times-tamp	The date and time the task ended This may not exist if the task is not completed.	2018-04-17 08:32:14.872
exit-Code	integer	Exit Code of the task.	0
reason	string	An optional reason string that describes why the task failed.	No destination available to write backup

### Example

```

$ curl -X GET http://localhost:8081/fledge/task/ea334d3b-8a33-4a29-845c-8be50efd44a4
{
  "id": "ea334d3b-8a33-4a29-845c-8be50efd44a4",
  "name": "certificate checker",
  "processName": "certificate checker",
  "state": "Complete",
  "startTime": "2020-05-28 09:35:00.009",
  "endTime": "2020-05-28 09:35:00.057",
  "exitCode": 0,
  "reason": ""
}
$

```



## Cancel task by ID

PUT /fledge/task/{id}/cancel - cancel a task

### Path Parameters

- **id** - the uuid of the task to cancel.

### Response Payload

The response payload is a JSON object with the details of the cancelled task.

Name	Type	Description	Example
id	string	A unique identifier for the task. This takes the form of a uuid and not a Linux process id as the ID's must survive restarts and failovers	0a787bf3-4f48-4235-ae9a-2816f8ac76cc
name	string	The name of the task	purge
state	string	The current state of the task	Running
start-Time	times-tamp	The date and time the task started	2018-04-17 08:32:15.071
end-Time	times-tamp	The date and time the task ended This may not exist if the task is not completed.	2018-04-17 08:32:14.872
reason	string	An optional reason string that describes why the task failed.	No destination available to write backup

### Example

```
$ curl -X PUT http://localhost:8081/fledge/task/ea334d3b-8a33-4a29-845c-8be50efd44a4/
↪cancel
{"id": "ea334d3b-8a33-4a29-845c-8be50efd44a4", "message": "Task cancelled successfully
↪"}
$
```

## 14.2.4 Other Administrative API calls

### ping

The *ping* interface gives a basic confidence check that the Fledge appliance is running and the API aspect of the appliance is functional. It is designed to be a simple test that can be applied by a user or by an HA monitoring system to test the liveness and responsiveness of the system.

### GET ping

GET /fledge/ping - return liveness of Fledge

**NOTE:** the GET method can be executed without authentication even when authentication is required. This behaviour is configurable via a configuration option.

### Response Payload

The response payload is some basic health information in a JSON object.



Name	Type	Description	Example
uptime	numeric	Time in seconds since Fledge started	2113.076449394226
dataRead	numeric	A count of the number of sensor readings	1452
dataSent	numeric	A count of the number of readings sent to PI	347
dataPurged	numeric	A count of the number of readings purged	226
authenticationOptional	boolean	When true, the REST API does not require authentication. When false, users must successfully login in order to call the rest API. Default is <i>true</i>	true
serviceName	string	Name of service	Fledge
hostName	string	Name of host machine	fledge
ipAddresses	list	IPv4 and IPv6 address of host machine	["10.0.0.0","123:234:345:456:567:789:1011:1213"]
health	string	Health of Fledge services	"green"
safeMode	boolean	True if Fledge is started in safe mode (only core and storage services will be started)	2113.076449394226

### Example

```
$ curl -s http://localhost:8081/fledge/ping
{
  "uptime": 276818,
  "dataRead": 0,
  "dataSent": 0,
  "dataPurged": 0,
  "authenticationOptional": true,
  "serviceName": "Fledge",
  "hostName": "fledge",
  "ipAddresses": [
    "x.x.x.x",
    "x:x:x:x:x:x:x:x"
  ],
  "health": "green",
  "safeMode": false
}
```

## statistics

The *statistics* interface allows the retrieval of live statistics and statistical history for the Fledge device.

### GET statistics

GET /fledge/statistics - return a general set of statistics

#### Response Payload

The response payload is a JSON document with statistical information (all numerical), these statistics are absolute counts since Fledge started.



Key	Description
BUFFERED	Readings currently in the Fledge buffer
DISCARDED	Readings discarded by the South Service before being placed in the buffer. This may be due to an error in the readings themselves.
PURGED	Readings removed from the buffer by the purge process
READINGS	Readings received by Fledge
UNSENT	Readings filtered out in the send process
UNSNPURGED	Readings that were purged from the buffer before being sent

### Example

```
$ curl -s http://localhost:8081/fledge/statistics
[ {
  "key": "BUFFERED",
  "description": "Readings currently in the Fledge buffer",
  "value": 0
},
...
{
  "key": "UNSNPURGED",
  "description": "Readings that were purged from the buffer before being sent",
  "value": 0
},
... ]
$
```

### GET statistics/history

GET /fledge/statistics/history - return a historical set of statistics. This interface is normally used to check if a set of sensors or devices are sending data to Fledge, by comparing the recent statistics and the number of readings received for an asset.

#### Request Parameters

- **limit** - limit the result set to the  $N$  most recent entries.

#### Response Payload

A JSON document containing an array of statistical information, these statistics are delta counts since the previous entry in the array. The time interval between values is a constant defined that runs the gathering process which populates the history statistics in the storage layer.

Key	Description
interval	The interval in seconds between successive statistics values
statistics[].BUFFERED	Readings currently in the Fledge buffer
statistics[].DISCARDED	Readings discarded by the South Service before being placed in the buffer. This may be due to an error in the readings themselves.
statistics[].PURGED	Readings removed from the buffer by the purge process
statistics[].READINGS	Readings received by Fledge
statistics[*NORTH_TASK_NAME*]	The number of readings sent to the PI system via the OMF plugin with north instance name
statistics[].UNSENT	Readings filtered out in the send process
statistics[].UNSNPURGED	Readings that were purged from the buffer before being sent
statistics[*ASSET-CODE*]	The number of readings received by Fledge since startup with name <i>asset-code</i>



### Example

```
$ curl -s http://localhost:8081/fledge/statistics/history?limit=2
{
  "interval": 15,
  "statistics": [
    {
      "history_ts": "2020-06-01 11:21:04.357",
      "READINGS": 0,
      "BUFFERED": 0,
      "UNSENT": 0,
      "PURGED": 0,
      "UNSNPURGED": 0,
      "DISCARDED": 0,
      "Readings Sent": 0
    },
    {
      "history_ts": "2020-06-01 11:20:48.740",
      "READINGS": 0,
      "BUFFERED": 0,
      "UNSENT": 0,
      "PURGED": 0,
      "UNSNPURGED": 0,
      "DISCARDED": 0,
      "Readings Sent": 0
    }
  ]
}
```

## 14.3 User API Reference

The user API provides a mechanism to access the data that is buffered within Fledge. It is designed to allow users and applications to get a view of the data that is available in the buffer and do analysis and possibly trigger actions based on recently received sensor readings.

In order to use the entry points in the user API, with the exception of the `/fledge/authenticate` entry point, there must be an authenticated client calling the API. The client must provide a header field in each request, `authtoken`, the value of which is the token that was retrieved via a call to `/fledge/authenticate`. This token must be checked for validity and also that the authenticated entity has user or admin permissions.

### 14.3.1 Browsing Assets

#### **asset**

The `asset` method is used to browse all or some assets, based on search and filtering.



## GET all assets

GET /fledge/asset - Return an array of asset codes buffered in Fledge and a count of assets by code.

### Response Payload

An array of JSON objects, one per asset.

Name	Type	Description	Example
[].assetCode	string	The code of the asset	fogbench/accelerometer
[].count	number	The number of recorded readings for the asset code	22359

### Example

```
$ curl -s http://localhost:8081/fledge/asset
[ { "count": 18, "assetCode": "fogbench/accelerometer" },
  { "count": 18, "assetCode": "fogbench/gyroscope" },
  { "count": 18, "assetCode": "fogbench/humidity" },
  { "count": 18, "assetCode": "fogbench/luxometer" },
  { "count": 18, "assetCode": "fogbench/magnetometer" },
  { "count": 18, "assetCode": "fogbench/mouse" },
  { "count": 18, "assetCode": "fogbench/pressure" },
  { "count": 18, "assetCode": "fogbench/switch" },
  { "count": 18, "assetCode": "fogbench/temperature" },
  { "count": 18, "assetCode": "fogbench/wall clock" } ]
$
```

## GET asset readings

GET /fledge/asset/{code} - Return an array of readings for a given asset code.

### Path Parameters

- **code** - the asset code to retrieve.

### Request Parameters

- **limit** - set the limit of the number of readings to return. If not specified, the default is 20 readings.

### Response Payload

An array of JSON objects with the readings data for a series of readings sorted in reverse chronological order.

Name	Type	Description	Example
[].timestamp	timestamp	The time at which the reading was received.	2018-04-16 14:33:18.215
[].reading	JSON object	The JSON reading object received from the sensor.	{ "reading": { "x": 0, "y": 0, "z": 1 } }

### Example

```
$ curl -s http://localhost:8081/fledge/asset/fogbench%2Faccelerometer
[ { "reading": { "x": 0, "y": -2, "z": 0 }, "timestamp": "2018-04-19 14:20:59.692" },
  { "reading": { "x": 0, "y": 0, "z": -1 }, "timestamp": "2018-04-19 14:20:54.643" },
  { "reading": { "x": -1, "y": 2, "z": 1 }, "timestamp": "2018-04-19 14:20:49.899" },
  { "reading": { "x": -1, "y": -1, "z": 1 }, "timestamp": "2018-04-19 14:20:47.026" },
```

(continues on next page)



(continued from previous page)

```

{ "reading": { "x": -1, "y": -2, "z": -2 }, "timestamp": "2018-04-19 14:20:42.746" }
→,
{ "reading": { "x": 0, "y": 2, "z": 0 }, "timestamp": "2018-04-19 14:20:37.418" },
{ "reading": { "x": -2, "y": -1, "z": 2 }, "timestamp": "2018-04-19 14:20:32.650" },
{ "reading": { "x": 0, "y": 0, "z": 1 }, "timestamp": "2018-04-19 14:06:05.870" },
{ "reading": { "x": 1, "y": 1, "z": 1 }, "timestamp": "2018-04-19 14:06:05.870" },
{ "reading": { "x": 0, "y": 0, "z": -1 }, "timestamp": "2018-04-19 14:06:05.869" },
{ "reading": { "x": 2, "y": -1, "z": 0 }, "timestamp": "2018-04-19 14:06:05.868" },
{ "reading": { "x": -1, "y": -2, "z": 2 }, "timestamp": "2018-04-19 14:06:05.867" },
{ "reading": { "x": 2, "y": 1, "z": 1 }, "timestamp": "2018-04-19 14:06:05.867" },
{ "reading": { "x": 1, "y": -2, "z": 1 }, "timestamp": "2018-04-19 14:06:05.866" },
{ "reading": { "x": 2, "y": -1, "z": 1 }, "timestamp": "2018-04-19 14:06:05.865" },
{ "reading": { "x": 0, "y": -1, "z": 2 }, "timestamp": "2018-04-19 14:06:05.865" },
{ "reading": { "x": 0, "y": -2, "z": 1 }, "timestamp": "2018-04-19 14:06:05.864" },
{ "reading": { "x": -1, "y": -2, "z": 0 }, "timestamp": "2018-04-19 13:45:15.881" }
→ ]
$
$ curl -s http://localhost:8081/fledge/asset/fogbench%2Faccelerometer?limit=5
[ { "reading": { "x": 0, "y": -2, "z": 0 }, "timestamp": "2018-04-19 14:20:59.692" },
{ "reading": { "x": 0, "y": 0, "z": -1 }, "timestamp": "2018-04-19 14:20:54.643" },
{ "reading": { "x": -1, "y": 2, "z": 1 }, "timestamp": "2018-04-19 14:20:49.899" },
{ "reading": { "x": -1, "y": -1, "z": 1 }, "timestamp": "2018-04-19 14:20:47.026" },
{ "reading": { "x": -1, "y": -2, "z": -2 }, "timestamp": "2018-04-19 14:20:42.746" }
→ ]
$

```

## GET asset reading

GET /fledge/asset/{code}/{reading} - Return an array of single readings for a given asset code.

### Path Parameters

- **code** - the asset code to retrieve.
- **reading** - the sensor from the assets JSON formatted reading.

### Request Parameters

- **limit** - set the limit of the number of readings to return. If not specified, the defaults is 20 single readings.

### Response Payload

An array of JSON objects with a series of readings sorted in reverse chronological order.

Name	Type	Description	Example
timestamp	timestamp	The time at which the reading was received.	2018-04-16 14:33:18.215
{reading}	JSON object	The value of the specified reading.	"temperature": 20

### Example

```

$ curl -s http://localhost:8081/fledge/asset/fogbench%2Fhumidity/temperature
[ { "temperature": 20, "timestamp": "2018-04-19 14:20:59.692" },
{ "temperature": 33, "timestamp": "2018-04-19 14:20:54.643" },
{ "temperature": 35, "timestamp": "2018-04-19 14:20:49.899" },
{ "temperature": 0, "timestamp": "2018-04-19 14:20:47.026" },
{ "temperature": 37, "timestamp": "2018-04-19 14:20:42.746" },

```

(continues on next page)



(continued from previous page)

```

{ "temperature": 47, "timestamp": "2018-04-19 14:20:37.418" },
{ "temperature": 26, "timestamp": "2018-04-19 14:20:32.650" },
{ "temperature": 12, "timestamp": "2018-04-19 14:06:05.870" },
{ "temperature": 38, "timestamp": "2018-04-19 14:06:05.869" },
{ "temperature": 7, "timestamp": "2018-04-19 14:06:05.869" },
{ "temperature": 21, "timestamp": "2018-04-19 14:06:05.868" },
{ "temperature": 5, "timestamp": "2018-04-19 14:06:05.867" },
{ "temperature": 40, "timestamp": "2018-04-19 14:06:05.867" },
{ "temperature": 39, "timestamp": "2018-04-19 14:06:05.866" },
{ "temperature": 29, "timestamp": "2018-04-19 14:06:05.865" },
{ "temperature": 41, "timestamp": "2018-04-19 14:06:05.865" },
{ "temperature": 46, "timestamp": "2018-04-19 14:06:05.864" },
{ "temperature": 10, "timestamp": "2018-04-19 13:45:15.881" } ]
$
$ curl -s http://localhost:8081/fledge/asset/fogbench%2Faccelerometer?limit=5
[ { "temperature": 20, "timestamp": "2018-04-19 14:20:59.692" },
  { "temperature": 33, "timestamp": "2018-04-19 14:20:54.643" },
  { "temperature": 35, "timestamp": "2018-04-19 14:20:49.899" },
  { "temperature": 0, "timestamp": "2018-04-19 14:20:47.026" },
  { "temperature": 37, "timestamp": "2018-04-19 14:20:42.746" } ]
$

```

## GET asset reading summary

GET /fledge/asset/{code}/{reading}/summary - Return minimum, maximum and average values of a reading by asset code.

### Path Parameters

- **code** - the asset code to retrieve.
- **reading** - the sensor from the assets JSON formatted reading.

### Response Payload

An array of JSON objects with a series of readings sorted in reverse chronological order.

Name	Type	Description	Example
{reading}.average	number	The average value of the set of sensor values selected in the query string	27
{reading}.min	number	The minimum value of the set of sensor values selected in the query string	0
{reading}.max	number	The maximum value of the set of sensor values selected in the query string	47

### Example

```

$ curl -s http://localhost:8081/fledge/asset/fogbench%2Fhumidity/temperature/summary
{ "temperature": { "max": 47, "min": 0, "average": 27 } }
$

```



## GET timed average asset reading

GET /fledge/asset/{code}/{reading}/series - Return minimum, maximum and average values of a reading by asset code in a time series. The default interval in the series is one second.

### Path Parameters

- **code** - the asset code to retrieve.
- **reading** - the sensor from the assets JSON formatted reading.

### Request Parameters

- **limit** - set the limit of the number of readings to return. If not specified, the defaults is 20 single readings.

### Response Payload

An array of JSON objects with a series of readings sorted in reverse chronological order.

Name	Type	Description	Example
times-tamp	times-tamp	The time the reading represents.	2018-04-16 14:33:18
average	number	The average value of the set of sensor values selected in the query string	27
min	number	The minimum value of the set of sensor values selected in the query string	0
max	number	The maximum value of the set of sensor values selected in the query string	47

### Example

```
$ curl -s http://localhost:8081/fledge/asset/fogbench%2Fhumidity/temperature/series
[ { "timestamp": "2018-04-19 14:20:59", "max": 20, "min": 20, "average": 20 },
  { "timestamp": "2018-04-19 14:20:54", "max": 33, "min": 33, "average": 33 },
  { "timestamp": "2018-04-19 14:20:49", "max": 35, "min": 35, "average": 35 },
  { "timestamp": "2018-04-19 14:20:47", "max": 0, "min": 0, "average": 0 },
  { "timestamp": "2018-04-19 14:20:42", "max": 37, "min": 37, "average": 37 },
  { "timestamp": "2018-04-19 14:20:37", "max": 47, "min": 47, "average": 47 },
  { "timestamp": "2018-04-19 14:20:32", "max": 26, "min": 26, "average": 26 },
  { "timestamp": "2018-04-19 14:06:05", "max": 46, "min": 5, "average": 27.8 },
  { "timestamp": "2018-04-19 13:45:15", "max": 10, "min": 10, "average": 10 } ]

$
$ curl -s http://localhost:8081/fledge/asset/fogbench%2Fhumidity/temperature/series
[ { "timestamp": "2018-04-19 14:20:59", "max": 20, "min": 20, "average": 20 },
  { "timestamp": "2018-04-19 14:20:54", "max": 33, "min": 33, "average": 33 },
  { "timestamp": "2018-04-19 14:20:49", "max": 35, "min": 35, "average": 35 },
  { "timestamp": "2018-04-19 14:20:47", "max": 0, "min": 0, "average": 0 },
  { "timestamp": "2018-04-19 14:20:42", "max": 37, "min": 37, "average": 37 } ]
```



## BUILDING FLEDGE

### 15.1 Building Developers Guide

#### 15.1.1 Introduction

##### What Is Fledge?

Fledge is an open source platform for the **Internet of Things** and an essential component in **Fog Computing**. It uses a modular **microservices architecture** including sensor data collection, storage, processing and forwarding to historians, Enterprise systems and Cloud-based services. Fledge can run in highly available, stand alone, unattended environments that assume unreliable network connectivity.

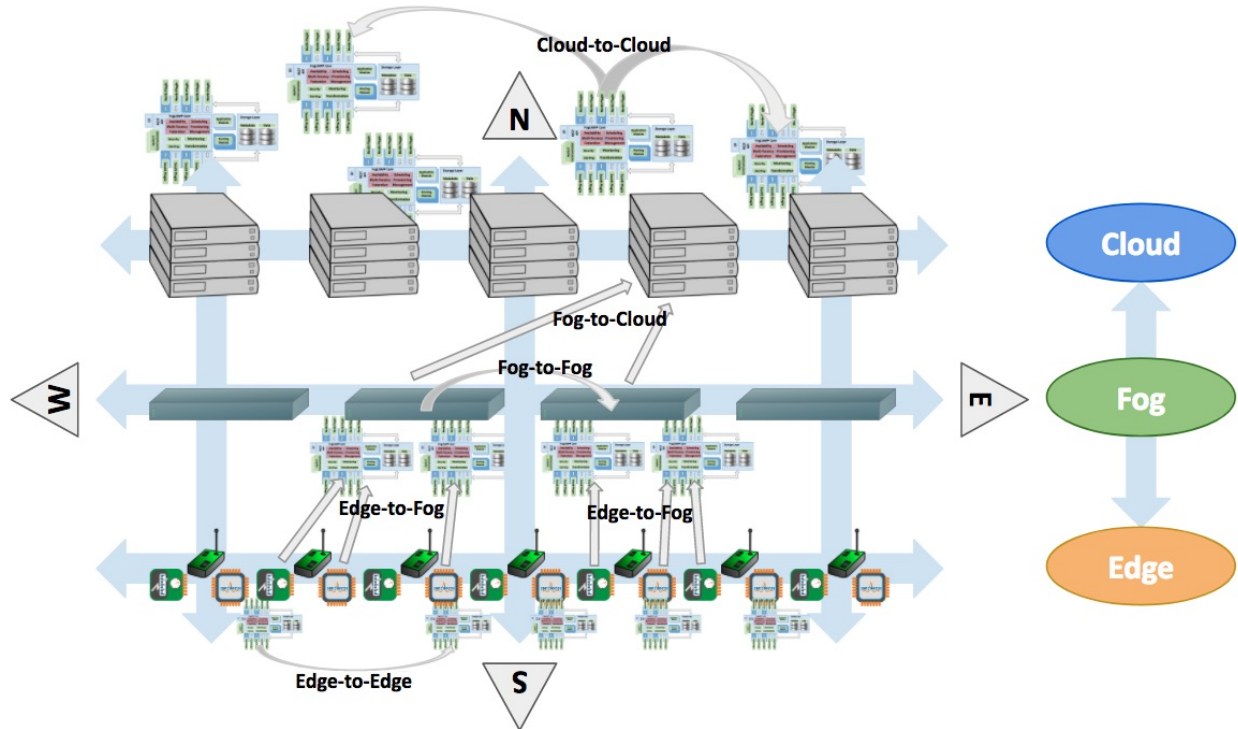
By providing a modular and distributable framework under an open source Apache v2 license, Fledge is the best platform to manage the data infrastructure for IoT. The modules can be distributed in any layer - Edge, Fog and Cloud - and they act together to provide scalability, elasticity and resilience.

Fledge offers an “all-round” solution for data management, combining a bi-directional **Northbound/Southbound** data and metadata communication with a **Eastbound/Westbound** service and object distribution.

##### Fledge Positioning in an IoT and IIoT Infrastructure

Fledge can be used in IoT and IIoT infrastructure at Edge and in the Fog. It stretches bi-directionally South-North/North-South and it is distributed East-West/West-East (see figure below).





**Note:** In this scenario we refer to “Cloud” as the layer above the Fog. “Fog” is where historians, gateways and middle servers coexist. In practice, the Cloud may also represent internal Enterprise systems, concentrated in regional or global corporate data centers, where larger historians, Big Data and analytical systems reside.

In practical terms, this means that:

- Intra-layer communication and data exchange:
  - At the **Edge**, microservices are installed on devices, sensors and actuators.
  - In the **Fog**, data is collected and aggregated in gateways and regional servers.
  - In the **Cloud**, data is distributed and analysed on multiple servers, such as Big Data Systems and Data Historians.
- Inter-layer communication and data exchange:
  - From **Edge to Fog**, data is retrieved from multiple sensors and devices and it is aggregated on resilient and highly available middle servers and gateways, either in traditional Data Historians and in the new edge of Machine Learning systems.
  - From **Fog to Edge**, configuration information, metadata and other valuable data is transferred to sensors and devices.
  - From **Fog to Cloud**, the data collected and optionally transformed is transferred to more powerful distributed Cloud and Enterprise systems.
  - From **Cloud to Fog**, results of complex analysis and other valuable information are sent to the designated gateways and middle servers that will interact with the Edge.
- Intra-layer service distribution:



- A microservice architecture based on secure communication allows lightweight service distribution and information exchange among **Edge to Edge** devices.
- Fledge provides high availability, scalability and data distribution among **Fog-to-Fog** systems. Due to its portability and modularity, Fledge can be installed on a large number of intermediate servers and gateways, as application instances, appliances, containers or virtualized environments.
- **Cloud to Cloud Fledge server** capabilities provide scalability and elasticity in data storage, retrieval and analytics. The data collected at the Edge and Fog, also combined with external data, can be distributed to multiple systems within a Data Center and replicated to multiple Data Centers to guarantee local and faster access.

All these operations are **scheduled, automated and executed securely, unattended** and in a **transactional** fashion (i.e. the system can always revert to a previous state in case of failures or unexpected events).

## Fledge Features

In a nutshell, these are main features of Fledge:

- Transactional, always on, server platform designed to work unattended and with zero maintenance.
- Microservice architecture with secured inter-communication:
  - Core System
  - Storage Layer
  - South side, sensors and device communication
  - North side, Cloud and Enterprise communication
  - Application Modules, internal application logic
- Pluggable modules for:
  - South side: multiple, data and metadata bi-directional communication
  - North side: multiple, data and metadata bi-directional communication
  - East/West side: IN/OUT Communicator with external applications
  - Plus:
    - \* Data and communication authentication
    - \* Data and status monitoring and alerting
    - \* Data transformation
    - \* Data storage and retrieval
- Small memory and processing footprint. Fledge can be installed and executed on inexpensive Edge devices; microservices can be distributed on sensors and actuator boards.
- Resilient and optionally highly available.
- Discoverable and cluster-based.
- Based on APIs (RESTful and non-RESTful) to communicate with sensors and other devices, to interact with user applications, to manage the platform and to be integrated with a Cloud or Data Center-based data infrastructure.
- Hardened with default secure communication that can be optionally relaxed.



### 15.1.2 Building Fledge

Let's get started! In this chapter we will see where to find and how to build, install and run Fledge for the first time.

#### Fledge Platforms

Due to the use of standard libraries, Fledge can run on a large number of platforms and operating environments, but its primary target is Linux distributions. Our testing environment includes Ubuntu 18.04 LTS, Ubuntu 20.04 LTS and Raspbian, but we have installed and tested Fledge on other Linux distributions. In addition to the native support, Fledge can also run on Virtual Machines, Docker and LXC containers.

#### Requirements

Fledge requires a number of software packages and libraries to be installed in order to be built, the process of installing these has been streamlined and automated for all the currently supported platforms. A single script, *requirements.sh* can be run and this will install all of the packages needed to to build and run Fledge.

#### Building Fledge

In this section we will describe how to build Fledge on any of the supported platforms. If you are not familiar with Linux and you do not want to build Fledge from the source code, you can download a ready-made package (the list of packages is [available here](#)).

#### Obtaining the Source Code

Fledge is available on GitHub. The link to the repository is . In order to clone the code in the repository, type:

```
$ git clone https://github.com/fledge-iot/fledge.git
Cloning into 'fledge'...
remote: Enumerating objects: 83394, done.
remote: Counting objects: 100% (2093/2093), done.
remote: Compressing objects: 100% (903/903), done.
remote: Total 83394 (delta 1349), reused 1840 (delta 1161), pack-reused 81301
Receiving objects: 100% (83394/83394), 34.85 MiB | 7.38 MiB/s, done.
Resolving deltas: 100% (55599/55599), done.
$
```

The code should now be loaded on your machine in a directory called *fledge*. The name of the repository directory is *fledge*:

```
$ ls -l fledge
total 228
drwxrwxr-x  7 fledge fledge  4096 Aug 26 11:20 C
-rw-rw-r--  1 fledge fledge  1659 Aug 26 11:20 CMakeLists.txt
drwxrwxr-x  2 fledge fledge  4096 Aug 26 11:20 contrib
-rw-rw-r--  1 fledge fledge  4786 Aug 26 11:20 CONTRIBUTING.md
drwxrwxr-x  4 fledge fledge  4096 Aug 26 11:20 data
drwxrwxr-x  2 fledge fledge  4096 Aug 26 11:20 dco-signoffs
drwxrwxr-x 10 fledge fledge  4096 Aug 26 11:20 docs
-rw-rw-r--  1 fledge fledge 108680 Aug 26 11:20 doxy.config
drwxrwxr-x  3 fledge fledge  4096 Aug 26 11:20 examples
drwxrwxr-x  4 fledge fledge  4096 Aug 26 11:20 extras
```

(continues on next page)



(continued from previous page)

```

-rw-rw-r-- 1 fledge fledge 11346 Aug 26 11:20 LICENSE
-rw-rw-r-- 1 fledge fledge 24216 Aug 26 11:20 Makefile
-rwxrwxr-x 1 fledge fledge 310 Aug 26 11:20 mkversion
drwxrwxr-x 4 fledge fledge 4096 Aug 26 11:20 python
-rw-rw-r-- 1 fledge fledge 9292 Aug 26 11:20 README.rst
-rwxrwxr-x 1 fledge fledge 8177 Aug 26 11:20 requirements.sh
drwxrwxr-x 8 fledge fledge 4096 Aug 26 11:20 scripts
drwxrwxr-x 4 fledge fledge 4096 Aug 26 11:20 tests
drwxrwxr-x 3 fledge fledge 4096 Aug 26 11:20 tests-manual
-rwxrwxr-x 1 fledge fledge 38 Aug 26 11:20 VERSION
$

```

## Selecting the Correct Version

The git repository created on your local machine, creates several branches. More specifically:

- The **main** branch is the latest, stable version. You should use this branch if you are interested in using Fledge with the last release features and fixes.
- The **develop** branch is the current working branch used by our developers. The branch contains the latest version and features, but it may be unstable and there may be issues in the code. You may consider to use this branch if you are curious to see one of the latest features we are working on, but you should not use this branch in production.
- The branches with versions **majorID.minorID** or **majorID.minorID.patchID**, such as *1.0* or *1.4.2*, contain the code of that specific version. You may use one of these branches if you need to check the code used in those versions.
- The branches with name **FOGL-XXXX**, where 'XXXX' is a sequence number, are working branches used by developers and contributors to add features, fix issues, modify and release code and documentation of Fledge. Those branches are free for you to see and learn from the work of the contributors.

Note that the default branch is *develop*.

Once you have cloned the Fledge project, in order to check the branches available, use the `git branch` command:

```

$ pwd
/home/ubuntu
$ cd fledge
$ git branch --all
* develop
remotes/origin/1.0
...
remotes/origin/FOGL-822
remotes/origin/FOGL-823
remotes/origin/HEAD -> origin/develop
...
remotes/origin/develop
remotes/origin/main
$

```

Assuming you want to use the latest released, stable version, use the `git checkout` command to select the *master* branch:

```

$ git checkout main
Branch main set up to track remote branch main from origin.

```

(continues on next page)



(continued from previous page)

```
Switched to a new branch 'main'
$
```

You can always use the `git status` command to check the branch you have checked out.

### Building Fledge

You are now ready to build your first Fledge project.

- Move to the *fledge* project directory
- Load the requirements needed to build Fledge by typing

```
$ sudo ./requirements.sh
[sudo] password for john:
Platform is Ubuntu, Version: 18.04
apt 1.6.14 (amd64)
Reading package lists...
Building dependency tree...
...
```

- Type the `make` command and let the magic happen.

```
$ make
Building Fledge version X.X., DB schema X
scripts/certificates "fledge" "365"
Creating a self signed SSL certificate ...
Certificates created successfully, and placed in data/etc/certs
scripts/auth_certificates ca "ca" "365"
...
Successfully installed aiohttp-3.8.1 aiohttp-cors-0.7.0 aiosignal-1.2.0 async-
↳timeout-4.0.2 asynctest-0.13.0 attrs-22.1.0 cchardet-2.1.4 certifi-2022.6.15
↳charset-normalizer-2.1.1 frozenlist-1.2.0 idna-3.3 idna-ssl-1.1.0 ifaddr-0.2.0
↳multidict-5.2.0 pyjq-2.3.1 pyjwt-1.6.4 requests-2.27.1 requests-toolbelt-0.9.1
↳six-1.16.0 typing-extensions-4.1.1 urllib3-1.26.12 yarl-1.7.2 zeroconf-0.27.0
$
```

Depending on the version of Ubuntu or other Linux distribution you are using, you may have found some issues. For example, there is a bug in the GCC compiler that raises a warning under specific circumstances. The output of the build will be something like:

```
/home/ubuntu/Fledge/C/services/storage/storage.cpp:97:14: warning: ignoring return
↳value of 'int dup(int)', declared with attribute warn_unused_result [-Wunused-
↳result]
    (void)dup(0);          // stdout GCC bug 66425 produces warning
           ^
/home/ubuntu/Fledge/C/services/storage/storage.cpp:98:14: warning: ignoring return
↳value of 'int dup(int)', declared with attribute warn_unused_result [-Wunused-
↳result]
    (void)dup(0);          // stderr GCC bug 66425 produces warning
           ^
```

The bug is documented . For our project, you should ignore it.

The other issue is related to the version of pip (more specifically pip3), the Python package manager. If you see this warning in the middle of the build output:



```
/usr/lib/python3.6/distutils/dist.py:261: UserWarning: Unknown distribution option:
↪ 'python_requires'
  warnings.warn(msg)
```

... and this output at the end of the build process:

```
You are using pip version 8.1.1, however version 9.0.1 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.
```

In this case, what you need to do is to upgrade the pip software for Python3:

```
$ sudo pip3 install --upgrade pip
Collecting pip
  Downloading pip-9.0.1-py2.py3-none-any.whl (1.3MB)
    100% || 1.3MB 1.1MB/s
Installing collected packages: pip
Successfully installed pip-9.0.1
$
```

At this point, run the make command again and the Python warning should disappear.

## Testing Fledge from the Build Environment

If you are eager to test Fledge straight away, you can do so! All you need to do is to set the *FLEDGE\_ROOT* environment variable and you are good to go. Stay in the Fledge project directory, set the environment variable with the path to the Fledge directory and start fledge with the `fledge start` command:

```
$ pwd
/home/ubuntu/fledge
$ export FLEDGE_ROOT=/home/ubuntu/fledge
$ ./scripts/fledge start
Starting Fledge vX.X.....
Fledge started.
$
```

You can check the status of Fledge with the `fledge status` command. For few seconds you may see service starting, then it will show the status of the Fledge services and tasks:

```
$ ./scripts/fledge status
Fledge starting.
$
$ scripts/fledge status
Fledge vX.X.X running.
Fledge Uptime: 9065 seconds.
Fledge records: 86299 read, 86851 sent, 0 purged.
Fledge does not require authentication.
=== Fledge services:
fledge.services.core
fledge.services.storage --address=0.0.0.0 --port=42583
fledge.services.south --port=42583 --address=127.0.0.1 --name=Sine
fledge.services.notification --port=42583 --address=127.0.0.1 --name=Fledge_
↪ Notifications
=== Fledge tasks:
fledge.tasks.purge --port=42583 --address=127.0.0.1 --name=purge
tasks/sending_process --port=42583 --address=127.0.0.1 --name=PI Server
$
```



If you are curious to see a proper output from Fledge, you can query the Core microservice using the REST API:

```
$ curl -s http://localhost:8081/fledge/ping ; echo
{"uptime": 10480, "dataRead": 0, "dataSent": 0, "dataPurged": 0,
  "authenticationOptional": true, "serviceName": "Fledge", "hostName": "fledge",
  "ipAddresses": ["x.x.x.x", "x:x:x:x:x:x:x"], "health": "green", "safeMode": false}
$
$ curl -s http://localhost:8081/fledge/statistics ; echo
[{"key": "BUFFERED", "description": "Readings currently in the Fledge buffer", "value": 0}, {"key": "DISCARDED", "description": "Readings discarded by the South Service before being placed in the buffer. This may be due to an error in the readings themselves.", "value": 0}, {"key": "PURGED", "description": "Readings removed from the buffer by the purge process", "value": 0}, {"key": "READINGS", "description": "Readings received by Fledge", "value": 0}, {"key": "UNSENT", "description": "Readings filtered out in the send process", "value": 0}, {"key": "UNSNPURGED", "description": "Readings that were purged from the buffer before being sent", "value": 0}]
$
```

Congratulations! You have installed and tested Fledge! If you want to go extra mile (and make the output of the REST API more readable, download the *jq* JSON processor and pipe the output of the *curl* command to it:

```
$ sudo apt install jq
...
$
$ curl -s http://localhost:8081/fledge/statistics | jq
[
  {
    "key": "BUFFERED",
    "description": "Readings currently in the Fledge buffer",
    "value": 0
  },
  {
    "key": "DISCARDED",
    "description": "Readings discarded by the South Service before being placed in the buffer. This may be due to an error in the readings themselves.",
    "value": 0
  },
  {
    "key": "PURGED",
    "description": "Readings removed from the buffer by the purge process",
    "value": 0
  },
  {
    "key": "READINGS",
    "description": "Readings received by Fledge",
    "value": 0
  },
  {
    "key": "UNSENT",
    "description": "Readings filtered out in the send process",
    "value": 0
  },
  {
    "key": "UNSNPURGED",
    "description": "Readings that were purged from the buffer before being sent",
    "value": 0
  }
]
```

(continues on next page)



(continued from previous page)

```
]
$
```

## Now I Want to Stop Fledge!

Easy, you have learnt `fledge start` and `fledge status`, simply type `fledge stop`:

```
$ scripts/fledge stop
Stopping Fledge.....
Fledge stopped.
$
```

As a next step, let's install Fledge!

## Appendix: Setting the PostgreSQL Database

If you intend to use the PostgreSQL database as storage engine, make sure that PostgreSQL is installed and running correctly:

```
$ sudo systemctl status postgresql
postgresql.service - PostgreSQL RDBMS
   Loaded: loaded (/lib/systemd/system/postgresql.service; enabled; vendor preset: _
   →enabled)
   Active: active (exited) since Fri 2017-12-08 15:56:07 GMT; 15min ago
   Main PID: 14572 (code=exited, status=0/SUCCESS)
   CGroup: /system.slice/postgresql.service

Dec 08 15:56:07 ubuntu systemd[1]: Starting PostgreSQL RDBMS...
Dec 08 15:56:07 ubuntu systemd[1]: Started PostgreSQL RDBMS.
Dec 08 15:56:11 ubuntu systemd[1]: Started PostgreSQL RDBMS.
$
$ ps -ef | grep postgres
postgres 14806      1  0 15:56 ?           00:00:00 /usr/lib/postgresql/9.5/bin/postgres -
   →D /var/lib/postgresql/9.5/main -c config_file=/etc/postgresql/9.5/main/postgresql.
   →conf
postgres 14808 14806  0 15:56 ?           00:00:00 postgres: checkpointer process
postgres 14809 14806  0 15:56 ?           00:00:00 postgres: writer process
postgres 14810 14806  0 15:56 ?           00:00:00 postgres: wal writer process
postgres 14811 14806  0 15:56 ?           00:00:00 postgres: autovacuum launcher process
postgres 14812 14806  0 15:56 ?           00:00:00 postgres: stats collector process
ubuntu   15198 1225  0 17:22 pts/0    00:00:00 grep --color=auto postgres
$
```

PostgreSQL 13 is the version available for Ubuntu 18.04 when we have published this page. Other versions of PostgreSQL, such as 9.6 to newer version work just fine. When you install the Ubuntu package, PostgreSQL is set for a *peer authentication*, i.e. the database user must match with the Linux user. Other packages may differ. You may quickly check the authentication mode set in the `pg_hba.conf` file. The file is in the same directory of the `postgresql.conf` file you may see as output from the `ps` command shown above, in our case `/etc/postgresql/9.5/main`:

```
$ sudo grep '^local' /etc/postgresql/9.5/main/pg_hba.conf
local    all                                postgres                                peer
```

(continues on next page)



(continued from previous page)

```
local    all          all          peer
$
```

The installation procedure also creates a Linux *postgres* user. In order to check if everything is set correctly, execute the *psql* utility as *sudo* user:

```
$ sudo -u postgres psql -l
```

List of databases						
Name	Owner	Encoding	Collate	Ctype	Access privileges	
postgres	postgres	UTF8	en_GB.UTF-8	en_GB.UTF-8		
template0	postgres	UTF8	en_GB.UTF-8	en_GB.UTF-8	=c/postgres	+
					postgres=CTc/postgres	
template1	postgres	UTF8	en_GB.UTF-8	en_GB.UTF-8	=c/postgres	+
					postgres=CTc/postgres	

```
(3 rows)
$
```

Encoding and collations may differ, depending on the choices made when you installed your operating system. Before you proceed, you must create a PostgreSQL user that matches your Linux user. Supposing that your user is *<fledge\_user>*, type:

```
$ sudo -u postgres createuser -d <fledge_user>
```

The *-d* argument is important because the user will need to create the Fledge database.

**A more generic command is:** `$ sudo -u postgres createuser -d $(whoami)`

Finally, you should now be able to see the list of the available databases from your current user:

```
$ psql -l
```

List of databases						
Name	Owner	Encoding	Collate	Ctype	Access privileges	
postgres	postgres	UTF8	en_GB.UTF-8	en_GB.UTF-8		
template0	postgres	UTF8	en_GB.UTF-8	en_GB.UTF-8	=c/postgres	+
					postgres=CTc/postgres	
template1	postgres	UTF8	en_GB.UTF-8	en_GB.UTF-8	=c/postgres	+
					postgres=CTc/postgres	

```
(3 rows)
$
```

### 15.1.3 Fledge Installation

Installing Fledge using defaults is straightforward: depending on the usage, you may install a new version from source or from a pre-built package. In environments where the defaults do not fit, you will need to execute few more steps. This chapter describes the default installation of Fledge and the most common scenarios where administrators need to modify the default behavior.



## Installing Fledge from a Build

Once you have built Fledge following the instructions presented, you can execute the default installation with the `make install` command. By default, Fledge is installed from build in the root directory, under `/usr/local/fledge`. Since the root directory `/` is a protected system location, you will need superuser privileges to execute the command. Therefore, if you are not superuser, you should login as superuser or you should use the `sudo` command.

```
$ sudo make install
mkdir -p /usr/local/fledge
Installing Fledge version 1.8.0, DB schema 2
-- Fledge DB schema check OK: Info: /usr/local/fledge is empty right now. Skipping DB
↳schema check.
cp VERSION /usr/local/fledge
cd cmake_build ; cmake /home/fledge/Fledge/
-- Boost version: 1.58.0
-- Found the following Boost libraries:
--   system
--   thread
--   chrono
--   date_time
--   atomic
-- Found SQLite version 3.11.0: /usr/lib/x86_64-linux-gnu/libsqlite3.so
-- Boost version: 1.58.0
-- Found the following Boost libraries:
--   system
--   thread
--   chrono
--   date_time
--   atomic
-- Configuring done
-- Generating done
-- Build files have been written to: /home/fledge/Fledge/cmake_build
cd cmake_build ; make
make[1]: Entering directory '/home/fledge/Fledge/cmake_build'
...
$
```

These are the main steps of the installation:

- Create the `/usr/local/fledge` directory, if it does not exist
- Build the code that has not been compiled and built yet
- Create all the necessary destination directories and copy the executables, scripts and configuration files
- Change the ownership of the `data` directory, if the install user is a superuser (we recommend to run Fledge as regular user, i.e. not as superuser).

Fledge is now present in `/usr/local/fledge` and ready to start. The start script is in the `/usr/local/fledge/bin` directory

```
$ cd /usr/local/fledge/
$ ls -l
total 32
drwxr-xr-x 2 root    root    4096 Apr 24 18:07 bin
drwxr-xr-x 4 fledge  fledge  4096 Apr 24 18:07 data
drwxr-xr-x 4 root    root    4096 Apr 24 18:07 extras
drwxr-xr-x 4 root    root    4096 Apr 24 18:07 plugins
drwxr-xr-x 3 root    root    4096 Apr 24 18:07 python
drwxr-xr-x 6 root    root    4096 Apr 24 18:07 scripts
```

(continues on next page)



(continued from previous page)

```

drwxr-xr-x 2 root    root    4096 Apr 24 18:07 services
-rwxr-xr-x 1 root    root      37 Apr 24 18:07 VERSION
$
$ bin/fledge
Usage: fledge {start|start --safe-mode|stop|status|reset|kill|help|version}
$
$ bin/fledge help
Usage: fledge {start|start --safe-mode|stop|status|reset|kill|help|version}
Fledge v1.3.1 admin script
The script is used to start Fledge
Arguments:
  start                - Start Fledge core (core will start other services).
  start --safe-mode    - Start in safe mode (only core and storage services will be
↳ started)
  stop                 - Stop all Fledge services and processes
  kill                 - Kill all Fledge services and processes
  status               - Show the status for the Fledge services
  reset                - Restore Fledge factory settings
                       WARNING! This command will destroy all your data!
  version              - Print Fledge version
  help                 - This text
$
$ bin/fledge start
Starting Fledge.....
Fledge started.
$

```

## Environment Variables

In order to operate, Fledge requires two environment variables:

- **FLEDGE\_ROOT**: the root directory for Fledge. The default is `/usr/local/fledge`
- **FLEDGE\_DATA**: the data directory. The default is `$FLEDGE_ROOT/data`, hence whichever value `FLEDGE_ROOT` has plus the `data` sub-directory, or `/usr/local/fledge/data` in case `FLEDGE_ROOT` is set as default value.

## The setenv.sh Script

In the `extras/scripts` folder of the newly installed Fledge you can find the `setenv.sh` script. This script can be used to set the environment variables used by Fledge and update your `PATH` environment variable. You can call the script from your shell or you can add the same command to your `.profile` script:

```

$ cat /usr/local/fledge/extras/scripts/setenv.sh
#!/bin/sh

##-----
## Copyright (c) 2018 OSIsoft, LLC
##
## Licensed under the Apache License, Version 2.0 (the "License");
## you may not use this file except in compliance with the License.
## You may obtain a copy of the License at
##
##      http://www.apache.org/licenses/LICENSE-2.0

```

(continues on next page)



(continued from previous page)

```
##
## Unless required by applicable law or agreed to in writing, software
## distributed under the License is distributed on an "AS IS" BASIS,
## WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
## See the License for the specific language governing permissions and
## limitations under the License.
##-----

#
# This script sets the user environment to facilitate the administration
# of Fledge
#
# You can execute this script from shell, using for example this command:
#
# source /usr/local/fledge/extras/scripts/setenv.sh
#
# or you can add the same command at the bottom of your profile script
# {HOME}/.profile.
#

export FLEDGE_ROOT="/usr/local/fledge"
export FLEDGE_DATA="${FLEDGE_ROOT}/data"

export PATH="${FLEDGE_ROOT}/bin:${PATH}"
export LD_LIBRARY_PATH="${FLEDGE_ROOT}/lib:${LD_LIBRARY_PATH}"

$ source /usr/local/fledge/extras/scripts/setenv.sh
$
```

## The fledge.service Script

Another file available in the *extras/scripts* folder is the *fledge.service* script. This script can be used to set Fledge as a Linux service. If you wish to do so, we recommend to install the Fledge package, but if you have a special build or for other reasons you prefer to work with Fledge built from source, this script will be quite helpful.

You can install Fledge as a service following these simple steps:

- After the `make install` command, copy *fledge.service* with a simple name *fledge* in the */etc/init.d* folder.
- Execute the command `systemctl enable fledge.service` to enable Fledge as a service
- Execute the command `systemctl start fledge.service` if you want to start Fledge

```
$ sudo cp /usr/local/fledge/extras/scripts/fledge.service /etc/init.d/fledge
$ sudo systemctl status fledge.service
fledge.service
   Loaded: not-found (Reason: No such file or directory)
   Active: inactive (dead)
$ sudo systemctl enable fledge.service
fledge.service is not a native service, redirecting to systemd-sysv-install
Executing /lib/systemd/systemd-sysv-install enable fledge
$ sudo systemctl status fledge.service
fledge.service - LSB: Fledge
   Loaded: loaded (/etc/init.d/fledge; bad; vendor preset: enabled)
   Active: inactive (dead)
   Docs: man:systemd-sysv-generator(8)
```

(continues on next page)



(continued from previous page)

```

$ sudo systemctl start fledge.service
$ sudo systemctl status fledge.service
fledge.service - LSB: Fledge
Loaded: loaded (/etc/init.d/fledge; generated)
Active: active (running) since Thu 2020-05-28 18:42:07 IST; 9min ago
  Docs: man:systemd-sysv-generator(8)
Process: 5047 ExecStart=/etc/init.d/fledge start (code=exited, status=0/SUCCESS)
Tasks: 27 (limit: 4680)
CGroup: /system.slice/fledge.service
        └─5123 python3 -m fledge.services.core
        └─5331 /usr/local/fledge/services/fledge.services.storage --address=0.0.0.0 -
↪-port=34827
        └─8119 /bin/sh tasks/north_c --port=34827 --address=127.0.0.1 --name=OMF to_
↪PI north
        └─8120 ./tasks/sending_process --port=34827 --address=127.0.0.1 --name=OMF_
↪to PI north
...
$

```

## Installing the Debian Package

We have versions of Fledge available as Debian packages for you. Check the to review which versions and platforms are available.

## Obtaining and Installing the Debian Package

Check the to find the package to install.

Once you have downloaded the package, install it using the `apt-get` command. You can use `apt-get` to install a local Debian package and automatically retrieve all the necessary packages that are defined as pre-requisites for Fledge. Note that you may need to install the package as superuser (or by using the `sudo` command) and move the package to the apt cache directory first (`/var/cache/apt/archives`).

For example, if you are installing Fledge on an Intel x86\_64 machine, you can type this command to download the package:

```

$ wget https://fledge-iot.s3.amazonaws.com/1.8.0/ubuntu1804/x86_64/fledge-1.8.0_x86_
↪64_ubuntu1804.tgz
--2020-05-28 18:24:12-- https://fledge-iot.s3.amazonaws.com/1.8.0/ubuntu1804/x86_64/
↪fledge-1.8.0_x86_64_ubuntu1804.tgz
Resolving fledge-iot.s3.amazonaws.com (fledge-iot.s3.amazonaws.com)... 52.217.40.188
Connecting to fledge-iot.s3.amazonaws.com (fledge-iot.s3.amazonaws.com)|52.217.40.
↪188|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 24638625 (23M) [application/x-tar]
Saving to: 'fledge-1.8.0_x86_64_ubuntu1804.tgz'

fledge-1.8.0_x86_64_ubuntu1804.tg 100
↪%[=====>] 23.50M 4.30MB/s
↪ in 8.3s

```

(continues on next page)



(continued from previous page)

```
2020-05-28 18:24:26 (2.84 MB/s) - 'fledge-1.8.0_x86_64_ubuntu1804.tgz' saved_
↪[24638625/24638625]
$
```

We recommend to execute an *update-upgrade-update* of the system first, then you may untar the fledge-1.8.0\_x86\_64\_ubuntu1804.tgz file and copy the Fledge package in the *apt cache* directory and install it.

```
$ sudo apt update
Hit:1 http://gb.archive.ubuntu.com/ubuntu xenial InRelease
...
$ sudo apt upgrade
...
$ sudo apt update
...
$ sudo cp fledge-1.8.0-x86_64.deb /var/cache/apt/archives/.
...
$ sudo apt install /var/cache/apt/archives/fledge-1.8.0-x86_64.deb
Reading package lists... Done
Building dependency tree
Reading state information... Done
Note, selecting 'fledge' instead of '/var/cache/apt/archives/fledge-1.8.0-x86_64.deb'
The following packages were automatically installed and are no longer required:
...
Unpacking fledge (1.8.0) ...
Setting up fledge (1.8.0) ...
Resolving data directory
Data directory does not exist. Using new data directory
Installing service script
Generating certificate files
Certificate files do not exist. Generating new certificate files.
Creating a self signed SSL certificate ...
Certificates created successfully, and placed in data/etc/certs
Generating auth certificate files
CA Certificate file does not exist. Generating new CA certificate file.
Creating ca SSL certificate ...
ca certificate created successfully, and placed in data/etc/certs
Admin Certificate file does not exist. Generating new admin certificate file.
Creating user SSL certificate ...
user certificate created successfully for admin, and placed in data/etc/certs
User Certificate file does not exist. Generating new user certificate file.
Creating user SSL certificate ...
user certificate created successfully for user, and placed in data/etc/certs
Setting ownership of Fledge files
Calling Fledge package update script
Linking update task
Changing setuid of update_task.apt
Removing task/update
Create link file
Copying sudoers file
Setting setuid bit of cmdutil
Enabling Fledge service
fledge.service is not a native service, redirecting to systemd-sysv-install.
Executing: /lib/systemd/systemd-sysv-install enable fledge
Starting Fledge service
$
```

As you can see from the output, the installation automatically registers Fledge as a service, so it will come up at startup



and it is already up and running when you complete the command.

Check the newly installed package:

```
$ sudo dpkg -l | grep fledge
ii fledge      1.8.0      amd64      Fledge, the open source platform for the
↳ Internet of Things
$
```

You can also check the service currently running:

```
$ sudo systemctl status fledge.service
fledge.service - LSB: Fledge
Loaded: loaded (/etc/init.d/fledge; generated)
Active: active (running) since Thu 2020-05-28 18:42:07 IST; 9min ago
Docs: man:systemd-sysv-generator(8)
Process: 5047 ExecStart=/etc/init.d/fledge start (code=exited, status=0/SUCCESS)
Tasks: 27 (limit: 4680)
CGroup: /system.slice/fledge.service
├─5123 python3 -m fledge.services.core
├─5331 /usr/local/fledge/services/fledge.services.storage --address=0.0.0.0 -
↳ -port=34827
├─8119 /bin/sh tasks/north_c --port=34827 --address=127.0.0.1 --name=OMF to_
↳ PI north
└─8120 ./tasks/sending_process --port=34827 --address=127.0.0.1 --name=OMF_
↳ to PI north
...
$
```

Check if Fledge is up and running with the fledge command:

```
$ /usr/local/fledge/bin/fledge status
Fledge v1.8.0 running.
Fledge Uptime: 162 seconds.
Fledge records: 0 read, 0 sent, 0 purged.
Fledge does not require authentication.
=== Fledge services:
fledge.services.core
...
=== Fledge tasks:
...
$
```

Don't forget to add the *setenv.sh* available in the */usr/local/fledge/extras/scripts\** directory to your *.profile* user startup script if you want to have an easy access to the Fledge tools, and...

... Congratulations! This is all you need to do, now Fledge is ready to run.



## Upgrading or Downgrading Fledge

Upgrading or downgrading Fledge, starting from version 1.2, is as easy as installing it from scratch: simply follow the instructions in the previous section regarding the installation and the package will take care of the upgrade/downgrade path. The installation will not proceed if there is not a path to upgrade or downgrade from the currently installed version. You should still check the pre-requisites before you apply the upgrade. Clearly the old data will not be lost, there will be a schema upgrade/downgrade, if required.

## Uninstalling the Debian Package

Use the `apt` or the `apt-get` command to uninstall Fledge:

```
$ sudo apt purge fledge
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  libmodbus-dev libmodbus5
Use 'sudo apt autoremove' to remove them.
The following packages will be REMOVED:
  fledge*
0 upgraded, 0 newly installed, 1 to remove and 0 not upgraded.
After this operation, 0 B of additional disk space will be used.
Do you want to continue? [Y/n] y
(Reading database ... 160251 files and directories currently installed.)
Removing fledge (1.8.0) ...
Fledge is currently running.
Stop Fledge service.
Kill Fledge.
Disable Fledge service.
fledge.service is not a native service, redirecting to systemd-sysv-install.
Executing: /lib/systemd/systemd-sysv-install disable fledge
Remove Fledge service script
Reset systemctl
Cleanup of files
Remove fledge sudoers file
(Reading database ... 159822 files and directories currently installed.)
Purging configuration files for fledge (1.8.0) ...
Cleanup of files
Remove fledge sudoers file
dpkg: warning: while removing fledge, directory '/usr/local/fledge' not empty so not
↳ removed
$
```

The command also removes the service installed. You may notice the warning in the last row of the command output: this is due to the fact that the data directory (`/usr/local/fledge/data` by default) has not been removed, in case an administrator might want to analyze or reuse the data.



### 15.1.4 Testing Fledge

After the installation, you are now ready to test Fledge. An end-to-end test involves three types of tests:

- The **South** side, i.e. testing the collection of information from South microservices and associated plugins
- The **North** side, i.e. testing the tasks that send data North to historians, databases, Enterprise and Cloud systems
- The **East/West** side, i.e. testing the interaction of external applications with Fledge via REST API.

This chapter describes how to tests Fledge in these three directions.

#### First Checks: Fledge Status

Before we start, let's make sure that Fledge is up and running and that we have the tasks and services in place to execute the tests. First, run the `fledge status` command to check if Fledge has already started. The result of the command can be:

- `Fledge not running.` - it means that we must start Fledge with `fledge start`
- `Fledge starting.` - it means that we have started Fledge but the starting phase has not been completed yet. You should wait for a little while (from few seconds to about a minute) to see Fledge running.
- `Fledge running.` - (plus extra rows giving the uptime and other info. It means that Fledge is up and running, hence it is ready for use.

When you have a running Fledge, check the extra information provided by the `fledge status` command:

```
$ fledge status
Fledge v1.8.0 running.
Fledge Uptime:  9065 seconds.
Fledge records: 86299 read, 86851 sent, 0 purged.
Fledge does not require authentication.
=== Fledge services:
fledge.services.core
fledge.services.storage --address=0.0.0.0 --port=42583
fledge.services.south --port=42583 --address=127.0.0.1 --name=Sine
fledge.services.notification --port=42583 --address=127.0.0.1 --name=Fledge_
↳Notifications
=== Fledge tasks:
fledge.tasks.purge --port=42583 --address=127.0.0.1 --name=purge
tasks/sending_process --port=42583 --address=127.0.0.1 --name=PI Server
$
```

Let's analyze the output of the command:

- `Fledge running.` - The Fledge Core microservice is running on this machine and it is responding to the status command as *running* because other basic microservices are also running.
- `Fledge uptime: 282 seconds.` - This is a simple uptime in second provided by the Core microservice. It is equivalent to the `ping` method called via the REST API.
- `Fledge records:` - This is a summary of the number of records received from sensors and devices (South), sent to other services (North) and purged from the buffer.
- `Fledge authentication` - This row describes if a user or an application must authenticate to ogLAMP in order to operate with the REST API.

The following lines provide a list of the modules running in this installation of Fledge. They are separated by dots and described in this way:

- The prefix `fledge` is always present and identifies the Fledge modules.



- The following term describes the type of module: *services* for microservices, *tasks* for tasks etc.
- The following term is the name of the module: *core*, *storage*, *north*, *south*, *app*, *alert*
- The last term is the name of the plugin executed as part of the module.
- Extra arguments may be available: they are the arguments passed to the module by the core when it is launched.
- `=== Fledge services:` - This block contains the list of microservices running in the Fledge platform.
  - `fledge.services.core` is the Core microservice itself
  - `fledge.services.south --port=44180 --address=127.0.0.1 --name=COAP` - This South microservice is a listener of data pushed to Fledge via a CoAP protocol
- `=== Fledge tasks:` - This block contains the list of tasks running in the Fledge platform.
  - `fledge.tasks.north.sending_process ... --name=sending process` is a North task that prepares and sends data collected by the South modules to the OSIsoft PI System in OMF (OSIsoft Message Format).
  - `fledge.tasks.north.sending_process ... --name=statistics to pi` is a North task that prepares and sends the internal statistics to the OSIsoft PI System in OMF (OSIsoft Message Format).

## Hello, Foggy World!

The output of the `fledge status` command gives you an idea of the modules running in your machine, but let's try to get more information from Fledge.

## The Fledge REST API

First of all, we need to familiarize with the Fledge REST API. The API provides a set of methods used to monitor and administer the status of Fledge. Users and developers can also use the API to interact with external applications.

This is a short list of the methods available to the administrators. A more detailed list will be available soon: - **ping** provides the uptime of the Fledge Core microservice - **statistics** provides a set of statistics of the Fledge platform, such as data collected, sent, purged, rejected etc. - **asset** provides a list of asset that have readings buffered in Fledge. - **category** provides a list of the configuration of the modules and components in Fledge.

## Useful Tools

Systems Administrators and Developers may already have their favorite tools to interact with a REST API, and they can probably use the same tools with Fledge. If you are not familiar with any tool, we recommend one of these:

- If you are familiar with the Linux shell and command lines, is the simplest and most useful tool available. It comes with every Linux distribution (or you can easily add it if it is not available in the default installation).
- If you prefer to use a browser-like interface, we recommend . Postman is an application available on Linux, MacOS and Windows and allows you to save queries, results, and run a set of queries with a single click.



### Hello World!

Let's execute the *ping* method. First, you must identify the IP address where Fledge is running. If you have installed Fledge on your local machine, you can use *localhost*. Alternatively, check the IP address of the machine where Fledge is installed.

---

**Note:** This version of Fledge does not have any security setup by default, therefore you may be able to access the entry point for the REST API by any external application, but there may be security setting on your operating environment that prevent access to specific ports from external applications. If you receive an error using the ping method, and the `fledge status` command says that everything is running, it is likely that you are experiencing a security issue.

---

The default port for the REST API is 8081. Using curl, try this command:

```
$ curl -s http://localhost:8081/fledge/ping ; echo
{"uptime": 10480, "dataRead": 0, "dataSent": 0, "dataPurged": 0,
↪ "authenticationOptional": true, "serviceName": "Fledge", "hostName": "fledge",
↪ "ipAddresses": ["x.x.x.x", "x:x:x:x:x:x:x:x"], "health": "green", "safeMode": false}
$
```

The `echo` at the end of the line is simply used to add an extra new line to the output.

If you are using Postman, select the *GET* method and type `http://localhost:8081/fledge/ping` in the URI address input. If you are accessing a remote machine, replace *localhost* with the correct IP address. The output should be something like:



This is the first message you may receive from Fledge!

### Hello from the Southern Hemisphere of the Fledge World

Let's now try something more exciting. The primary job of Fledge is to collect data from the Edge (we call it *South*), buffer it in our storage engine and then we send the data to Cloud historians and Enterprise Servers (we call them *North*). We also offer information to local or networked applications, something we call *East* or *West*.

In order to insert data you may need a sensor or a device that generates data. If you want to try Fledge but you do not have any sensor at hand, do not worry, we have a tool that can generate data as if it is a sensor.

### fogbench: a Brief Intro

Fledge comes with a little but pretty handy tool called **fogbench**. The tool is written in Python and it uses the same libraries of other modules of Fledge, therefore no extra libraries are needed. With *fogbench* you can do many things, like inserting data stored in files, running benchmarks to understand how Fledge performs in a given environment, or test an end-to-end installation.

Note: The following instructions assume you have downloaded and installed the CoAP south plugin from <https://github.com/fledge-iot/fledge-south-coap>.



```
$ git clone https://github.com/fledge-iot/fledge-south-coap
$ cd fledge-south-coap
$ sudo cp -r python/fledge/plugins/south/coap /usr/local/fledge/python/fledge/plugins/
  ↳ south/
$ sudo cp python/requirements-coap.txt /usr/local/fledge/python/
$ sudo pip3 install -r /usr/local/fledge/python/requirements-coap.txt
$ sudo chown -R root:root /usr/local/fledge/python/fledge/plugins/south/coap
$ curl -sX POST http://localhost:8081/fledge/service -d '{"name": "CoAP", "type":
  ↳ "south", "plugin": "coap", "enabled": true}'
```

Depending on your environment, you can call *fogbench* in one of those ways:

- In a development environment, use the script *scripts/extras/fogbench*, inside your project repository (remember to set the *FLEDGE\_ROOT* environment variable with the path to your project repository folder).
- In an environment deployed with `sudo make install`, use the script *bin/fogbench*.

You may call the *fogbench* tool like this:

```
$ /usr/local/fledge/bin/fogbench
>>> Make sure south CoAP plugin service is running & listening on specified host and
  ↳ port
usage: fogbench [-h] [-v] [-k {y,yes,n,no}] -t TEMPLATE [-o OUTPUT]
               [-I ITERATIONS] [-O OCCURRENCES] [-H HOST] [-P PORT]
               [-i INTERVAL] [-S {total}]
fogbench: error: the following arguments are required: -t/--template
$
```

...or more specifically, when you call invoke *fogbench* with the *-help* or *-h* argument:

```
$ /usr/local/fledge/bin/fogbench -h
>>> Make sure south CoAP plugin service is running & listening on specified host and
  ↳ port
usage: fogbench [-h] [-v] [-k {y,yes,n,no}] -t TEMPLATE [-o OUTPUT]
               [-I ITERATIONS] [-O OCCURRENCES] [-H HOST] [-P PORT]
               [-i INTERVAL] [-S {total}]

fogbench -- a Python script used to test Fledge (simulate payloads)

optional arguments:
  -h, --help                show this help message and exit
  -v, --version              show program's version number and exit
  -k {y,yes,n,no}, --keep {y,yes,n,no}
                           Do not delete the running sample (default: no)
  -t TEMPLATE, --template TEMPLATE
                           Set the template file, json extension
  -o OUTPUT, --output OUTPUT
                           Set the statistics output file
  -I ITERATIONS, --iterations ITERATIONS
                           The number of iterations of the test (default: 1)
  -O OCCURRENCES, --occurrences OCCURRENCES
                           The number of occurrences of the template (default: 1)
  -H HOST, --host HOST      CoAP server host address (default: localhost)
  -P PORT, --port PORT      The Fledge port. (default: 5683)
  -i INTERVAL, --interval INTERVAL
                           The interval in seconds for each iteration (default:
                           0)
  -S {total}, --statistics {total}
```

(continues on next page)



(continued from previous page)

```
The type of statistics to collect (default: total)
```

The initial version of fogbench is meant to test the sensor/device interface of Fledge using CoAP

```
$
```

In order to use *fogbench* you need a template file. The template is a set of JSON elements that are used to create a random set of values that simulate the data generated by one or more sensors. Fledge comes with a template file named *fogbench\_sensor\_coap.template.json*. The template is located here:

- In a development environment, look in *data/extras/fogbench* in the project repository folder.
- In an environment deployed using `sudo make install`, look in *\$FLEDGE\_DATA/extras/fogbench*.

The template file looks like this:

```
$ cat /usr/local/fledge/data/extras/fogbench/fogbench_sensor_coap.template.json
[
  { "name"          : "fogbench_luxometer",
    "sensor_values" : [ { "name": "lux", "type": "number", "min": 0, "max": 130000,
↳ "precision":3 } ] },
  { "name"          : "fogbench_pressure",
    "sensor_values" : [ { "name": "pressure", "type": "number", "min": 800.0, "max": 1100.0, "precision":1 } ] },
↳ "precision":1 } ] },
  { "name"          : "fogbench_humidity",
    "sensor_values" : [ { "name": "humidity", "type": "number", "min": 0.0, "max": 100.0 },
↳ "precision":1 } ] },
    { "name": "temperature", "type": "number", "min": 0.0, "max": 50.0 } ] },
↳ "precision":1 } ] },
  { "name"          : "fogbench_temperature",
    "sensor_values" : [ { "name": "object", "type": "number", "min": 0.0, "max": 50.0 },
↳ "precision":1 } ] },
    { "name": "ambient", "type": "number", "min": 0.0, "max": 50.0 } ] },
↳ "precision":1 } ] },
  { "name"          : "fogbench_accelerometer",
    "sensor_values" : [ { "name": "x", "type": "number", "min": -2.0, "max": 2.0 },
    { "name": "y", "type": "number", "min": -2.0, "max": 2.0 },
    { "name": "z", "type": "number", "min": -2.0, "max": 2.0 } ] },
↳ "precision":1 } ] },
  { "name"          : "fogbench_gyroscope",
    "sensor_values" : [ { "name": "x", "type": "number", "min": -255.0, "max": 255.0 },
↳ "precision":1 } ] },
    { "name": "y", "type": "number", "min": -255.0, "max": 255.0 },
↳ "precision":1 } ] },
    { "name": "z", "type": "number", "min": -255.0, "max": 255.0 },
↳ "precision":1 } ] },
  { "name"          : "fogbench_magnetometer",
    "sensor_values" : [ { "name": "x", "type": "number", "min": -255.0, "max": 255.0 },
↳ "precision":1 } ] },
    { "name": "y", "type": "number", "min": -255.0, "max": 255.0 },
↳ "precision":1 } ] },
    { "name": "z", "type": "number", "min": -255.0, "max": 255.0 },
↳ "precision":1 } ] },
  { "name"          : "fogbench_mouse",
    "sensor_values" : [ { "name": "button", "type": "enum", "list": [ "up", "down" ] } ] },
↳ "precision":1 } ] },
  { "name"          : "fogbench_switch",
```

(continues on next page)



(continued from previous page)

```

    "sensor_values" : [ { "name": "button", "type": "enum", "list": [ "up", "down" ] }
→ ] },
  { "name"           : "fogbench_wall clock",
    "sensor_values" : [ { "name": "tick", "type": "enum", "list": [ "tock" ] } ] }
]
$

```

In the array, each element simulates a message from a sensor, with a name, a set of data points that have their name, value type and range.

## Data Coming from South

Now you should have all the information necessary to test the CoAP South microservice. From the command line, type:

- `$FLEDGE_ROOT/scripts/extras/fogbench -t $FLEDGE_ROOT/data/extras/fogbench/fogbench_sensor_coap.template.json`, if you are in a development environment, with the *FLEDGE\_ROOT* environment variable set with the path to your project repository folder
- `$FLEDGE_ROOT/bin/fogbench -t $FLEDGE_DATA/extras/fogbench/fogbench_sensor_coap.template.json`, if you are in a deployed environment, with *FLEDGE\_ROOT* and *FLEDGE\_DATA* set correctly. - If you have installed Fledge in the default location (i.e. */usr/local/fledge*), type `cd /usr/local/fledge; bin/fogbench -t data/extras/fogbench/fogbench_sensor_coap.template.json`.
- `fledge.fogbench -t /snap/fledge/current/usr/local/fledge/data/extras/fogbench/fogbench_sensor_coap.template.json`, if you have installed a snap version of Fledge.

In development environment the output of your command should be:

```

$ $FLEDGE_ROOT/scripts/extras/fogbench -t data/extras/fogbench/fogbench_sensor_coap.
→template.json
>>> Make sure south CoAP plugin service is running & listening on specified host and
→port
Total Statistics:

Start Time: 2017-12-17 07:17:50.615433
End Time:   2017-12-17 07:17:50.650620

Total Messages Transferred: 10
Total Bytes Transferred:    2880

Total Iterations: 1
Total Messages per Iteration: 10.0
Total Bytes per Iteration:    2880.0

Min messages/second: 284.19586779208225
Max messages/second: 284.19586779208225
Avg messages/second: 284.19586779208225

Min Bytes/second: 81848.4099241197
Max Bytes/second: 81848.4099241197
Avg Bytes/second: 81848.4099241197
$

```



Congratulations! You have just inserted data into Fledge from the CoAP South microservice. More specifically, the output informs you that the data inserted has been composed by 10 different messages for a total of 2,880 Bytes, for an average of 284 messages per second and 81,848 Bytes per second.

If you want to stress Fledge a bit, you may insert the same data sample several times, by using the *-I* or *-iterations* argument:

```
$ $FLEDGE_ROOT/scripts/extras/fogbench -t data/extras/fogbench/fogbench_sensor_coap.  
↳template.json -I 100  
>>> Make sure south CoAP plugin service is running & listening on specified host and_  
↳port  
Total Statistics:  
  
Start Time: 2017-12-17 07:33:40.568130  
End Time: 2017-12-17 07:33:43.205626  
  
Total Messages Transferred: 1000  
Total Bytes Transferred: 288000  
  
Total Iterations: 100  
Total Messages per Iteration: 10.0  
Total Bytes per Iteration: 2880.0  
  
Min messages/second: 98.3032852957946  
Max messages/second: 625.860558267618  
Avg messages/second: 455.15247432732866  
  
Min Bytes/second: 28311.346165188843  
Max Bytes/second: 180247.840781074  
Avg Bytes/second: 131083.9126062706  
$
```

Here we have inserted the same set of data 100 times, therefore the total number of Bytes inserted is 288,000. The performance and insertion rates varies with each iteration and *fogbench* presents the minimum, maximum and average values.

### Checking What's Inside Fledge

We can check if Fledge has now stored what we have inserted from the South microservice by using the *asset* API. From curl or Postman, use this URL:

```
$ curl -s http://localhost:8081/fledge/asset ; echo  
[{"assetCode": "fogbench_switch", "count": 11}, {"assetCode": "fogbench_temperature",  
↳"count": 11}, {"assetCode": "fogbench_humidity", "count": 11}, {"assetCode":  
↳"fogbench_luxometer", "count": 11}, {"assetCode": "fogbench_accelerometer", "count  
↳": 11}, {"assetCode": "wall clock", "count": 11}, {"assetCode": "fogbench_  
↳magnetometer", "count": 11}, {"assetCode": "mouse", "count": 11}, {"assetCode":  
↳"fogbench_pressure", "count": 11}, {"assetCode": "fogbench_gyroscope", "count": 11}]  
$
```

The output of the asset entry point provides a list of assets buffered in Fledge and the count of elements stored. The output is a JSON array with two elements:

- **assetCode** : the name of the sensor or device that provides the data
- **count** : the number of occurrences of the asset in the buffer



## Feeding East/West Applications

Let's suppose that we are interested in the data collected for one of the assets listed in the previous query, for example *fogbench\_temperature*. The *asset* entry point can be used to retrieve the data points for individual assets by simply adding the code of the asset to the URI:

```
$ curl -s http://localhost:8081/fledge/asset/fogbench_temperature ; echo
[{"timestamp": "2017-12-18 10:38:29.652", "reading": {"ambient": 13, "object": 41}}, {
↪ "timestamp": "2017-12-18 10:38:29.652", "reading": {"ambient": 13, "object": 41}}, {
↪ "timestamp": "2017-12-18 10:38:29.652", "reading": {"ambient": 13, "object": 41}}, {
↪ "timestamp": "2017-12-18 10:38:29.652", "reading": {"ambient": 13, "object": 41}}, {
↪ "timestamp": "2017-12-18 10:38:29.652", "reading": {"ambient": 13, "object": 41}}, {
↪ "timestamp": "2017-12-18 10:38:29.652", "reading": {"ambient": 13, "object": 41}}, {
↪ "timestamp": "2017-12-18 10:38:29.652", "reading": {"ambient": 13, "object": 41}}, {
↪ "timestamp": "2017-12-18 10:38:29.652", "reading": {"ambient": 13, "object": 41}}, {
↪ "timestamp": "2017-12-18 10:38:29.652", "reading": {"ambient": 13, "object": 41}}, {
↪ "timestamp": "2017-12-18 10:38:29.652", "reading": {"ambient": 13, "object": 41}}, {
↪ "timestamp": "2017-12-18 10:38:12.580", "reading": {"ambient": 33, "object": 7}}]
```

Let's see the JSON output on a more readable format:

```
[ { "timestamp": "2017-12-18 10:38:29.652", "reading": { "ambient": 13, "object": 41 } }
↪ ,
  { "timestamp": "2017-12-18 10:38:29.652", "reading": { "ambient": 13, "object": 41 } }
↪ ,
  { "timestamp": "2017-12-18 10:38:29.652", "reading": { "ambient": 13, "object": 41 } }
↪ ,
  { "timestamp": "2017-12-18 10:38:29.652", "reading": { "ambient": 13, "object": 41 } }
↪ ,
  { "timestamp": "2017-12-18 10:38:29.652", "reading": { "ambient": 13, "object": 41 } }
↪ ,
  { "timestamp": "2017-12-18 10:38:29.652", "reading": { "ambient": 13, "object": 41 } }
↪ ,
  { "timestamp": "2017-12-18 10:38:29.652", "reading": { "ambient": 13, "object": 41 } }
↪ ,
  { "timestamp": "2017-12-18 10:38:29.652", "reading": { "ambient": 13, "object": 41 } }
↪ ,
  { "timestamp": "2017-12-18 10:38:29.652", "reading": { "ambient": 13, "object": 41 } }
↪ ,
  { "timestamp": "2017-12-18 10:38:12.580", "reading": { "ambient": 33, "object": 7 } } ]
```

The JSON structure depends on the sensor and the plugin used to capture the data. In this case, the values shown are:

- **timestamp** : the timestamp generated by the sensors. In this case, since we have inserted 10 times the same value and one time a new value using *fogbench*, the result is 10 timestamps with the same value and one timestamp with a different value.
- **reading** : a JSON structure that is the set of data points provided by the sensor. In this case:
- **ambient** : the ambient temperature in Celsius
- **object** : the object temperature in Celsius. Again, the values are repeated 10 times, due to the iteration executed by *fogbench*, plus an isolated element, so there are 11 readings in total. Also, it is very unlikely that in a real sensor the ambient and the object temperature differ so much, but here we are using a random number generator.



You can dig even more in the data and extract only a subset of the reading. Fog example, you can select the ambient temperature and limit to the last 5 readings:

```
$ curl -s http://localhost:8081/fledge/asset/fogbench_temperature/ambient?limit=5 ;  
↪echo  
[ { "ambient": 13, "timestamp": "2017-12-18 10:38:29.652" },  
  { "ambient": 13, "timestamp": "2017-12-18 10:38:29.652" },  
  { "ambient": 13, "timestamp": "2017-12-18 10:38:29.652" },  
  { "ambient": 13, "timestamp": "2017-12-18 10:38:29.652" },  
  { "ambient": 13, "timestamp": "2017-12-18 10:38:29.652" } ]  
$
```

We have beautified the JSON output for you, so it is more readable.

---

**Note:** When you select a specific element in the reading, the timestamp and the element are presented in the opposite order compared to the previous example. This is a known issue that will be fixed in the next version.

---

## Sending Greetings to the Northern Hemisphere

The next and last step is to send data to North, which means that we can take all of some of the data we buffer in Fledge and we can send it to a historian or a database using a North task or microservice.

### The OMF Translator

Fledge comes with a North plugin called *OMF Translator*. OMF is the OSIsoft Message Format, which is the message format accepted by the PI Connector Relay OMF. The PI Connector Relay OMF is provided by OSIsoft and it is used to feed the OSIsoft PI System.

- Information regarding OSIsoft are available
- Information regarding OMF are available
- Information regarding the OSIsoft PI System are available

*OMF Translator* is scheduled as a North task that is executed every 30 seconds (the time may vary, we set it to 30 seconds to facilitate the testing).

## Preparing the PI System

In order to test the North task and plugin, first you need to setup the PI system. Here we assume you are already familiar with PI and you have a Windows server with PI installed, up and running. The minimum installation must include the PI System and the PI Connector Relay OMF. Once you have checked that everything is installed and works correctly, you should collect the IP address of the Windows system.



## Setting the OMF Translator Plugin

Fledge uses the same *OMF Translator* plugin to send the data coming from the South modules and buffered in Fledge.

**Note:** In this version, only the South data can be sent to the PI System.

If you are curious to see which categories are available in Fledge, simply type:

```
$ curl -s http://localhost:8081/fledge/category ; echo
{
  "categories":
  [
    {
      "key": "SCHEDULER",
      "description": "Scheduler configuration",
      "displayName": "Scheduler"
    },
    {
      "key": "SMNTR",
      "description": "Service Monitor",
      "displayName": "Service Monitor"
    },
    {
      "key": "rest_api",
      "description": "Fledge Admin and User REST API",
      "displayName": "Admin API"
    },
    {
      "key": "service",
      "description": "Fledge Service",
      "displayName": "Fledge Service"
    },
    {
      "key": "Installation",
      "description": "Installation",
      "displayName": "Installation"
    },
    {
      "key": "General",
      "description": "General",
      "displayName": "General"
    },
    {
      "key": "Advanced",
      "description": "Advanced",
      "displayName": "Advanced"
    },
    {
      "key": "Utilities",
      "description": "Utilities",
      "displayName": "Utilities"
    }
  ]
}
```

For each plugin, you will see corresponding category e.g. For fledge-south-coap the registered category will be



{ "key": "COAP", "description": "CoAP Listener South Plugin"}. The configuration for the OMF Translator used to stream the South data is initially disabled, all you can see about the settings is:

```
$ curl -sX GET http://localhost:8081/fledge/category/OMF%20to%20PI%20north
{
  "enable": {
    "description": "A switch that can be used to enable or disable execution of the
    ↪sending process.",
    "type": "boolean",
    "readonly": "true",
    "default": "true",
    "value": "true"
  },
  "streamId": {
    "description": "Identifies the specific stream to handle and the related
    ↪information, among them the ID of the last object streamed.",
    "type": "integer",
    "readonly": "true",
    "default": "0",
    "value": "4",
    "order": "16"
  },
  "plugin": {
    "description": "PI Server North C Plugin",
    "type": "string",
    "default": "OMF",
    "readonly": "true",
    "value": "OMF"
  },
  "source": {
    "description": "Defines the source of the data to be sent on the stream, this
    ↪may be one of either readings, statistics or audit.",
    "type": "enumeration",
    "options": [
      "readings",
      "statistics"
    ],
    "default": "readings",
    "order": "5",
    "displayName": "Data Source",
    "value": "readings"
  },
  ...}
$ curl -sX GET http://localhost:8081/fledge/category/Stats%20OMF%20to%20PI%20north
{
  "enable": {
    "description": "A switch that can be used to enable or disable execution of the
    ↪sending process.",
    "type": "boolean",
    "readonly": "true",
    "default": "true",
    "value": "true"
  },
  "streamId": {
    "description": "Identifies the specific stream to handle and the related
    ↪information, among them the ID of the last object streamed.",
    "type": "integer",
    "readonly": "true",
```

(continues on next page)



(continued from previous page)

```

    "default": "0",
    "value": "5",
    "order": "16"
  },
  "plugin": {
    "description": "PI Server North C Plugin",
    "type": "string",
    "default": "OMF",
    "readonly": "true",
    "value": "OMF"
  },
  "source": {
    "description": "Defines the source of the data to be sent on the stream, this may
    ↳ be one of either readings, statistics or audit.",
    "type": "enumeration",
    "options": [
      "readings",
      "statistics"
    ],
    "default": "readings",
    "order": "5",
    "displayName": "Data Source",
    "value": "statistics"
  },
  ...}
$

```

At this point it may be a good idea to familiarize with the tool, it will help you a lot in selecting and using data via the REST API. You may remember, we discussed it in the chapter.

First, we can see the list of all the scheduled tasks (the process of sending data to a PI Connector Relay OMF is one of them). The command is:

```

$ curl -s http://localhost:8081/fledge/schedule | jq
{
  "schedules": [
    {
      "id": "ef8bd42b-da9f-47c4-ade8-751ce9a504be",
      "name": "OMF to PI north",
      "processName": "north_c",
      "type": "INTERVAL",
      "repeat": 30.0,
      "time": 0,
      "day": null,
      "exclusive": true,
      "enabled": false
    },
    {
      "id": "27501b35-e0cd-4340-afc2-a4465fe877d6",
      "name": "Stats OMF to PI north",
      "processName": "north_c",
      "type": "INTERVAL",
      "repeat": 30.0,
      "time": 0,
      "day": null,
      "exclusive": true,
      "enabled": true
    }
  ]
}

```

(continues on next page)



(continued from previous page)

```

    },
    ...
  ]
}
$

```

... which means: “show me all the tasks that can be scheduled”, The output has been made more readable by jq. There are several tasks, we need to identify the one we need and extract its unique id. We can achieve that with the power of jq: first we can select the JSON object that shows the elements of the sending task:

```

$ curl -s http://localhost:8081/fledge/schedule | jq '.schedules[] | select( .name ==
↳ "OMF to PI north" )'
{
  "id": "ef8bd42b-da9f-47c4-ade8-751ce9a504be",
  "name": "OMF to PI north",
  "processName": "north_c",
  "type": "INTERVAL",
  "repeat": 30,
  "time": 0,
  "day": null,
  "exclusive": true,
  "enabled": true
}
$

```

Let’s have a look at what we have found:

- **id** is the unique identifier of the schedule.
- **name** is a user-friendly name of the schedule.
- **type** is the type of schedule, in this case a schedule that is triggered at regular intervals.
- **repeat** specifies the interval of 30 seconds.
- **time** specifies when the schedule should run: since the type is INTERVAL, this element is irrelevant.
- **day** indicates the day of the week the schedule should run, in this case it will be constantly every 30 seconds
- **exclusive** indicates that only a single instance of this task should run at any time.
- **processName** is the name of the task to be executed.
- **enabled** indicates whether the schedule is currently enabled or disabled.

Now let’s identify the plugin used to send data to the PI Connector Relay OMF.

```

$ curl -s http://localhost:8081/fledge/category | jq '.categories[] | select ( .key_
↳ == "OMF to PI north" )'
{
  "key": "OMF to PI north",
  "description": "Configuration of the Sending Process",
  "displayName": "OMF to PI north"
}
$

```

We can get the specific information adding the name of the task to the URL:

```

$ curl -s http://localhost:8081/fledge/category/OMF%20to%20PI%20north | jq .plugin
{

```

(continues on next page)



(continued from previous page)

```

"description": "PI Server North C Plugin",
"type": "string",
"default": "OMF",
"readonly": "true",
"value": "OMF"
}
$

```

Now, the output returned does not say much: this is because the plugin has never been enabled, so the configuration has not been loaded yet. First, let's enable the schedule. From a previous query of the schedulable tasks, we know the id is *ef8bd42b-da9f-47c4-ade8-751ce9a504be*:

```

$ curl -X PUT http://localhost:8081/fledge/schedule/ef8bd42b-da9f-47c4-ade8-
↪751ce9a504be -d '{ "enabled" : true }'
{
  "schedule": {
    "id": "ef8bd42b-da9f-47c4-ade8-751ce9a504be",
    "name": "OMF to PI north",
    "processName": "north_c",
    "type": "INTERVAL",
    "repeat": 30,
    "time": 0,
    "day": null,
    "exclusive": true,
    "enabled": true
  }
}
$

```

Once enabled, the plugin will be executed inside the *OMF to PI north* task within 30 seconds, so you have to wait up to 30 seconds to see the new, full configuration. After 30 seconds or so, you should see something like this:

```

$ curl -s http://localhost:8081/fledge/category/OMF%20to%20PI%20north | jq
{
  "enable": {
    "description": "A switch that can be used to enable or disable execution of the_
↪sending process.",
    "type": "boolean",
    "readonly": "true",
    "default": "true",
    "value": "true"
  },
  "streamId": {
    "description": "Identifies the specific stream to handle and the related_
↪information, among them the ID of the last object streamed.",
    "type": "integer",
    "readonly": "true",
    "default": "0",
    "value": "4",
    "order": "16"
  },
  "plugin": {
    "description": "PI Server North C Plugin",
    "type": "string",
    "default": "OMF",
    "readonly": "true",

```

(continues on next page)



(continued from previous page)

```

    "value": "OMF"
  },
  "PIServerEndpoint": {
    "description": "Select the endpoint among PI Web API, Connector Relay, OSIssoft_
↪Cloud Services or Edge Data Store",
    "type": "enumeration",
    "options": [
      "PI Web API",
      "Connector Relay",
      "OSIssoft Cloud Services",
      "Edge Data Store"
    ],
    "default": "Connector Relay",
    "order": "1",
    "displayName": "Endpoint",
    "value": "Connector Relay"
  },
  "ServerHostname": {
    "description": "Hostname of the server running the endpoint either PI Web API or_
↪Connector Relay",
    "type": "string",
    "default": "localhost",
    "order": "2",
    "displayName": "Server hostname",
    "validity": "PIServerEndpoint != \"Edge Data Store\" && PIServerEndpoint != \"
↪OSIssoft Cloud Services\"",
    "value": "localhost"
  },
  "ServerPort": {
    "description": "Port on which the endpoint either PI Web API or Connector Relay_
↪or Edge Data Store is listening, 0 will use the default one",
    "type": "integer",
    "default": "0",
    "order": "3",
    "displayName": "Server port, 0=use the default",
    "validity": "PIServerEndpoint != \"OSIssoft Cloud Services\"",
    "value": "0"
  },
  "producerToken": {
    "description": "The producer token that represents this Fledge stream",
    "type": "string",
    "default": "omf_north_0001",
    "order": "4",
    "displayName": "Producer Token",
    "validity": "PIServerEndpoint == \"Connector Relay\"",
    "value": "omf_north_0001"
  },
  "source": {
    "description": "Defines the source of the data to be sent on the stream, this may_
↪be one of either readings, statistics or audit.",
    "type": "enumeration",
    "options": [
      "readings",
      "statistics"
    ],
    "default": "readings",
    "order": "5",

```

(continues on next page)



(continued from previous page)

```

    "displayName": "Data Source",
    "value": "readings"
  },
  "StaticData": {
    "description": "Static data to include in each sensor reading sent to the PI_
↪Server.",
    "type": "string",
    "default": "Location: Palo Alto, Company: Dianomic",
    "order": "6",
    "displayName": "Static Data",
    "value": "Location: Palo Alto, Company: Dianomic"
  },
  "OMFRetrySleepTime": {
    "description": "Seconds between each retry for the communication with the OMF PI_
↪Connector Relay, NOTE : the time is doubled at each attempt.",
    "type": "integer",
    "default": "1",
    "order": "7",
    "displayName": "Sleep Time Retry",
    "value": "1"
  },
  "OMFMaxRetry": {
    "description": "Max number of retries for the communication with the_
↪OMF PI Connector Relay",
    "type": "integer",
    "default": "3",
    "order": "8",
    "displayName": "Maximum Retry",
    "value": "3"
  },
  "OMFHttpTimeout": {
    "description": "Timeout in seconds for the HTTP operations with the OMF PI_
↪Connector Relay",
    "type": "integer",
    "default": "10",
    "order": "9",
    "displayName": "HTTP Timeout",
    "value": "10"
  },
  "formatInteger": {
    "description": "OMF format property to apply to the type Integer",
    "type": "string",
    "default": "int64",
    "order": "10",
    "displayName": "Integer Format",
    "value": "int64"
  },
  "formatNumber": {
    "description": "OMF format property to apply to the type Number",
    "type": "string",
    "default": "float64",
    "order": "11",
    "displayName": "Number Format",
    "value": "float64"
  },
  "compression": {
    "description": "Compress readings data before sending to PI server",

```

(continues on next page)



(continued from previous page)

```

    "type": "boolean",
    "default": "true",
    "order": "12",
    "displayName": "Compression",
    "value": "false"
  },
  "DefaultAFLocation": {
    "description": "Defines the hierarchies tree in Asset Framework in which the
    ↪ assets will be created, each level is separated by /, PI Web API only.",
    "type": "string",
    "default": "/fledge/data_piwebapi/default",
    "order": "13",
    "displayName": "Asset Framework hierarchies tree",
    "validity": "PIServerEndpoint == \"PI Web API\"",
    "value": "/fledge/data_piwebapi/default"
  },
  "AFMap": {
    "description": "Defines a set of rules to address where assets should be placed
    ↪ in the AF hierarchy.",
    "type": "JSON",
    "default": "{ }",
    "order": "14",
    "displayName": "Asset Framework hierarchies rules",
    "validity": "PIServerEndpoint == \"PI Web API\"",
    "value": "{ }"
  },
  "notBlockingErrors": {
    "description": "These errors are considered not blocking in the communication
    ↪ with the PI Server, the sending operation will proceed with the next block of data
    ↪ if one of these is encountered",
    "type": "JSON",
    "default": "{ \"errors400\" : [ \"Redefinition of the type with the same ID is
    ↪ not allowed\", \"Invalid value type for the property\", \"Property does not exist
    ↪ in the type definition\", \"Container is not defined\", \"Unable to find the
    ↪ property of the container of type\" ] }",
    "order": "15",
    "readonly": "true",
    "value": "{ \"errors400\" : [ \"Redefinition of the type with the same ID is not
    ↪ allowed\", \"Invalid value type for the property\", \"Property does not exist in
    ↪ the type definition\", \"Container is not defined\", \"Unable to find the property
    ↪ of the container of type\" ] }"
  },
  "PIWebAPIAuthenticationMethod": {
    "description": "Defines the authentication method to be used with the PI Web API.
    ↪ ",
    "type": "enumeration",
    "options": [
      "anonymous",
      "basic",
      "kerberos"
    ],
    "default": "anonymous",
    "order": "17",
    "displayName": "PI Web API Authentication Method",
    "validity": "PIServerEndpoint == \"PI Web API\"",
    "value": "anonymous"
  },

```

(continues on next page)



(continued from previous page)

```

"PIWebAPIUserId": {
  "description": "User id of PI Web API to be used with the basic access_
↪authentication.",
  "type": "string",
  "default": "user_id",
  "order": "18",
  "displayName": "PI Web API User Id",
  "validity": "PIServerEndpoint == \"PI Web API\" && PIWebAPIAuthenticationMethod_
↪== \"basic\"",
  "value": "user_id"
},
"PIWebAPIPassword": {
  "description": "Password of the user of PI Web API to be used with the basic_
↪access authentication.",
  "type": "password",
  "default": "password",
  "order": "19",
  "displayName": "PI Web API Password",
  "validity": "PIServerEndpoint == \"PI Web API\" && PIWebAPIAuthenticationMethod_
↪== \"basic\"",
  "value": "****"
},
"PIWebAPIKerberosKeytabFileName": {
  "description": "Keytab file name used for Kerberos authentication in PI Web API.",
  "type": "string",
  "default": "piwebapi_kerberos_https.keytab",
  "order": "20",
  "displayName": "PI Web API Kerberos keytab file",
  "validity": "PIServerEndpoint == \"PI Web API\" && PIWebAPIAuthenticationMethod_
↪== \"kerberos\"",
  "value": "piwebapi_kerberos_https.keytab"
},
"OCSNamespace": {
  "description": "Specifies the OCS namespace where the information are stored and_
↪it is used for the interaction with the OCS API",
  "type": "string",
  "default": "name_space",
  "order": "21",
  "displayName": "OCS Namespace",
  "validity": "PIServerEndpoint == \"OSIsoft Cloud Services\"",
  "value": "name_space"
},
"OCSTenantId": {
  "description": "Tenant id associated to the specific OCS account",
  "type": "string",
  "default": "ocs_tenant_id",
  "order": "22",
  "displayName": "OCS Tenant ID",
  "validity": "PIServerEndpoint == \"OSIsoft Cloud Services\"",
  "value": "ocs_tenant_id"
},
"OCSClientId": {
  "description": "Client id associated to the specific OCS account, it is used to_
↪authenticate the source for using the OCS API",
  "type": "string",
  "default": "ocs_client_id",
  "order": "23",

```

(continues on next page)



(continued from previous page)

```

    "displayName": "OCS Client ID",
    "validity": "PIServerEndpoint == \"OSIsoft Cloud Services\"",
    "value": "ocs_client_id"
  },
  "OCSCClientSecret": {
    "description": "Client secret associated to the specific OCS account, it is used_
↪to authenticate the source for using the OCS API",
    "type": "password",
    "default": "ocs_client_secret",
    "order": "24",
    "displayName": "OCS Client Secret",
    "validity": "PIServerEndpoint == \"OSIsoft Cloud Services\"",
    "value": "****"
  }
}
$

```

You can look at the descriptions to have a taste of what you can control with this plugin. The default configuration should be fine, with the exception of the *ServerHostname*, which of course should refer to the IP address of the machine and the port used by the PI Connector Relay OMF. The PI Connector Relay OMF 1.0 used the HTTP protocol with port 8118 and version 1.2, or higher, uses the HTTPS and port 5460. Assuming that the port is 5460 and the IP address is 192.168.56.101, you can set the new *ServerHostname* with this PUT method:

```

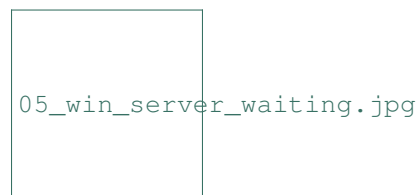
$ curl -sH'Content-Type: application/json' -X PUT -d '{ "ServerHostname": "192.168.56.
↪101" }' http://localhost:8081/fledge/category/OMF%20to%20PI%20north | jq
"ServerHostname": {
  "description": "Hostname of the server running the endpoint either PI Web API or_
↪Connector Relay",
  "type": "string",
  "default": "localhost",
  "order": "2",
  "displayName": "Server hostname",
  "validity": "PIServerEndpoint != \"Edge Data Store\" && PIServerEndpoint != \"
↪OSIsoft Cloud Services\"",
  "value": "192.168.56.101"
}
$

```

You can note that the *value* element is the only one that can be changed in *URL* (the other elements are factory settings). Now we are ready to send data North, to the PI System.

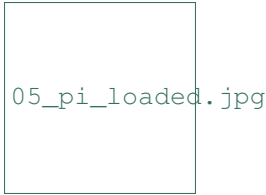
## Sending Data to the PI System

The last bit to accomplish is to start the PI Connector Relay OMF on the Windows Server. The output may look like this screenshot, where you can see the Connector Relay debug window on the left and teh PI Data Explorer on the right.



Wait a few seconds ... et voilà! Readings and statistics are in the PI System:





Congratulations! You have experienced an end-to-end test of Fledge, from South with sensor data through Fledge and East/West applications and finally to North towards Historians.

### 15.1.5 Fledge Utilities and Scripts

The Fledge platform comes with a set of utilities and scripts to help users, developers and administrators with their day-by-day operations. These tools are under heavy development and you may expect incompatibilities in future versions, therefore it is highly recommended to check the revision history to verify the changes in new versions.

#### fledge

`fledge` is the first utility available with the platform, it is the control center for all the admin operations on Fledge.

In the current implementation, *fledge* provides these features:

- *start* Fledge
- *stop* Fledge
- *kill* Fledge processes
- Check the *status* of Fledge, i.e. whether it is running, starting or not running
- *reset* Fledge to its factory settings

#### Starting Fledge

`fledge start` is the command to start Fledge. Since only one core microservice of Fledge can be executed in the same environment, the command checks if Fledge is already running, and if it does, it ends. The command also checks the presence of the `FLEDGE_ROOT` and `FLEDGE_DATA` environment variables. If the variables have not been set, it verifies if Fledge has been installed in the default position, which is `/usr/local/fledge` or a position defined by the installed package, and it will set the missing variables accordingly. It will also take care of the `PYTHONPATH` variable.

In more specific terms, the command executes these steps:

- Check if Fledge is already running
- Check if the storage layer is *managed* or *unmanaged*. “managed” means that the storage layer relies on a storage system (i.e. a database, a set of files or in-memory structures) that are under exclusive control of Fledge. “unmanaged” means that the storage system is generic and potentially shared with other applications.
- Check if the storage plugin and the related storage system (for example a PostgreSQL database) is available.
- Check if the metadata structure that is necessary to execute Fledge is already available in the storage layer. If the metadata is not available, it creates the data model and sets the factory settings that are necessary to start and use Fledge.
- Start the core microservice.



- Wait until the core microservice starts the Storage microservice and the initial required process that are necessary to handle other tasks and microservices.

### Safe Mode

It is possible to start Fledge in safe mode by passing the flag `--safe-mode` to the start command. In safe mode Fledge will not start any of the south services or schedule any tasks, such as purge or north bound tasks. Safe mode allows Fledge to be started and configured in those situations where a previous misconfiguration has rendered it impossible to start and interact with Fledge.

Once started in safe mode any configuration changes should be made and then Fledge should be restarted in normal mode to test those configuration changes.

### Stopping Fledge

`fledge stop` is the command used to stop Fledge. The command waits until all the tasks and services have been completed, then it stops the core service.

### If Fledge Does Not Stop

If Fledge does not stop, i.e. if by using the process status command `ps` you see Fledge processes still running, you can use `fledge kill` to kill them.

---

**Note:** The command issues a `kill -9` against the processes associated to Fledge. This is not recommended, unless Fledge cannot be stopped. The `stop` command. In other words, *kill* is your last resort before a reboot. If you must use the kill command, it means that there is a problem: please report this to the Fledge project slack channel.

---

### Checking the Status of Fledge

`fledge status` is used to provide the current status of tasks and microservices on the machine. The output is something like:

```
$ fledge status
Fledge running.
Fledge uptime: 2034 seconds.
=== Fledge services:
fledge.services.core
fledge.services.south --port=33074 --address=127.0.0.1 --name=HTTP_SOUTH
fledge.services.south --port=33074 --address=127.0.0.1 --name=COAP
=== Fledge tasks:
$ fledge_use_from_here stop
Fledge stopped.
$ fledge_use_from_here status
Fledge not running.
$
```

- The first row always indicates if Fledge is running or not
- The second row provides the uptime in seconds
- The next set of rows provides information regarding the microservices running on the machine



- The last set of rows provides information regarding the tasks running on the machine

## Resetting Fledge

It may occur that you want to restore Fledge to its factory settings, and this is what `fledge reset` does. The command also destroys all the data and all the configuration currently stored in Fledge, so you must use it at your own risk!

Fledge can be restored to its factory settings only when it is not running, hence you should stop it first.

The command forces you to insert the word *YES*, all in uppercase, to continue:

```
$ fledge reset
This script will remove all data stored in the server.
Enter YES if you want to continue: YES
$
```

### 15.1.6 Fledge Tasks

Tasks are part of the Fledge IoT platform. They are like services, but with a clear distinction:

- *services* are started at a certain point (usually at startup) and they are likely to continue to work until Fledge stops.
- *tasks* are started when required, they execute a job and then they terminate.

In simple terms, a service is meant to always listen and react to requests, while a task is triggered by an event and then when job is terminated, the tasks ends.

That said, tasks and services shared these same features:

- They are both started by the Fledge scheduler. It is likely that services are started at startup, while tasks can start at a given time or interval.
- They both use the internal API to communicate with other services.
- They both use the same pluggable architecture to separate a common logic, usually associated to the internal features of Fledge, from a more generic logic, usually closer to the type of operations that must be performed.

In this chapter we present a set of tasks that are commonly available in Fledge.

## Purge

The *Purge* task is triggered by the scheduler to purge old data that is still stored (buffered) in Fledge. The logic applied to the task is relatively simple:

- The task is called exclusively (i.e. there cannot be more than one *Purge* task running at any given time) by the Fledge scheduler every hour (by default).
- Data that is older than a certain date/time is removed.
- Optionally, data is removed if the total size of the stored objects is bigger than 1GByte (default)
- Optionally, data is not removed if it has not been extracted and used by any North task or service yet.
- All purge operations are stored in the audit log.



## Purge Schedule

*Purge* is one of the tasks launched by the Fledge scheduler. You can retrieve information about the scheduling by calling the *GET* method of the *schedule* call. The name and the process name of the task are both *purge*:

```
$ curl -sX GET http://localhost:8081/fledge/schedule
...
{ "id"          : "cea17db8-6ccc-11e7-907b-a6006ad3dba0",
  "name"        : "purge",
  "time"        : 0,
  "enabled"     : true,
  "repeat"      : 3600,
  "type"        : "INTERVAL",
  "exclusive"   : true,
  "processName" : "purge",
  "day"         : null },
...
$
```

As you can see from the JSON output, the task is scheduled to be executed every hour (3,600 seconds). In order to change the interval between *Purge* tasks, you can call the *PUT* method of the *schedule* call by passing the associated *id*. For example, in order to change the task to be executed any 5 minutes (i.e. 300 seconds) you should call:

```
$ curl -sX PUT http://localhost:8081/fledge/schedule/cea17db8-6ccc-11e7-907b-a6006ad3dba0 -d '{"repeat": 300}'
{"schedule": { "id": "cea17db8-6ccc-11e7-907b-a6006ad3dba0",
               "name"      : "purge",
               "time"      : 0,
               "enabled"   : true,
               "repeat"    : 300,
               "type"      : "INTERVAL",
               "exclusive" : true,
               "processName": "purge",
               "day"       : null }
}
$
```

## Purge Configuration

The configuration of the *Purge* task is stored in the metadata structures of Fledge and it can be retrieved using the *GET* method of the *category/PURGE\_READ* call. This is the command used to retrieve the configuration in JSON format:

```
$ curl -sX GET http://localhost:8081/fledge/category/PURGE_READ
{ "retainUnsent" : { "type": "boolean",
                    "default": "False",
                    "description": "Retain data that has not been sent to any_
↳historian yet.",
                    "value": "False" },
  "age"          : { "type": "integer",
                    "default": "72",
                    "description": "Age of data to be retained, all data that is_
↳older than this value will be removed, unless retained. (in Hours)",
                    "value": "72" },
  "size"         : { "type": "integer",
                    "default": "1000000",
```

(continues on next page)



(continued from previous page)

```

    "description": "Maximum size of data to be retained, the oldest
↳ data will be removed to keep below this size, unless retained. (in Kbytes)",
    "value": "1000000" } }
$

```

Changes can be applied using the *PUT* method for each parameter call. For example, in order to change the retention policy for data that has not been sent to historians yet, you can use this call:

```

$ curl -sX PUT http://localhost:8081/fledge/category/PURGE_READ/retainUnsent -d '{
↳ "value": "True"}'
{ "type": "boolean",
  "default": "False",
  "description": "Retain data that has not been sent to any historian yet.",
  "value": "True" }
$

```

The following table shows the list of parameters that can be changed in the *Purge* task:

Item	Type	Default	Description
retainUnsent	boolean	False	Retain data that has not been sent to “North” yet When <i>True</i> , data that has not yet been retrieved by any North service or task, will not be purged. When <i>False</i> , data is purged without checking whether it has been sent to a North destination yet or not.
age	integer	72	Age in hours of the data to be retained. Data that is older than this value, will be purged.
size	integer	1000000	Size in KBytes of data that will be retained in Fledge. Older data will be removed to keep the data stored in Fledge below this size.

## 15.2 Building and using Fledge on Raspbian

Fledge requires the use of Python 3.5.3+ in order to support the asynchronous IO mechanisms used by Fledge. Earlier Raspberry Pi Raspbian distributions support Python 3.4 as the latest version of Python. In order to build and run Fledge on Raspbian the version of Python must be updated manually if your distribution has an older version.

**NOTE:** These steps must be executed *in addition* to what is described in the README file when you install Fledge on Raspbian.

Check your Python version by running the command

```

$ python3 --version
$

```

If your version is less than 3.5.3 then follow the instructions below to update your Python version.

Install and update the build tools required for Python to be built

```

$ sudo apt-get update
$ sudo apt-get install build-essential tk-dev
$ sudo apt-get install libncurses5-dev libncursesw5-dev libreadline6-dev
$ sudo apt-get install libdb5.3-dev libgdbm-dev libsqlite3-dev libssl-dev
$ sudo apt-get install libbz2-dev libexpat1-dev liblzma-dev zlib1g-dev
$

```



Now build and install the new version of Python

```
$ wget https://www.python.org/ftp/python/3.5.3/Python-3.5.3.tgz
$ tar zxvf Python-3.5.3.tgz
$ cd Python-3.5.3
$ ./configure
$ make
$ sudo make install
```

Confirm the Python version

```
$ python3 --version
$ pip3 --version
```

These should both return a version number as 3.5.3+, if not then check which python3 and pip3 you are running and replace these with the newly built versions. This may be caused by the newly built version being installed in /usr/local/bin and the existing python3 and pip3 being in /usr/bin. If this is the case then remove the /usr/bin versions

```
$ sudo rm /usr/bin/python3 /usr/bin/pip3
```

You may also link to the new version if you wish

```
$ sudo ln -s /usr/bin/python3 /usr/local/bin/python3
$ sudo ln -s /usr/bin/pip3 /usr/local/bin/pip3
```

Once python3.5 has been installed you may follow the instructions in the README file to build, install and run Fledge on Raspberry Pi using the Raspbian distribution.



## OMF KERBEROS AUTHENTICATION

### 16.1 Introduction

The bundled OMF north plugin in Fledge can use a number of different authentication schemes when communicating with the various OSIssoft products. The PI Web API method in the OMF plugin supports the use of a Kerberos scheme.

The Fledge *requirements.sh* script installs the Kerberos client to allow the integration with what in the specific terminology is called KDC (the Kerberos server).

### 16.2 PI Server as the North endpoint

The OSI *Connector Relay* allows token authentication while *PI Web API* supports Basic and Kerberos authentication.

There could be more than one configuration to allow the Kerberos authentication, the easiest one is the Windows server on which the PI Server is executed act as the Kerberos server also.

The Windows Active directory should be installed and properly configured for allowing the Windows server to authenticate Kerberos requests.

### 16.3 North plugin

The North plugin has a set of configurable options that should be changed, using either the Fledge API or the Fledge GUI, to select the Kerberos authentication.

The North plugin supports the configurable option *PIServerEndpoint* for allowing to select the target among:

- Connector Relay
- PI Web API
- Edge Data Store
- OSIssoft Cloud Services

The *PIWebAPIAuthenticationMethod* option permits to select the desired authentication among:

- anonymous
- basic
- kerberos

The Kerberos authentication requires a keytab file, the *PIWebAPIKerberosKeytabFileName* option specifies the name of the file expected under the directory:



```
${FLEDGE_ROOT}/data/etc/kerberos
```

### NOTE:

- A *keytab* is a file containing pairs of Kerberos principals and encrypted keys (which are derived from the Kerberos password). A *keytab* file allows to authenticate to various remote systems using Kerberos without entering a password.

the *AFHierarchyLevel* option allows to specific the first level of the hierarchy that will be created into the Asset Framework and will contain the information for the specific North plugin.

## 16.4 Fledge server configuration

The server on which Fledge is going to be executed needs to be properly configured to allow the Kerberos authentication.

The following steps are needed:

- *IP Address resolution for the KDC*
- *Kerberos client configuration*
- *Kerberos keytab file setup*

### 16.4.1 IP Address resolution of the KDC

The Kerberos server name should be resolved to the corresponding IP Address, editing the */etc/hosts* is one of the possible and the easiest way, sample row to add:

```
192.168.1.51    pi-server.dianomic.com pi-server
```

try the resolution of the name using the usual *ping* command:

```
$ ping -c 1 pi-server.dianomic.com

PING pi-server.dianomic.com (192.168.1.51) 56(84) bytes of data.
64 bytes from pi-server.dianomic.com (192.168.1.51): icmp_seq=1 ttl=128 time=0.317 ms
64 bytes from pi-server.dianomic.com (192.168.1.51): icmp_seq=2 ttl=128 time=0.360 ms
64 bytes from pi-server.dianomic.com (192.168.1.51): icmp_seq=3 ttl=128 time=0.455 ms
```

### NOTE:

- *the name of the KDC should be the first in the list of aliases*

### 16.4.2 Kerberos client configuration

The server on which Fledge runs act like a Kerberos client and the related configuration file should be edited for allowing the proper Kerberos server identification. The information should be added into the */etc/krb5.conf* file in the corresponding section, for example:

```
[libdefaults]
    default_realm = DIANOMIC.COM

[realms]
```

(continues on next page)



(continued from previous page)

```
DIANOMIC.COM = {
    kdc = pi-server.dianomic.com
    admin_server = pi-server.dianomic.com
}
```

### 16.4.3 Kerberos keytab file

The keytab file should be generated on the Kerberos server and copied into the Fledge server in the directory:

```
${FLEDGE_DATA}/etc/kerberos
```

#### NOTE:

- if **FLEDGE\_DATA** is not set its value should be *\$FLEDGE\_ROOT/data*.

The name of the file should match the value of the North plugin option *PIWebAPIKerberosKeytabFileName*, by default *piwebapi\_kerberos\_https.keytab*

```
$ ls -l ${FLEDGE_DATA}/etc/kerberos
-rwxrwxrwx 1 fledge fledge 91 Jul 17 09:07 piwebapi_kerberos_https.keytab
-rw-rw-r-- 1 fledge fledge 199 Aug 13 15:30 README.rst
```

The way the keytab file is generated depends on the type of the Kerberos server, in the case of Windows Active Directory this is an sample command:

```
ktpass -princ HTTPS/pi-server@DIANOMIC.COM -mapuser Administrator@DIANOMIC.COM -pass_
↪Password -crypto AES256-SHA1 -ptype KRB5_NT_PRINCIPAL -out C:\Temp\piwebapi_
↪kerberos_https.keytab
```

### 16.4.4 Troubleshooting the Kerberos authentication

- 1) check the North plugin configuration, a sample command

```
curl -s -S -X GET http://localhost:8081/fledge/category/North_Readings_to_PI | jq ".|
↪{URL, \"PIServerEndpoint\", PIWebAPIAuthenticationMethod, PIWebAPIKerberosKeytabFileName,
↪AFHierarchyLevel}\"
```

- 2) check the presence of the keytab file

```
$ ls -l ${FLEDGE_ROOT}/data/etc/kerberos
-rwxrwxrwx 1 fledge fledge 91 Jul 17 09:07 piwebapi_kerberos_https.keytab
-rw-rw-r-- 1 fledge fledge 199 Aug 13 15:30 README.rst
```

- 3) verify the reachability of the Kerberos server (usually the PI Server) - Network reachability

```
$ ping pi-server.dianomic.com
PING pi-server.dianomic.com (192.168.1.51) 56(84) bytes of data.
64 bytes from pi-server.dianomic.com (192.168.1.51): icmp_seq=1 ttl=128 time=5.07 ms
64 bytes from pi-server.dianomic.com (192.168.1.51): icmp_seq=2 ttl=128 time=1.92 ms
```

Kerberos reachability and keys retrieval



```
$ kinit -p HTTPS/pi-server@DIANOMIC.COM
Password for HTTPS/pi-server@DIANOMIC.COM:
$ klist
Ticket cache: FILE:/tmp/krb5cc_1001
Default principal: HTTPS/pi-server@DIANOMIC.COM

Valid starting      Expires            Service principal
09/27/2019 11:51:47  09/27/2019 21:51:47  krbtgt/DIANOMIC.COM@DIANOMIC.COM
    renew until 09/28/2019 11:51:46
$
```



## FLEDGE PLUGINS

The following set of plugins are available for Fledge. These plugins extend the functionality by adding new sources of data, new destinations, processing filters that can enhance or modify the data, rules for notification delivery and notification delivery mechanisms.

### 17.1 South Plugins

South plugins add new ways to get data into Fledge, a number of south plugins are available ready built or users may add new south plugins of their own by writing them in Python or C/C++.

Table 1: Fledge South Plugins

Name	Description
am2315	Fledge south plugin for an AM2315 temperature and humidity sensor
b100-modbus-python	A south plugin to read data from a Dynamic Ratings B100 device over Modbus
benchmark	A Fledge benchmark plugin to measure the ingestion rates on particular hardware
cc2650	A Fledge south plugin for the Texas Instruments SensorTag CC2650
coap	A south plugin for Fledge that pulls data from a COAP sensor
coral-enviro	A south plugin for the Google Coral Environmental Sensor Board
csv	A Fledge south plugin in C++ for reading CSV files
csv-async	A Fledge asynchronous plugin for reading CSV data
csvplayback	Plays a CSV at some configurable speed and each column of the file will become a datapoint of an asset using pandas library.
dht	A Fledge south plugin in C++ that interfaces to a DHT-11 temperature and humidity sensor
dht11	A Fledge south plugin that interfaces a DHT-11 temperature sensor
dnp3	A south plugin for Fledge that implements the DNP3 protocol
expression	A Fledge south plugin that uses a user define expression to generate data
FlirAX8	A Fledge hybrid south plugin that uses fledge-south-modbus-c to get temperature data from a Flir Thermal camera
game	The south plugin used for the Fledge lab session game involving remote controlled cars
http	A Python south plugin for Fledge used to connect one Fledge instance to another
ina219	A Fledge south plugin for the INA219 voltage and current sensor
J1708	A plugin that uses the SAE J1708 protocol to load data from the ECU of heavy duty vehicles.
J1939	A CANBUS J1839 plugin to collect data into Fledge.
lathesim	A simulation plugin used as a demonstration to show how data can be collected within Fledge. This plugin simulates various properties of a lathe.

continues on next page



Table 1 – continued from previous page

Name	Description
modbus-c	A Fledge south plugin that implements modbus-tcp and modbus-rtu
modbustcp	A Fledge south plugin that implements modbus-tcp in Python
mqtt	Fledge South MQTT Subscriber Plugin
mqtt-sparkplug	A Fledge south plugin that implements the Sparkplug API over MQTT
opcua	A Fledge south service that pulls data from an OPC-UA server
openweathermap	A Fledge south plugin to pull weather data from OpenWeatherMap
person-detection	Fledge south service plugin that detects person in the live video stream
playback	A Fledge south plugin to replay data stored in a CSV file
pt100	A Fledge south plugin for the PT100 temperature sensor
random	A south plugin for Fledge that generates random numbers
randomwalk	A Fledge south plugin that returns data that with randomly generated steps
roxtec	A Fledge south plugin for the Roxtec cable gland project
rpienviro	A Fledge south service for the Raspberry Pi EnviroPhat sensors
s2opcua	An OPCUA south plugin based on the Safe & Secure OPCUA library. This plugin offers similar functionality to the fledge-south-opcua plugin but also offers encryption and authentication.
s7-python	fledge-south-s7-python repository
sensehat	A Fledge south plugin for the Raspberry Pi Sensehat sensors
sensorphone	A Fledge south plugin the task to the iPhone SensorPhone app
sinusoid	A Fledge south plugin that produces a simulated sine wave
systeminfo	A Fledge south plugin that gathers information about the system it is running on.
usb4704	A Fledge south plugin the Advantech USB-4704 data acquisition module
wind-turbine	A Fledge south plugin for a number of sensor connected to a wind turbine demo

## 17.2 North Plugins

North plugins add new destinations to which data may be sent by Fledge. A number of north plugins are available ready built or users may add new north plugins of their own by writing them in Python or C/C++.

Table 2: Fledge North Plugins

Name	Description
azure	A north plugin that sends data to Microsoft Azure IoT Core.
gcp	A north plugin to send data to Google Cloud Platform IoT Core
harperdb	A north plugin that sends data to the HarperDB SQL/NoSQL data management platform
http	A Python implementation of a north plugin to send data between Fledge instances using HTTP
http-c	A Fledge north plugin that sends data between Fledge instances using HTTP/HTTPS
kafka	A Fledge plugin for sending data north to Apache Kafka
kafka-python	A Python implementation of a north plugin that can send data to Apache Kafka
mqtt-scripted	A mqtt-scripted north plugin
s7-python	fledge-north-s7-python repository
thingspeak	A Fledge north plugin to send data to Matlab's ThingSpeak cloud
OMF	Send data to OSIsoft PI Server, Edge Data Store or OSIsoft Cloud Services



## 17.3 Filter Plugins

Filter plugins add new ways in which data may be modified, enhanced or cleaned as part of the ingress via a south service or egress to a destination system. A number of north plugins are available ready built or users may add new north plugins of their own by writing them in Python or C/C++.

It is also possible, using particular filters, to supply expressions or script snippets that can operate on the data as well. This provides a simple way to process the data in Fledge as it is read from devices or written to destination systems.

Table 3: Fledge Filter Plugins

Name	Description
asset	A Fledge processing filter that is used to block or allow certain assets to pass onwards in the data stream
change	A Fledge processing filter plugin that only forwards data that changes by more than a configurable amount
delta	A Fledge processing filter plugin that removes duplicates from the stream of data and only forwards new values that differ from previous values by more than a given tolerance
expression	A Fledge processing filter plugin that applies a user define formula to the data as it passes through the filter
fft	A Fledge processing filter plugin that calculates a Fast Fourier Transform across sensor data
Flir-Validity	A Fledge processing filter used for processing temperature data from a Flir thermal camera
log	A Fledge filter that converts the readings data to a logarithmic scale. This is the example filter used in the plugin developers guide.
metadata	A Fledge processing filter plugin that adds metadata to the readings in the data stream
omfhint	A filter plugin that allows data to be added to assets that will provide extra information to the OMF north plugin.
python27	A Fledge processing filter that allows Python 2 code to be run on each sensor value.
python35	A Fledge processing filter that allows Python 3 code to be run on each sensor value.
rate	A Fledge processing filter plugin that sends reduced rate data until an expression triggers sending full rate data
rename	A Fledge processing filter that is used to modify the name of an asset, datapoint or with both
replace	Filter to replace characters in the names of assets and data points in readings object.
rms	A Fledge processing filter plugin that calculates RMS value for sensor data
scale	A Fledge processing filter plugin that applies an offset and scale factor to the data
scale-set	A Fledge processing filter plugin that applies a set of scale factors to the data
threshold	A Fledge processing filter that only forwards data when a threshold is crossed



## 17.4 Notification Rule Plugins

Notification rule plugins provide the logic that is used by the notification service to determine if a condition has been met that should trigger or clear that condition and hence send a notification. A number of notification plugins are available as standard, however as with any plugin the user is able to write new plugins in Python or C/C++ to extend the set of notification rules.

Table 4: Fledge Notification Rule Plugins

Name	Description
average	A Fledge notification rule plugin that evaluates an expression based sensor data notification rule plugin that triggers when sensors values depart from the moving average by more than a configured limit.
delta	A delta rule plugin
outofbound	A Fledge notification rule plugin that triggers when sensors values exceed limits set in the configuration of the plugin.
simple-expression	A Fledge notification rule plugin that evaluates an expression based sensor data
watchdog	Notification rule designed to be triggered if data for a given asset is not ingested for a period of time.

## 17.5 Notification Delivery Plugins

Notification delivery plugins provide the mechanisms to deliver the notification messages to the systems that will receive them. A number of notification delivery plugins are available as standard, however as with any plugin the user is able to write new plugins in Python or C/C++ to extend the set of notification deliveries.

Table 5: Fledge Notification Delivery Plugins

Name	Description
alexa-notifyme	A Fledge notification delivery plugin that sends notifications to the Amazon Alexa platform
asset	A Fledge notification delivery plugin that creates an asset in Fledge when a notification occurs
blynk	A Fledge notification delivery plugin that sends notifications to the Blynk service
control	A control notify plugin
customasset	A Fledge notification delivery plugin that creates an event asset in readings.
email	A Fledge notification delivery plugin that sends notifications via email
google-hangouts	A Fledge notification delivery plugin that sends alerts on the Google hangout platform
ifttt	A Fledge notification delivery plugin that triggers an action of IFTTT
mqtt	A notification delivery plugin that sends messages via MQTT when a notification is triggered or cleared. This is the example used in the notification delivery plugin writers guide.
operation	A notification delivery plugin that will cause an operation to be trigger via the set point control operation API of a south service.
python35	A Fledge notification delivery plugin that runs an arbitrary Python 3 script
setpoint	A fledge notification plugin that invokes a set point operation on a south service.
slack	A Fledge notification delivery plugin that sends notifications via the slack instant messaging platform
telegram	A Fledge notification delivery plugin that sends notifications via the telegram service



## VERSION HISTORY

### 18.1 Fledge v2

#### 18.1.1 v2.0.0

Release Date: 2022-09-09

- **Fledge Core**

- New Features:

- \* Add options for choosing the Fledge Asset name: Browser Name, Subscription Path and Full Path. Use the OPC UA Source timestamp as the User Timestamp in Fledge.
- \* The storage interface used to query generic configuration tables has been improved to support tests for null and non-null column values.
- \* The ability for north services to support control inputs coming from systems north of Fledge has been introduced.
- \* The handling of a failed storage service has been improved. The client now attempt to re-connect and if that fails they will down. The logging produced is now much less verbose, removing the repeated messages previously seen.
- \* A new service has been added to Fledge to facilitate the routing of control messages within Fledge. This service is responsible for determining which south services to send control requests to and also for the security aspects of those requests.
- \* Ensure that new Fledge data types not supported by OMF are not processed.
- \* The storage service now supports a richer set of queries against the generic table interface. In particular, joins between tables are now supported.
- \* OPC UA Security has been enhanced. This plugin now supports Security Policies Basic256 and Basic256Sha256, with Security Modes Sign and Sign & Encrypt. Authentication types are anonymous and username/password.
- \* South services that have a slow poll rate can take a long time to shutdown, this sometimes resulted in those services not shutting down cleanly. The shutdown process has been modified such that these services now shutdown promptly regardless of polling rate.
- \* A new configuration item type has been added for the selection of access control lists.
- \* Support has been added to the Python query builder for NULL and NOT NULL columns.
- \* The Python query builder has been updated to support nested database queries.



- \* The third party packages on which Fledge is built have been updated to use the latest versions to resolve issues with vulnerabilities in these underlying packages.
- \* When the data stream from a south plugin included an OMF Hint of AFLocation, performance of the OMF North plugin would degrade. In addition, process memory would grow over time. These issues have been fixed.
- \* The version of the PostgreSQL database used by the Postgres storage plugin has been updated to PostgreSQL 13.
- \* An enhancement has been added to the North service to allow the user to specify the block size to use when sending data to the plugin. This helps tune the north services and is described in the tuning guide within the documentation.
- \* The notification server would previously output warning messages when it was starting, these were not an indication of a problem and should have been information messages. This has now been resolved.
- \* The backup mechanism has been improved to include some external items in the backup and provide a more secure backup.
- \* The purge option that controls if unsent assets can be purged or not has been enhanced to provide options for sent to any destination or sent to all destinations as well as sent to no destinations.
- \* It is now possible to add control features to Python south plugins.
- \* Certificate based authentication is now possible between services in a single instance. This allows for secure control messages to be implemented between services.
- \* Performance improvements have been made such that the display of south service data when large numbers of assets are in use.
- \* The new micro service, control dispatcher, is now available as a package that can be installed via the package manager.
- \* New data types are now supported for data points within an asset and are encoded into various Python types when passed to Python plugins or scripts run within standard plugin. This includes numpy arrays for images and data buffers, 2 dimensional Python lists and others. Details of the type encoding can be found in the plugin developers guide of the online product documentation.
- \* The mechanism for online update of configuration has been extended to allow for more configuration to be modified without the need to restart any services.
- \* Support has been added for the Raspberry Pi Bullseye release.
- \* A problem with a file descriptor leak in Python that could cause Fledge to fail has been resolved.
- \* The control of logging levels has now been added to the Python code run within a service such that the advanced settings option is now honoured by the Python code.
- \* Enhancements have been made to the asset tracker API to retrieve the service responsive for the ingest of a given asset.
- \* A new API has been added to allow external viewing and managing of the data that various plugins persist.
- \* A new REST API entry point has been added that allows all instances of a specified asset to be purged from the buffer. A further entry point has also been added to purge all data from the reading buffer. These entry points should be used with care as they will cause data to be discarded.
- \* A new parameter has been added to the asset retrieval API that allows image data to be returned, `images=include`. By default image type datapoints will be replaced with a message, "Image removed for brevity", in order to reduce the size of the returned payload.



- \* A new API has been added to the management API that allows services to request that URL's in the public API are proxied to the service API. This is used when extending the functionality of the system with custom microservices.
- \* A new set of API calls have been added to the public REST API of the product to support the control dispatcher and for the creation and management of control scripts.
- \* A new API has been added to the public API that will return the latest reading for a given asset. This will return all data types including images.
- \* A new API has been added that allows asset tracking records to be marked as deprecated. This allows the flushing of relationships between assets and the services that have processed them. It is useful only in development systems and should not be used in production systems.
- \* A new API call has been added that allows the persisted data related to a plugin to be retrieved via the public REST API. This is intended for use by plugin writers and to allow for better tracking of data persisted between service executions.
- \* A new query parameter has been added to the API used to fetch log messages from the system log, nontotals. This will increase the performance of the call at the expense of not returning the total number of logs that match the search criteria.
- \* New API entry points have been added for the management of Python packages.
- \* Major performance improvements have been made to the code for retrieving log messages from the system log. This is mainly an issue on systems with very large log files.
- \* The storage service API has been extended to support the creation of private schemas for the use of optional micro services registered to a Fledge instance.
- \* Filtering by service type has now been added to the API that retrieve service information via the public REST API.
- \* A number of new features have been added to the user interface to aid developers creating data pipelines and plugins. These features allow for manual purging of data, deprecating the relationship between the services and the assets they have ingested and viewing the persisted data of the plugins. These are all documented in the section on developing pipelines within the online documentation.
- \* A new section has been added to the documentation which discusses the process and best practices for building data pipelines in Fledge.
- \* A glossary has been added to the documentation for the product.
- \* The documentation that describes the writing of asynchronous Python plugins has been updated in line with the latest code changes.
- \* The documentation has been updated to reflect the new tabs available in the Fledge user interface for editing the configuration of services and tasks.
- \* A new introduction section has been added to the Fledge documentation that describes the new features and some typical use cases of Fledge.
- \* A new section has been added to the Fledge Tuning guide that discusses the tuning of North services and tasks. Also scheduler tuning has been added to the tuning guide along with the tuning of the service monitor which is used to detect failures of services within Fledge.
- \* The Tuning Fledge section of the documentation has been updated to include information on tuning the Fledge service monitor that is used to monitor and restart Fledge services. A section has also been added that describes the tuning of north services and tasks. A new section describes the different storage plugins available, when they should be used and how to tune them.



- \* Added an article on Developing with Windows Subsystem for Linux (WSL2) to the Plugin Developer Guide. WSL2 allows you to run a Linux environment directly on Windows without the overhead of Windows Hyper-V. You can run Fledge and develop plugins on WSL2.
  - \* Documentation has been added for the purge process and the new options recently added.
  - \* Documentation has been added to the plugin developer guides that explain what needs to be done to allow the packaging mechanism to be able to package a plugin.
  - \* Documentation has been added to the Building Pipelines section of the documentation for the new UI feature that allows Python packages to be installed via the user interface.
  - \* Documentation has been updated to show how to build Fledge using the requirements.sh script.
  - \* The documentation ordering has been changed to make the section order more logical.
  - \* The plugin developers guide has been updated to include information on the various flags that are used to communicate the options implemented by a plugin.
  - \* Updated OMF North plugin documentation to include current OSIsoft (AVEVA) product names.
  - \* Fixed a typo in the quick start guide.
  - \* Improved north plugin developers documentation is now available.
- Bug Fix:
- \* The Fledge control script has options for purge and reset that requires a confirmation before it will continue. The message that was produced if this confirmation was not given was unclear. This has now been improved.
  - \* An issue that could cause a north service or task that had been disabled for a long period of time to fail to send data when it was re-enabled has been resolved.
  - \* S2OPCUA Toolkit changes required an update in build procedures for the S2OPCUA South Plugin.
  - \* Previously it has not been possible to configure the advanced configuration of a south service until it has been run at least once. This has now been resolved and it is possible to add a south service in disable mode and edit the advanced configuration.
  - \* The diagnostics when a plugin fails to load have been improved.
  - \* The South Plugin shutdown problem was caused by errors in the plugin startup procedure which would throw an exception for any error. The plugin startup has been fixed so errors are reported properly. The problem of plugin shutdown when adding a filter has been resolved.
  - \* The S2OPCUA South Plugin would throw an exception for any error during startup. This would cause the core system to shut down the plugin permanently after a few retries. This has been fixed. Error messages has been recategorized to properly reflect informational, warning and error messages.
  - \* The update process has been optimised to remove an unnecessary restart if no new version of the software are available.
  - \* The OMF North plugin was unable to process configuration changes or shut down if the PI Web API hostname was not correct. This has been fixed.
  - \* S2OPC South plugin builds have been updated to explicitly reference S2OPC Toolkit Version 1.2.0.
  - \* An issue that could on rare occasions cause the SQLite plugin to silently discard readings has been resolved.
  - \* An issue with the automatic renewal of authentication certificates has been resolved.



- \* Deleting a service which had a filter pipeline could cause some orphaned configuration information to be left stored. This prevented creating filters of the same name in the future. This has now been resolved.
- \* The error reporting has been improved when downloading backups from the system.
- \* An issue that could cause north plugins to occasionally fail to shutdown correctly has now been resolved.
- \* Some fixes are made in Package update API that allows the core package to be updated.
- \* The documentation has been updated to correct a statement regarding running the south side as a task.

- **GUI**

- New Features:

- \* A new *Developer* item has been added to the user interface to allow for the management of Python packages via the UI. This is enabled by turning on developer features in the user interface *Settings* page.
    - \* A control has been added that allows the display of assets in the *South* screen to be collapsed or expanded. This allows for more services to be seen when services ingest multiple assets.
    - \* A new feature has been added to the south page that allows the relationship between an asset and a service to be deprecated. This is a special feature enabled with the Developer Features option. See the documentation on building pipelines for a full description.
    - \* A new feature has been added to the Assets and Readings page that allows for manual purging of named assets or all assets. This is a developer only feature and should not be used on production systems. The feature is enabled, along with other developer features via the Settings page.
    - \* A new feature has been added to the South and North pages for each service that allows the user to view, import, export and delete the data persisted by a plugin. This is a developer only feature and should not be used on production systems. It is enabled via the Setting page.
    - \* A new configuration type, Access Control List, is now supported in user interface. This allows for selection of an ACL from those already created.
    - \* A new tabbed layout has been adopted for the editing of south and north services and tasks. Configuration, Advanced and Security tabs are supported as our tabs for developer features if enabled.
    - \* The user interface for displaying system logs has been modified to improve the performance of log viewing.
    - \* The User Interface has been updated to use the latest versions of a number of packages it depends upon, due to vulnerabilities reported in those packages.
    - \* With the introduction of image data types to the readings supported by the system the user interface has been updated to add visualisation features for these images. A new feature also allows the latest reading for a given asset to be shown.
    - \* A new feature has been added to the south and north pages that allows the user to view the logs for the service.
    - \* The service status display now includes the Control Dispatcher service if it has been installed.
    - \* The user interface now supports the new control dispatcher service. This includes the graphical creation and editing of control scripts and access control lists used by control features.
    - \* An option has been added to the Asset and Readings page to show just the latest values for a given asset.



- \* The notification user interface now links to the relevant sections of the online documentation allowing users to navigate to the help based on the current context.
- \* Some timezone inconsistencies in the user interface have been resolved.
- Bug Fix:
  - \* An issue that would cause the GUI to not always allow JSON data to be saved has been resolved.
  - \* An issue with the auto refresh in the systems log page that made selecting the service to filter difficult has been resolved.
  - \* The sorting of services and tasks in the South and North pages has been improved such that enabled services appear above disabled services.
  - \* An issue the prevented gaps in the data from appearing in the graphs displayed by the GUI has now been resolved.
  - \* Entering times in the GUI could sometimes be difficult and result in unexpected results. This has now been improved to ease the entry of time values.
- **Plugins**
  - New Features:
    - \* A new fledge-notify-control plugin has been added that allows notifications to be delivered via the control dispatcher service. This allows the full features of the control dispatcher to be used with the edge notification path.
    - \* A new fledge-notify-customasset notification delivery plugin that creates an event asset in readings.
    - \* A new fledge-rule-delta notification rule plugin that triggers when a data point value changes.
    - \* A new fledge-rule-watchdog notification rule plugin that allows notifications to be sent if data stops being ingress for specified assets.
    - \* Support has been added for proxy servers in the north HTTP-C plugin.
    - \* The OPCUA north plugin has been updated to include the ability for systems outside of Fledge to write to the server that Fledge advertises. These writes are taken as control input into the Fledge system.
    - \* The HTTPC North plugin has been enhanced to add an optional Python script that can be used to format the payload of the data sent in the HTTP REST request.
    - \* The SQLite storage plugins have been updated to support service extension schemas. This is a mechanism that allows services within the Fledge system to add new schemas within the storage service that are exclusive to that service.
    - \* The Python35 filter has been updated to use the common Python interpreter. This allows for packages such as numpy to be used. The resilience and error reporting of this plugin have also been improved.
    - \* A set of developer only features designed to aid the process of developing data pipelines and plugins has been added in this release. These features are turned on and off via a toggle setting on the Settings page.
    - \* A new option has been added to the Python35 filter that changes the way datapoint names are used in the JSON readings. Previously there had to be encoded and decoded by use of the b'xxx' mechanism. There is now a toggle that allows for either this to be required or simple text string use to be enabled.
    - \* The API of the storage service has been updated to allow for custom schemas to be created by services that extend the core functionality of the system.
    - \* New image type datapoints can now be sent between instances using the http north and south plugins.



- \* The ability to define response headers in the http south plugin has been added to aid certain circumstances where CORS provided data flows.
  - \* The documentation of the Python35 filter has been updated to include a fuller description of how to make use of the configuration data block supported by the plugin.
  - \* The documentation describing how to run services under the debugger has been improved along with other improvements to the documentation aimed at plugin developers.
  - \* Documentation has been added for fledge-north-azure plugin.
  - \* Documentation has now been added for fledge-north-harperdb plugin.
- Bug Fix:
- \* Build procedures were updated to accommodate breaking changes in the S2OPC OPCUA Toolkit.
  - \* Occasionally switching from the sqlite to the sqlitememory plugin for the storage of readings would cause a fatal error in the storage layer. This has now been fixed and it is possible to change to sqlitememory without an error.
  - \* A race condition within the modbus south plugin that could cause unfair scheduling of read versus write operations has been resolved. This could cause write operations to be delayed in some circumstances. The scheduling of set point write operations is now fairly interleaved between the read operations in all cases.
  - \* A problem that caused the HTTPC North plugin to fail if the path component of the URL was omitted has been resolved.
  - \* The modbus-c south plugin documentation has been enhanced to include details of the function codes used to read modbus data. Also incorrect error message and others have been improved to aid resolving configuration issues. The documentation has been updated to include descriptive text for the error messages that may occur.
  - \* The Python35 filter plugin has been updated such that if no data is to be passed onwards it may now simply return the None Python constant or an empty list. Also it allows simple Python scripts to be added into filter pipelines has had a number of updates to improve the robustness of the plugin in the event of incorrect script code being provided by the user. The behaviour of the plugin has also been updated such that any errors run the script will prevent data being passed onwards the filter pipeline. An error explaining the exact cause of the failure is now logged in the system log. Also its documentation has been updated to discuss Python package imports and issues when removing previously used imports.
  - \* The Average rule has been updated to improve the user interaction during the configuration of the rule.
  - \* The first time a plugin that persisted data is executed erroneous errors and warnings would be written to the system log. This has now been resolved.
  - \* An issue with the Kafka north plugin not sending data in certain circumstances has been resolved.
  - \* Adding some notification plugins would cause incorrect errors to be logged to the system log. The functioning of the notifications was not affected. This has now been resolved and the error logs no longer appear.
  - \* The documentation for the fledge-rule-delta plugin has been corrected.



## 18.2 Fledge v1

### 18.2.1 v1.9.2

Release Date: 2021-09-29

- **Fledge Core**

- New Features:

- \* The ability for south plugins to persist data between executions of south services has been added for plugins written in C/C++. This follows the same model as already available for north plugins.
- \* Notification delivery plugins now also receive the data that caused the rule to trigger. This can be used to deliver values in the notification delivery plugins.
- \* A new option has been added to the sqlite storage plugin only that allows assets to be excluded from consideration in the purge process.
- \* A new purge process has been added to control the growth of statistics history and audit trails. This new process is known as the “System Purge” process.
- \* The support bundle has been updated to include details of the packages installed.
- \* The package repository API endpoint has been updated to support Ubuntu 20.04 repository end point.
- \* The handling of updates from RPM package repositories has been improved.
- \* The certificate store has been updated to support more formats of certificates, including DER, P12 and PFX format certificates.
- \* The documentation has been updated to include an improved & detailed introduction to filters.
- \* The OMF north plugin documentation has been re-organised and updated to include the latest features that have been introduced to this plugin.
- \* A new section has been added to the documentation that discusses the tuning of the edge based control path.

- **Bug Fix:**

- \* A rare race condition during ingestion of readings would cause the south service to terminate and restart. This has now been resolved.
- \* In some circumstances it was seen that north services could send the same data more than once. This has now been corrected.
- \* An issue that caused an intermittent error in the tracking of data sent north has been resolved. This only impacted north services and not north tasks.
- \* An optimisation has been added to prevent north plugins being sent empty data sets when the filter chain removes all the data in a reading set.
- \* An issue that prevented a north service restarting correctly when certain combinations of filters were present has been resolved.
- \* The API for retrieving the list of backups on the system has been improved to honour the limit and offset parameters.
- \* An issue with the restore operation always restoring the latest backup rather than the chosen backup has been resolved.
- \* The support package failed to include log data if binary data had been written to syslog. This has now been resolved.



- \* The configuration category for the system purge was in the incorrect location with the configuration category tree, this has now been correctly placed underneath the “Utilities” item.
- \* It was not possible to set a notification to always retrigger as there was a limitation that there must always be 1 second between notification triggers. This restriction has now been removed and it is possible to set a retrigger time of zero.
- \* An error in the documentation for the plugin developers guide which incorrectly documented how to build debug binaries has been corrected.

- **GUI**

- New Features:

- \* The user interface has been updated to improve the filtering of logs when a large number of services have been defined within the instance.
    - \* The user interface input validation for hostnames and port has been improved in the setup screen. A message is now displayed when an incorrect port or address is entered.
    - \* The user interface now prompts to accept a self signed certificate if one is configured.

- Bug Fix:

- \* If a south or north plugin included a script type configuration item the GUI failed to allow the service or task using this plugin to be created correctly. This has now been resolved.
    - \* The ability to paste into password fields has been enabled in order to allow copy/paste of keys, tokens etc into configuration of the south and north services.
    - \* An issue that could result in filters not being correctly removed from a pipeline of 2 or more filters has been resolved.

- **Plugins**

- New Features:

- \* A new OPC/UA south plugin has been created based on the Safe and Secure OPC/UA library. This plugin supports authentication and encryption mechanisms.
    - \* Control features have now been added to the modbus south plugin that allows the writing of registers and coils via the south service control channel.
    - \* The modbus south control flow has been updated to use both 0x06 and 0x10 function codes. This allows items that are split across multiple modbus registers to be written in a single write operation.
    - \* The OMF plugin has been updated to support more complex scenarios for the placement of assets with the PI Asset Framework.
    - \* The OMF north plugin hinting mechanism has been extended to support asset framework hierarchy hints.
    - \* The OMF north plugin now defaults to using a concise naming scheme for tags in the PI server.
    - \* The Kafka north plugin has been updated to allow timestamps of higher granularity than 1 second, previously timestamps would be truncated to the previous second.
    - \* The Kafka north plugin has been enhanced to give the option of sending JSON objects as strings to Kafka, as previously the default, or sending them as JSON objects.
    - \* The HTTP-C north plugin has been updated to allow the inclusion of customer HTTP headers.
    - \* The Python35 Filter plugin did not correctly handle string type data points. This has now been resolved.



- \* The OMF Hint filter documentation has been updated to describe the use of regular expressions when defining the asset name to which the hint should be applied.
- Bug Fix:
  - \* An issue with string data that had quote characters embedded within the reading data has been resolved. This would cause data to be discarded with a bad formatting message in the log.
  - \* An issue that could result in the configuration for the incorrect plugin being displayed has now been resolved.
  - \* An issue with the modbus south plugin that could cause resource starvation in the threads used for set point write operations has been resolved.
  - \* A race condition in the modbus south that could cause an issue if the plugin configuration is changed during a set point operation.
  - \* The CSV playback south plugin installation on CentOS 7 platforms has now been corrected.
  - \* The error handling of the OMF north plugin has been improved such that assets that contain data types that are not supported by the OMF endpoint of the PI Server are removed and other data continues to be sent to the PI Server.
  - \* The Kafka north plugin was not always able to reconnect if the Kafka service was not available when it was first started. This issue has now been resolved.
  - \* The Kafka north plugin would on occasion duplicate data if a connection failed and was later reconnected. This has been resolved.
  - \* A number of fixes have been made to the Kafka north plugin, these include; fixing issues caused by quoted data in the Kafka payload, sending timestamps accurate to the millisecond, fixing an issue that caused data duplication and switching the the user timestamp.
  - \* A problem with the quoting of string type data points on the North HTTP-C plugin has been fixed.
  - \* String type variables in the OPC/UA north plugin were incorrectly having extra quotes added to them. This has now been resolved.
  - \* The delta filter previously did not manage calculating delta values when a datapoint changed from being an integer to a floating point value or vice versa. This has now been resolved and delta values are correctly calculated when these changes occur.
  - \* The example path shown in the DHT11 plugin in the developers guide was incorrect, this has now been fixed.

### 18.2.2 v1.9.1

Release Date: 2021-05-27

- **Fledge Core**

- New Features:
  - \* Support has been added for Ubuntu 20.04 LTS.
  - \* The core components have been ported to build and run on CentOS 8
  - \* A new option has been added to the command line tool that controls the system. This option, called purge, allows all readings related data to be purged from the system whilst retaining the configuration. This allows a system to be tested and then reset without losing the configuration.



- \* A new service interface has been added to the south service that allows set point control and operations to be performed via the south interface. This is the first phase of the set point control feature in the product.
- \* The documentation has been improved to include the new control functionality in the south plugin developers guide.
- \* An improvement has been made to the documentation layout for default plugins to make the GUI able to find the plugin documentation.
- \* Documentation describing the installation of PostgreSQL on CentOS has been updated.
- \* The documentation has been updated to give more detail around the topic of self-signed certificates.
- Bug Fix:
  - \* A security flaw that allowed non-privileged users to update the certificate store has been resolved.
  - \* A bug that prevented users being created with certificate based authentication rather than password based authentication has been fixed.
  - \* Switching storage plugins from SQLite to PostgreSQL caused errors in some circumstances. This has now been resolved.
  - \* The HTTP code returned by the ping command has been updated to correctly report 401 errors if the option to allow ping without authentication is turned off.
  - \* The HTTP error code returned when the notification service is not available has been corrected.
  - \* Disabling and re-enabling the backup and restore task schedules sometimes caused a restart of the system. This has now been resolved.
  - \* The error message returned when schedules could not be enabled or disabled has been improved.
  - \* A problem related to readings with nested data not correctly getting copied has been resolved.
  - \* An issue that caused problems if a service was deleted and then a new service was recreated using the name of the previously deleted service has been resolved.

- **GUI**

- New Features:
  - \* Links to the online help have been added on a number of screens in the user interface.
  - \* Improvements have been made to the user management screens of the GUI.

- **Plugins**

- New Features:
  - \* North services now support Python as well as C++ plugins.
  - \* A new delivery notification plugin has been added that uses the set point control mechanism to invoke an action in the south plugin.
  - \* A new notification delivery mechanism has been implemented that uses the set point control mechanism to assert control on a south service. The plugin allows you to set the values of one or more control items on the notification triggered and set a different set of values when the notification rule clears.
  - \* Support has been added in the OPC/UA north plugin for array data. This allows FFT spectrum data to be represented in the OPC/UA server.
  - \* The documentation for the OPC/UA north plugin has been updated to recommend running the plugin as a service.



- \* A new storage plugin has been added that uses SQLite. This is designed for situations with low bandwidth sensors and stores all the readings within a single SQLite file.
  - \* Support has been added to use RTSP video streams in the person detection plugin.
  - \* The delta filter has been updated to allow an optional set of asset specific tolerances to be added in addition to the global tolerance used by the plugin when deciding to forward data.
  - \* The Python script run by the MQTT scripted plugin now receives the topic as well as the message.
  - \* The OMF plugin has been updated in line with recommendations from the OMF group regarding the use of SCRF Defense.
  - \* The OMFHint plugin has been updated to support wildcarding of asset names in the rules for the plugin.
  - \* New documentation has been added to help in troubleshooting PI connection issues.
  - \* The pi\_server and ocs north plugins are deprecated in favour of the newer and more feature rich OMF north plugin. These deprecated plugins cannot be used in north services and are only provided for backward compatibility when run as north tasks. These plugins will be removed in a future release.
- Bug Fix:
- \* The OMF plugin has been updated to better deal with nested data.
  - \* Some improvements to error handling have been added to the InfluxDB north plugin for version 1.x of InfluxDB.
  - \* The Python 35 filter stated it used the Python version 3.5 always, in reality it uses whatever Python 3 version is installed on your system. The documentation has been updated to reflect this.
  - \* Fixed a bug that treated arrays of bytes as if they were strings in the OPC/UA south plugin.
  - \* The HTTP North C plugin would not correctly shutdown, this effected reconfiguration when run as an always on service. This issue has now been resolved.
  - \* An issue with the SQLite In Memory storage plugin that caused database locks under high load conditions has been resolved.

### 18.2.3 v1.9.0

Release Date: 2021-02-19

- **Fledge Core**

- New Features:

- \* Support has been added in the Python north sending process for nested JSON reading payloads.
- \* A new section has been added to the documentation to document the process of writing a notification delivery plugin. As part of this documentation a new delivery plugin has also been written which delivers notifications via an MQTT broker.
- \* The plugin developers guide has been updated with information regarding installation and debugging of new plugins.
- \* The developer documentation has been updated to include details for writing both C++ and Python filter plugins.
- \* An always on north service has been added. This compliments the current north task and allows a choice of using scheduled windows to send data north or sending data as soon as it is available.



- \* The Python north sending process required the JQ filter information to be mandatory in north plugins. JQ filtering has been deprecated and will be removed in the next major release.
  - \* Storage plugins may now have configuration options that are controllable via the API and the graphical interface.
  - \* The ping API call has been enhanced to return the version of the core component of the system.
  - \* The SQLite storage plugin has been enhanced to distribute readings for multiple assets across multiple databases. This improves the ingest performance and also improves the responsiveness of the system when very large numbers of readings are buffered within the instance.
  - \* Documentation has been added for configuration of the storage service.
- Bug Fix:
- \* The REST API for the notification service was missing the re-trigger time information for configured notification in the retrieval and update calls. This has now been added.
  - \* If the SQLite storage plugin is configured to use managed storage Fledge fails to restart. This has been resolved, the SQLite storage service no longer uses the managed option and will ignore it if set.
  - \* An upgraded version of the HTTPS library has been applied, this solves an issue with large payloads in HTTPS exchanges.
  - \* A number of Python source files contained incorrect references to the readthedocs page. This has now been resolved.
  - \* The retrieval of log information was incorrectly including debug log output if the requested level was information and higher. This is now correctly filtered out.
  - \* If a south plugin generates bad data that can not be inserted into the storage layer, that plugin will buffer the bad data forever and continually attempt to insert it. This causes the queue to build on the south plugin and eventually will exhaust system memory. To prevent this if data can not be inserted for a number of attempts it will be discarded in the south service. This allows the bad data to be dropped and newer, good data to be handled correctly.
  - \* When a statistics value becomes greater than 2,147,483,648 the storage layer would fail, this has now been fixed.
  - \* During installation of plugins the user interface would occasionally flag the system as down due to congestion in the API layer. This has now been resolved and the correct status of the system should be reflected.
  - \* The notification service previously logged errors if no rule/delivery notification plugins had been installed. This is no longer the case.
  - \* An issue with JSON configuration options that contained escaped strings within the JSON caused the service with the associated configuration to fail to run. This has now been resolved.
  - \* The Postgres storage engine limited the length of asset codes to 50 characters, this has now been increased to 255 characters.
  - \* Notifications based on asset names that contain the character ‘.’ in the name would not receive any data. This has now been resolved.
- Known Issues:
- \* Known issues with Postgres storage plugins. During the final testing of the 1.9.0 release a problem has been found with switching to the PostgreSQL storage plugin via the user interface. Until this is resolved switching to PostgreSQL is only supported by manual editing the storage.json as per version 1.8.0. A patch to resolve this is likely to be released in the near future.

- GUI



- New Features:

- \* The user interface now shows the retrigger time for a notification.
- \* The user interface now supports adding a north service as well as a north task.
- \* A new help menu item has been added to the user interface which will cause the readthedocs documentation to be displayed. Also the wizard to add the south and north services has been enhanced to give an option to display the help for the plugins.

- Bug Fix:

- \* The user interface now supports the ability to filter on all severity levels when viewing the system log.

- **Plugins**

- New Features:

- \* The OPC/UA south plugin has been updated to allow the definition of the minimum reporting time between updates. It has also been updated to support subscription to arrays and DATE\_TIME type with the OPC/UA server.
- \* AWS SiteWise requires the SourceTimestamp to be non-null when reading from an OPC/UA server. This was not always the case with the OPC/UA north plugin and caused issues when ingesting data into SiteWise. This has now been corrected such that SourceTimestamp is correctly set in addition to server timestamp.
- \* The HTTP-C north plugin has been updated to support primary and secondary destinations. It will automatically failover to the secondary if the primary becomes unavailable. Fail back will occur either when the secondary becomes unavailable or the plugin is restarted.

- Bug Fix:

- \* An issue with different versions of the libmodbus library prevented the modbus-c plugin building on Moxa gateways, this has now been resolved.
- \* An issue with building the MQTT notification plugin on CentOS/RedHat platforms has been resolved. This plugin now builds correctly on those platforms.
- \* The modbus plugin has been enhanced to support Modbus over IPv6, also request timeout has been added as a configuration option. There have been improvements to the error handling also.
- \* The DNP3 south plugin incorrectly treated all data as strings, this meant it was not easy to process the data with generic plugins. This has now been resolved and data is treated as floating point or integer values.
- \* The OMF north plugin previously reported the incorrect version information. This has now been resolved.
- \* A memory issue with the python35 filter integration has been resolved.
- \* Packaging conflicts between plugins that used the same additional libraries have been resolved to allow both plugins to be installed on the same machine. This issue impacted the plugins that used MQTT as a transport layer.
- \* The OPC/UA north plugin did not correctly handle the types for integer data, this has now been resolved.
- \* The OPCUA south plugin did not allow subscriptions to integer node ids. This has now been added.
- \* A problem with reading multiple modbus input registers into a single value has been resolved in the ModbusC plugin.
- \* OPC/UA north nested objects did not always generate unique node IDs in the OPC/UA server. This has now been resolved.



## 18.2.4 v1.8.2

Release Date: 2020-11-03

- **Fledge Core**

- Bug Fix:

- \* Following the release of a new version of a Python package the 1.8.1 release was no longer installable. This issue is resolved by the 1.8.2 patch release of the core package. All plugins from the 1.8.1 release will continue to work with the 1.8.2 release.

## 18.2.5 v1.8.1

Release Date: 2020-07-08

- **Fledge Core**

- New Features:

- \* Support has been added for the deployment on Moxa gateways running a variant of Debian 9 Stretch.
    - \* The purge process has been improved to also purge the statistics history and audit trail of the system. New configuration parameters have been added to manage the amount of data to be retain for each of these.
    - \* An issue with installing on the Mendel Day release on Google's Coral boards has been resolved.
    - \* The REST API has been expanded to allow an API call to be made to set the repository from which new packages will be pulled when installing plugins via the API and GUI.
    - \* A problem with the service discovery failing to respond correctly after it had been running for a short while has been rectified. This allows external micro services to now correctly discover the core micro service.
    - \* Details for making contributions to the Fledge project have been added to the source repository.
    - \* The support bundle has been improved to include more information needed to diagnose issues with sending data to PI Servers
    - \* The REST API has been extended to add a new call that will return statistics in terms of rates rather than absolute values.
    - \* The documentation has been updated to include guidance on setting up package repositories for installing the software and plugins.

- Bug Fix:

- \* If JSON type configuration parameters were marked as mandatory there was an issue that prevented the update of the parameters. This has now been resolved.
    - \* After changing storage engine from sqlite to Postgres using the configuration option in the GUI or via the API, the new storage engine would incorrectly report itself as sqlite in the API and user interface. This has now been resolved.
    - \* External micro-services that restarted without a graceful shutdown would fail to register with the service registry as nothing was able to unregister the failed service. This has now been relaxed to allow the recovered service to be correctly registered.
    - \* The configuration of the storage system was previously not available via the GUI. This has now been resolved and the configuration can be viewed in the Advanced category of the configuration user interface. Any changes made to the storage configuration will only take effect on the next restart of



Fledge. This allows administrators to change the storage plugins used without the need to edit the storage.json configuration file.

- **GUI**

- Bug Fix:

- \* An improvement to the user experience for editing password in the GUI has been implemented that stops the issue with passwords disappearing if the input field is clicked.
    - \* Password validation was not correctly occurring in the GUI wizard that adds south plugins. This has now be rectified.

- **Plugins**

- New Features:

- \* The Modbus plugin did not gracefully handle interrupted reads of data from modes TCP devices during the bulk transfer of data. This would result in assets missing certain data points and subsequent issues in the north systems that received those assets getting changes in the asset data type. This was a particular issue when dealign with the PI Web API and would result in excessive types being created. The Modbus plugin now detects the issues and takes action to ensure complete assets are read.
    - \* A new image processing plugin, south human detector, that uses the Google Tensor Flow machine learning platform has been added to the Fledge-iot project.
    - \* A new Python plugin has been added that can send data north to a Kafka system.
    - \* A new south plugin has been added for the Dynamic Ratings B100 Electronic Temperature Monitor used for monitoring the condition of electricity transformers.
    - \* A new plugin has been contributed to the project by Nexcom that implements the SAE J1708 protocol for accessing the ECU's of heavy duty vehicles.
    - \* An issue with missing dependencies on the Coral Mendel platform prevent 1.8.0 packages installing correctly without manual intervention. This has now been resolved.
    - \* The image recognition plugin, south-human-detector, has been updated to work with the Google Coral board running the Mendel Day release of Linux.

- Bug Fix:

- \* A missing dependency in v1.8.0 release for the package fledge-south-human-detector meant that it could not be installed without manual intervention. This has now been resolved.
    - \* Support has been added to the south-human-detector plugin for the Coral Camera module in addition to the existing support for USB connected cameras.
    - \* An issue with installation of the external shared libraries required by the USB4704 plugin has been resolved.

### 18.2.6 v1.8.0

Release Date: 2020-05-08

- **Fledge Core**

- New Features:

- \* Documentation has been added for the use of the SQLite In Memory storage plugin.
    - \* The support bundle functionality has been improved to include more detail in order to aid tracking down issues in installations.



- \* Improvements have been made to the documentation of the OMF plugin in line with the enhancements to the code. This includes the documentation of OCS and EDS support as well as PI Web API.
  - \* An issue with forwarding data between two Fledge instances in different time zones has been resolved.
  - \* A new API entry point has been added to the Fledge REST API to allow the removal of plugin packages.
  - \* The notification service has been updated to allow for the delivery of multiple notifications in parallel.
  - \* Improvements have been made to the handling of asset codes within the buffer in order to improve the ingest performance of Fledge. This is transparent to all services outside of the storage service and has no impact on the public APIs.
  - \* Extra information has been added to the notification trigger such that trigger time and the asset that triggered the notification is included.
  - \* A new configuration item type of “northTask” has been introduced. It allows the user to enter the name of a northTask in the configuration of another category within Fledge.
  - \* Data on multiple assets may now be requested in a single call to the asset growing API within Fledge.
  - \* An additional API has been added to the asset browser to allow time bucketed data to be returned for multiple data points of multiple assets in a single call.
  - \* Support has been added for nested readings within the reading data.
  - \* Messages about exceeding the configured latency of the south service may be repeated when the latency is above the configured value for a period of time. These have now been replaced with a single message when the latency is exceeded and another when the condition is cleared.
  - \* The feedback provided to the user when a configuration item is set to an invalid value has been improved.
  - \* Configuration items can now be marked as mandatory, this improves the user experience when configuring plugins.
  - \* A new configuration item type, code, has been added to improve the user experience when adding code snippets in configuration data.
  - \* Improvements have been made to the caching of configuration data within the core of Fledge.
  - \* The logging of package installation has been improved.
  - \* Additions have been added to the public API to allow multiple audit log sources to be extracted in a single API call.
  - \* The audit trail has been improved to show all package additions and updates in the audit trail.
  - \* A new API has been added to allow notification plugin packages to be updated.
  - \* A new API has been added to allow filter code versions to be updated.
  - \* A new API call has been added to allow retrieval of reading data over a period of time which is averaged into time buckets within that time period.
  - \* The notification service now supports rule plugins implemented in Python as well as C++.
  - \* Improvements have been made to the checking of configuration items such that minimum, maximum values and string lengths are now checked.
  - \* The plugin developers documentation has been updated to include a description building C/C++ south plugins.
- Bug Fix:



- \* Improvements have been made to the generation of the support bundle.
- \* An issue in the reporting of the task names in the fledge status script has been resolved.
- \* The purge by size (number of readings) would remove all data if the number of rows to retain was less than 1000, this has now been resolved.
- \* On occasions plugins would disappear from the list of available plugins, this has now been resolved.
- \* Improvements have been made to the management of the certificate store to ensure the correct files are uploaded to the store.
- \* An expensive and unnecessary test was being performed in the asset browsing API of Fledge. This slowed down the user interface and put load n the server. This has now been removed and has improved the performance of examining the buffered data within the Fledge instance.
- \* The FogBench utility used to send data to Fledge has been updated in line with new Python packages for the CoAP protocol.
- \* Configuration category relationships were not always correctly cleaned up when a filter is deleted, this has now been resolved.
- \* The support bundle functionality has been updated to provide information on the Python processes.
- \* The REST API incorrectly allowed configuration categories with a blank name to be created. This has now been prevented.
- \* Validation of minimum and maximum configuration item values was not correctly performed in the REST API, this has now been resolved.
- \* Nested objects within readings could cause the storage engine to fail and those readings to not be stored. This has now been resolved.
- \* On occasion shutting down a service may fail if the filters for that service have not been activated, this has now been resolved.
- \* An issue that cause notifications for asset whose names contain special characters has been resolved.
- \* The asset tracker was not correctly adding entries to the asset tracker, this has now been resolved.
- \* An intermittent issue that prevented the notification service being enabled on the Buster release on Raspberry Pi has been resolved.
- \* An intermittent problem that would prevent the north sending process to fail has been resolved.
- \* Performance improvements have been made to the installation of new packages from the package repository from within the Fledge API and user interface.
- \* It is now possible to reuse the name of a north process after deleting one with the same name.
- \* The incorrect HTTP error code is returned by the asset summary API call if an asset does not exist, this has now been resolved.
- \* Deleting and recreating a south service may cause errors in the log to appear. These have now been resolved.
- \* The SQLite and SQLiteInMemory storage engines have been updated to enable a purge to be defined that reduces the number of readings to a specified value rather than simply allowing a purge by the age of the data. This is designed to allow tighter controls on the size of the buffer database when high frequency data in particular is being stored within the Fledge buffer.

- **GUI**

- New Features:



- \* The user interface for viewing logs has been improve to allow filtering by service and task. A search facility has also been added.
  - \* The requirement that a key file is uploaded with every certificate file has been removed from the graphical user interface as this is not always true.
  - \* The performance of adding a new notification via the graphical user interface has been improved.
  - \* The feedback in the graphical user interface has been improved when installation of the notification service fails.
  - \* Installing the Fledge graphical user interface on OSX platforms fails due to the new version of the brew package manager. This has now been resolved.
  - \* Improve script editing has been added to the graphical user interface.
  - \* Improvements have been made to the user interface for the installations and enabling of the notification service.
  - \* The notification audit log user interface has been improved in the GUI to allow all the logs relating to notifications to be viewed in a single screen.
  - \* The user interface has been redesigned to make better use of the screen space when editing south and north services.
  - \* Support has been added to the graphical user interface to determine when configuration items are not valid based on the values of other items These items that are not valid in the current configuration are greyed out in the interface.
  - \* The user interface now shows the version of the code in the settings page.
  - \* Improvements have been made to the user interface layout to force footers to stay at the bottom of the screen.
- Bug Fix:
- \* Improvements have been made to the zoom and pan options within the graph displays.
  - \* The wizard used for the creation of new notifications in the graphical user interface would loose values when going back and forth between pages, this has now been resolved.
  - \* A memory leak that was affecting the performance of the graphical user interface has been fixed, improving performance of the interface.
  - \* Incorrect category names may be displayed int he graphical user interface, this has now be resolved.
  - \* Issues with the layout of the graphical user interface when viewed on an Apple iPad have been resolved.
  - \* The asset graph in the graphical user interface would sometimes not resize to fit the screen correctly, this has now been resolved.
  - \* The “Asset & Readings” option in the graphical user interface was initially slow to respond, this has now been improved.
  - \* The pagination of audit logs has bene improved when multiple sources are displayed.
  - \* The counts in the user interface for notifications have been corrected.
  - \* Asset data graphs are not able to handle correctly the transition between one day and the next. This is now resolved.

- **Plugins**

- New Features:



- \* The existing set of OMF north plugins have been rationalised and replaced by a single OMF north plugin that is able to support the connector rely, PI Web API, EDS and OCS.
- \* When a Modbus TCP connection is closed by the remote end we fail to read a value, we then reconnect and move on to read the next value. On device with short timeout values, smaller than the poll interval, we fail the same reading every time and never get a value for that reading. The behaviour has been modified to allow us to retry reading the original value after re-establishing the connection.
- \* The OMF north plugin has been updated to support the released version of the OSIsoft EDS product as a destination for data.
- \* New functionality has been added to the north data to PI plugin when using PI Web API that allows the location in the PI Server AF hierarchy to be defined. A default location can be set and an override based on the asset name or metadata within the reading. The data may also be placed in multiple locations within the AF hierarchy.
- \* A new notification delivery plugin has been added that allows a north task to be triggered to send data for a period of time either side of the notification trigger event. This allows conditional forwarding of large amounts of data when a trigger event occurs.
- \* The asset notification delivery plugin has been updated to allow creation of new assets both for notifications that are triggered and/or cleared.
- \* The rate filter now allows the termination of sending full rate data either by use of an expression or by specifying a time in milliseconds.
- \* A new simple Python filter has been added that calculates an exponential moving average,
- \* Some typos in the OPCUA south and north plugin configuration have been fixed.
- \* The OPCUA north plugin has been updated to support nested reading objects correctly and also to allow a name to be set for the OPCUA server. These have also been some stability fixes in the underlying OPCUA layer used by this and the south OPCUA plugin.
- \* The modbus map configuration now supports byte swapping and word swapping by use of the `{{swap}}` property of the map. This may take the values `{{bytes}}`, `{{words}}` or `{{both}}`.
- \* The people detection machine learning plugin now supports RTSP streams as input.
- \* The option list items in the OMF plugin have been updated to make them more user friendly and descriptive.
- \* The threshold notification rule has been updated such that the unused fields in the configuration now correctly grey out in the GUI dependent upon the setting of the window type or single item asset validation.
- \* The configuration of the OMF north plugin for connecting to the PI Server has been improved to give the user better feedback as to what elements are valid based on choice of connection method and security options chosen.
- \* Support has been added for simple Python code to be entered into a filter that does not require all of the support code. This is designed to allow a user to very quickly develop filters with limited programming.
- \* Support has been added for filters written entirely in Python, these are full featured filters as supported by the C++ filtering mechanism and include dynamic reconfiguration.
- \* The fledge-filter-expression filter has been modified to better deal with streams which contain multiple assets. It is now possible to use the syntax `<assetName>.<datapointName>` in an expression in addition to the previous `<datapointName>`. The result is that if two assets in the data stream have the same data point names it is now possible to differentiate between them.



- \* A new plugin to collect variables from Beckhoff PLC's has been written. The plugin uses the Twin-CAT 2 or TwinCAT 3 protocols to collect specified variable from the running PLC.
- Bug Fix:
- \* An issue in the sending of data to the PI server with large values has been resolved.
  - \* The playback south plugin was not correctly replaying timestamps within the file, this has now been resolved.
  - \* Use of the asset filter in a north task could result in the north task terminating. This has now resolved.
  - \* A small memory leak in the south service statistics handling code was impacting the performance of the south service, this is now resolved.
  - \* An issue has been discovered in the Flir camera plugin with the validity attribute of the spot temperatures, this has now been resolved.
  - \* It was not possible to send data for the same asset from two different Fledge's into the PI Server using PI Web API, this has now been resolved.
  - \* The filter Fledge RMS Trigger was not able to be dynamically reconfigured, this has now been resolved.
  - \* If a filter in the north sending process increased the number of readings it was possible that the limit of the number of readings sent in a single block . The sending process will now ensure this can not happen.
  - \* RMS filter plugin was not able to be dynamically reconfigured, this has now been resolved.
  - \* The HTTP South plugin that is used to receive data from another Fledge instance may fail with some combinations of filters applied to the service. This issue has now been resolved.
  - \* The rule filter may give errors if expressions have variables not satisfied in the reading data. Under some circumstances it has been seen that the filter fails to process data after giving this error. This has been resolved by changes to make the rate filter more robust.
  - \* Blank values for asset names in the south service may cause the service to become unresponsive. Blank asset names have now been correctly detected, asset names are required configuration values.
  - \* A new version of the driver software for the USB-4704 Data Acquisition Module has been released, the plugin has been updated to use this driver version.
  - \* The OPCUA North plugin might report incorrect counts for sent readings on some platforms, this has now been resolved.
  - \* The simple Python filter plugin was not adding correct asset tracking data, this has now been updated.
  - \* An issue with the asset filter failing when incorrect configuration was present has been resolved.
  - \* The benchmark plugin now enforces a minimum number of asset of 1.
  - \* The OPCUA plugins are now available on the Raspberry Pi Buster platform.
  - \* Errors that prevented the use of the Postgres storage plugin have been resolved.



## 18.2.7 v1.7.0

Release Date: 2019-08-15

- **Fledge Core**

- New Features:

- \* Added support for Raspbian Buster
- \* Additional, optional flow control has been added to the south service to prevent it from overwhelming the storage service. This is enabled via the throttling option in the south service advanced configuration.
- \* The mechanism for including JSON configuration in C++ plugins has been improved and the macros for the inline coding moved to a standard location to prevent duplication.
- \* An option has been added that allows the system to be updated to the latest version of the system packages prior to installing a new plugin or component.
- \* Fledge now supports password type configuration items. This allows passwords to be hidden from the user in the user interface.
- \* A new feature has been added that allows the logs of plugin or other package installation to be retrieved.
- \* Installation logs for package installations are now retained and available via the REST API.
- \* A mechanism has been added that allows plugins to be marked as deprecated prior to the removal of these plugins in future releases. Running a deprecated plugin will result in a warning being logged, but otherwise the plugin will operate as normal.
- \* The Fledge REST API has been updated to add a new entry point that will allow a plugin to be updated from the package repository.
- \* An additional API has been added to fetch the set of installed services within a Fledge installation.
- \* An API has been added that allows the caller to retrieve the list of plugins that are available in the Fledge package repository.
- \* The /fledge/plugins REST API has been extended to allow plugins to be installed from an APT/RPM repository.
- \* Addition of support for hybrid plugins. A hybrid plugin is a JSON file that defines another plugin to load along with some default configuration for that plugin. This gives a means to create a new plugin by customising the configuration of an existing plugin. An example might be a plugin for a specific modbus device type that uses the generic modbus plugin and a predefined modbus map.
- \* The notification service has been improved to allow the re-trigger time of a notification to be defined by the user on a per notification basis.
- \* A new environment variable, FLEDGE\_PLUGIN\_PATH has been added to allow plugins to be stored in multiple locations or locations outside of the usual Fledge installation directory.
- \* Added support for FLEDGE\_PLUGIN\_PATH environment variable, that would be used for searching additional directory paths for plugins/filters to use with Fledge.
- \* Fledge packages for the Google Coral Edge TPU development board have been made available.
- \* Support has been added to the OMF north plugin for the PI Web API OMF endpoint. The PI Server functionality to support this is currently in beta test.

- Bug Fix/Improvements:



- \* An issue with the notification service becoming unresponsive on the Raspberry Pi Buster release has been resolved.
- \* A debug message was being incorrectly logged as an error when adding a Python south plugin. The message level has now been corrected.
- \* A problem whereby not all properties of configuration items are updated when a new version of a configuration category is installed has been fixed.
- \* The notification service was not correctly honouring the notification types for one shot, toggled and retrigged notifications. This has now be bought in line with the documentation.
- \* The system log was becoming flooded with messages from the plugin discovery utility. This utility now logs at the correct level and only logs errors and warning by default.
- \* Improvements to the REST API allow for selective sets of statistic history to be retrieved. This reduces the size of the returned result set and improves performance.
- \* The order in which filters are shutdown in a pipeline of filters has been reversed to resolve an issue regarding releasing Python interpreters, under some circumstances shutdowns of later filters would fail if multiple Python filters were being used.
- \* The output of the *fledge status* command was corrupt, showing random text after the number of seconds for which fledge has been up. This has now been resolved.

- **GUI**

- New Features:

- \* A new log option has been added to the GUI to show the logs of package installations.
    - \* It is now possible to edit Python scripts directly in the GUI for plugins that load Python snippets.
    - \* A new log retrieval option has been added to the GUI that will show only notification delivery events. This makes it easier for a user to see what notifications have been sent by the system.
    - \* The GUI asset graphs have been improved such that multiple tabs are now available for graphing and tabular display of asset data.
    - \* The GUI menu has been reordered to move the Notifications entry below the South and North entries.
    - \* Support has been added to the Fledge GUI for entry of password fields. Data is obfuscated as it is entered or edited.
    - \* The GUI now shows plugin name and version for each north task defined.
    - \* The GUI now shows the plugin name and version for each south service that is configured.
    - \* The GUI has been updated such that it can install new plugins from the Fledge package repository for south services and north tasks. A list of available packages from the repository is displayed to allow the user to pick from that list. The Fledge instance must have connectivity tot he package repository to allow this feature to succeed.
    - \* The GUI now supports using certificates to authenticate with the Fledge instance.

- Bug Fix/Improvements:

- \* Improved editing of JSON configuration entities in the configuration editor.
    - \* Improvements have been made to the asset browser graphs in the GUI to make better use of the available space to show the graph itself.
    - \* The GUI was incorrectly showing Fledge as down in certain circumstances, this has now been resolved.



- \* An issue in the edit dialog for the north plugin which sometimes prevented the enabled state from being correctly modified has been resolved.
- \* Exported CSV data from the GUI would sometimes be missing column headers, these are now always present.
- \* The exporting of data as a CSV file in the GUI has been improved such that it no longer outputs the readings as a block of JSON, but rather individual columns. This allows the data to be imported into a spreadsheet with ease.
- \* Missing help text has been added for notification trigger and enabled elements.
- \* A number of issues in the filter configuration editor have been resolved. These issues meant that sometimes new values were not honoured or when changes were made with multiple filters in a chain only one filter would be updated.
- \* Under some rare circumstances the GUI asset graph may show incorrect dates, this issue has now been resolved.
- \* The Fledge GUI build and start commands did not work on Windows platforms and preventing the running on those platforms. This has now been resolved and the Fledge GUI can be built and run on Windows platforms.
- \* The GUI was not correctly interpreting the value of the readonly attribute of configuration items when the value was anything other than true. This has been resolved.
- \* The Fledge GUI RPM package had an error that caused installation to fail on some systems, this is now resolved.

### • Plugins

#### – New Features:

- \* A new filter has been created that looks for changes in values and only sends full rate data around the time of those changes. At other times the filter can be configured to send reduced rate averages of the data.
- \* A new rule plugin has been implemented that will create notifications if the value of a data point moves more than a defined percentage from the average for that data point. A moving average for each data point is calculated by the plugin, this may be a simple average or an exponential moving average.
- \* A new south plugin has been created that supports the DNP3 protocol.
- \* A south plugin has been created based on the Google TensorFlow people detection model. It uses a live feed from a video camera and returns data regarding the number of people detected and the position within the frame.
- \* A south plugin based on the Google TensorFlow demo model for people recognition has been created. The plugin reads an image from a file and returns the people co-ordinates of the people it detects within the image.
- \* A new north plugin has been added that creates an OPCUA server based on the data ingested by the Fledge instance.
- \* Support has been added for a Flir Thermal Imaging Camera connected via Modbus TCP. Both a south plugin to gather the data and a filter plugin, to clean the data, have been added.
- \* A new south plugin has been created based on the Google TensorFlow demo model that accepts a live feed from a Raspberry Pi camera and classifies the images.
- \* A new south plugin has been created based on the Google TensorFlow demo model for object detection. The plugin return object count, name position and confidence data.



- \* The change filter has been made available on CentOS and RedHat 7 releases.
- Bug Fix/Improvements:
  - \* Support for reading floating point values in a pair of 16 bit registers has been added to the modbus plugin.
  - \* Improvements have been made to the performance of the modbus plugin when large numbers of contiguous registers are read. Also the addition of support for floating point values in modbus registers.
  - \* Flir south service has been modified to support the Flir camera range as currently available, i.e. a maximum of 10 areas as opposed to the 20 that were previously supported. This has improved performance, especially on low performance platforms.
  - \* The python35 filter plugin did not allow the Python code to add attributes to the data. This has now been resolved.
  - \* The playback south plugin did not correctly take the timestamp data from the CSV file. An option is now available that will allow this.
  - \* The rate filter has been enhanced to accept a list of assets that should be passed through the filter without having the rate of those assets altered.
  - \* The filter plugin python35 crashed on the Buster release on the Raspberry Pi, this has now been resolved.
  - \* The FFT filter now enforces that the number of samples must be a power of 2.
  - \* The ThingSpeak north plugin was not updated in line with changes to the timestamp handling in Fledge, this resulted in a crash when it tried to send data to ThingSpeak. This has been resolved and the cause of the crash also fixed such that now an error will be logged rather than the task crashing.
  - \* The configuration of the simple expression notification rule plugin has been simplified.
  - \* The DHT 11 plugin mistakenly had a dependency on the Wiring PI package. This has now been removed.
  - \* The system information plugin was missing a dependency that would cause it to fail to install on systems that did not already have the package it was depend on installed. This has been resolved.
  - \* The phidget south plugin reconfiguration method would crash the service on occasions, this has now been resolved.
  - \* The notification service would sometimes become unresponsive after calling the notify-python35 plugin, this has now been resolved.
  - \* The configuration options regarding notification evaluation of single items and windows has been improved to make it less confusing to end users.
  - \* The OverMax and UnderMin notification rules have been combined into a single threshold rule plugin.
  - \* The OPCUA south plugin was incorrectly reporting itself as the upcua plugin. This is now resolved.
  - \* The OPCUA south plugin has been updated to support subscriptions both using browse names and Node Id's. Node ID is now the default subscription mechanism as this is much higher performance than traversing the object tree looking at browse names.
  - \* Shutting down the OPCUA service when it has failed to connect to an OPCUA server, either because of an incorrect configuration or the OPCUA server being down resulted in the service crashing. The service now shuts down cleanly.
  - \* In order to install the fledge-south-modbus package on RedHat Enterprise Linux or CentOS 7 you must have configured the epel repository by executing the command:

*sudo yum install epel-release*



- \* A number of packages have been renamed in order to obtain better consistency in the naming and to facilitate the upgrade of packages from the API and graphical interface to Fledge. This will result in duplication of certain plugins after upgrading to the release. This is only an issue of the plugins had been previously installed, these old plugin should be manually removed from the system to alleviate this problem.

The plugins involved are,

- fledge-north-http Vs fledge-north-http-north
- fledge-south-http Vs fledge-south-http-south
- fledge-south-Csv Vs fledge-south-csv
- fledge-south-Expression Vs fledge-south-expression
- fledge-south-dht Vs fledge-south-dht11V2
- fledge-south-modbus Vs fledge-south-modbus

### 18.2.8 v1.6.0

Release Date: 2019-05-22

- **Fledge Core**

- New Features:

- \* The scope of the Fledge certificate store has been widened to allow it to store .pem certificates and keys for accessing cloud functions.
    - \* The creation of a Docker container for Fledge has been added to the packaging options for Fledge in this version of Fledge.
    - \* Red Hat Enterprise Linux packages have been made available from this release of Fledge onwards. These packages include all the applicable plugins and notification service for Fledge.
    - \* The Fledge API now supports the creation of configuration snapshots which can be used to create configuration checkpoints and rollback configuration changes.
    - \* The Fledge administration API has been extended to allow the installation of new plugins via API.

- Improvements/Bug Fix:

- \* A bug that prevents multiple Fledge's on the same network being discoverable via multicast DNS lookup has been fixed.
    - \* Set, unset optional configuration attributes

- **GUI**

- New Features:

- \* The Fledge Graphical User Interface now has the ability to show sets of graphs over a time period for data such as the spectrum analysis produced but the Fast Fourier transform filter.
    - \* The Fledge Graphical User Interface is now available as an RPM file that may be installed on Red Hat Enterprise Linux or CentOS.

- Improvements/Bug Fix:

- \* Improvements have been made to the Fledge Graphical User Interface to allow more control of the time periods displayed in the graphs of asset values.



- \* Some improvements to screen layout in the Fledge Graphical User Interface have been made in order to improve the look and reduce the screen space used in some of the screens.
- \* Improvements have been made to the appearance of dropdown and other elements with the Fledge Graphical User Interface.

- **Plugins**

- **New Features:**

- \* A new threshold filter has been added that can be used to block onward transmission of data until a configured expression evaluates too true.
- \* The Modbus RTU/TCP south plugin is now available on CentOS 7 and RHEL 7.
- \* A new north plugin has been added to allow data to be sent the Google Cloud Platform IoT Core interface.
- \* The FFT filter now has an option to output raw frequency spectra. Note this can not be accepted into all north bound systems.
- \* Changed the release status of the FFT filter plugin.
- \* Added the ability in the modbus plugin to define multiple registers that create composite values. For example two 16 bit registers can be put together to make one 32 bit value. This is done using an array of register values in a modbus map, e.g. {"name": "rpm", "slave": 1, "register": [33, 34], "scale": 0.1, "offset": 0}. Register 33 contains the low 16 bits of the RPM and register 34 the high 16 bits of the RPM.
- \* Addition of a new Notification Delivery plugin to send notifications to a Google Hangouts chat-room.
- \* A new plugin has been created that uses machine learning based on Google's TensorFlow technology to classify image data and populate derived information the north side systems. The current TensorFlow model in use will recognise hard written digits and populate those digits. This plugin is currently a proof of concept for machine learning.

- **Improvements/Bug Fix:**

- \* Removal of unnecessary include directive from Modbus-C plugin.
- \* Improved error reporting for the modbus-c plugin and added documentation on the configuration of the plugin.
- \* Improved the subscription handling in the OPCUA south plugin.
- \* Stability improvements have been made to the notification service, these related to the handling of dynamic reconfigurations of the notifications.
- \* Removed erroneous default for script configuration option in Python35 notification delivery plugin.
- \* Corrected description of the enable configuration item.



### 18.2.9 v1.5.2

Release Date: 2019-04-08

- **Fledge Core**

- **New Features:**

- \* Notification service, notification rule and delivery plugins
    - \* Addition of a new notification delivery plugin that will create an asset reading when a notification is delivered. This can then be sent to any system north of the Fledge instance via the usual mechanisms
    - \* Bulk insert support for SQLite and Postgres storage plugins

- **Enhancements / Bug Fix:**

- \* Performance improvements for SQLite storage plugin.
    - \* Improved performance of data browsing where large datasets have been acquired
    - \* Optimized statistics history collection
    - \* Optimized purge task
    - \* The readings count shown on GUI and south page and corresponding API endpoints now shows total readings count and not what is currently buffered by Fledge. So these counts don't reduce when purge task runs
    - \* Static data in the OMF plugin was not being correctly taken from the plugin configuration
    - \* Reduced the number of informational log messages being sent to the syslog

- **GUI**

- **New Features:**

- \* Notifications UI

- **Bug Fix:**

- \* Backup creation time format

### 18.2.10 v1.5.1

Release Date: 2019-03-12

- **Fledge Core**

- Bug Fix: plugin loading errors

- **GUI**

- Bug Fix: uptime shows up to 24 hour clock only



## 18.2.11 v1.5.0

Release Date: 2019-02-21

- **Fledge Core**
  - Performance improvements and Bug Fixes
  - Introduction of Safe Mode in case Fledge is accidentally configured to generate so much data that it is overwhelmed and can no longer be managed.
- **GUI**
  - re-organization of screens for Health, Assets, South and North
  - bug fixes
- **South**
  - Many Performance improvements, including conversion to C++
  - Modbus plugin
  - many other new south plugins
- **North**
  - Compressed data via OMF
  - Kafka
- **Filters:** Perform data pre-processing, and allow distributed applications to be built on Fledge.
  - Delta: only send data upon change
  - Expression: run a complex mathematical expression across one or more data streams
  - Python: run arbitrary python code to modify a data stream
  - Asset: modify Asset metadata
  - RMS: Generate new asset with Root Mean Squared and Peak calculations across data streams
  - FFT (beta): execute a Fast Fourier Transform across a data stream. Valuable for Vibration Analysis
  - Many others
- **Event Notification Engine (beta)**
  - Run rules to detect conditions and generate events at the edge
  - Default Delivery Mechanisms: email, external script
  - Fully pluggable, so custom Rules and Delivery Mechanisms can be easily created
- **Debian Packages for All Repo's**



### 18.2.12 v1.4.1

Release Date: 2018-10-10

### 18.2.13 v1.4.0

Release Date: 2018-09-25

### 18.2.14 v1.3.1

Release Date: 2018-07-13

#### Fixed Issues

- **Open File Descriptors**
  - **open file descriptors:** Storage service did not close open files, leading to multiple open file descriptors

### 18.2.15 v1.3

Release Date: 2018-07-05

#### New Features

- **Python version upgrade**
  - **python 3 version:** The minimal supported python version is now python 3.5.3.
- **aiohttp python package version upgrade**
  - **aiohttp package version:** aiohttp (version 3.2.1) and aiohttp\_cors (version 0.7.0) is now being used
- **Removal of south plugins**
  - **coap:** coap south plugin was moved into its own repository <https://github.com/fledge-iot/fledge-south-coap>
  - **http:** http south plugin was moved into its own repository <https://github.com/fledge-iot/fledge-south-http>

#### Known Issues

- **Issues in Documentation**
  - **plugin documentation:** testing Fledge requires user to first install southbound plugins necessary (CoAP, http)



## 18.2.16 v1.2

Release Date: 2018-04-23

### New Features

- **Changes in the REST API**
  - **ping Method:** the ping method now returns uptime, number of records read/sent/purged and if Fledge requires REST API authentication.
- **Storage Layer**
  - **Default Storage Engine:** The default storage engine is now SQLite. We provide a script to migrate from PostgreSQL in 1.1.1 version to 1.2. PostgreSQL is still available in the main repository and package, but it will be removed to an operate repository in future versions.
- **Admin and Maintenance Scripts**
  - **fledge status:** the command now shows what the ping REST method provides.
  - **setenv script:** a new script has been added to simplify the user interaction. The script is in *\$FLEDGE\_ROOT/extras/scripts* and it is called *setenv.sh*.
  - **fledge service script:** a new service script has been added to setup Fledge as a service. The script is in *\$FLEDGE\_ROOT/extras/scripts* and it is called *fledge.service*.

### Known Issues

- **Issues in the REST API**
  - **asset method response:** the `asset` method returns a JSON object with asset code named `asset_code` instead of `assetCode`
  - **task method response:** the `task` method returns a JSON object with unexpected element `"exitCode"`

## 18.2.17 v1.1.1

Release Date: 2018-01-18

### New Features

- **Fixed aiohttp incompatibility:** This fix is for the incompatibility of *aiohttp* with *yaml*, discovered in the previous version. The issue has been fixed.
- **Fixed avahi-daemon issue:** Avahi daemon is a pre-requisite of Fledge, Fledge can now run as a snap or build from source without avahi daemon installed.



### Known Issues

- **PostgreSQL with Snap:** the issue described in version 1.0 still persists, see [Known Issues](#) in v1.0.

### 18.2.18 v1.1

Release Date: 2018-01-09

### New Features

- **Startup Script:**
  - `fledge start` script now checks if the Core microservice has started.
  - `fledge start` creates a `core.err` file in `$FLEDGE_DATA` and writes the stderr there.

### Known Issues

- **Incompatibility between aiohttp and yarl when Fledge is built from source:** in this version we use *aiohttp* 2.3.6 (). This version is incompatible with updated versions of *yarl* (0.18.0+). If you intend to use this version, change the requirements for *aiohttp* for version 2.3.8 or higher.
- **PostgreSQL with Snap:** the issue described in version 1.0 still persists, see [Known Issues](#) in v1.0.

### 18.2.19 v1.0

Release Date: 2017-12-11

### Features

- All the essential microservices are now in place: *Core*, *Storage*, *South*, *North*.
- Storage plugins available in the main repository:
  - **Postgres:** The storage layer relies on PostgreSQL for data and metadata
- South plugins available in the main repository:
  - **CoAP Listener:** A CoAP microservice plugin listening to client applications that send data to Fledge
- North plugins available in the main repository:
  - **OMF Translator:** A task plugin sending data to OSIsoft PI Connector Relay 1.0

### Known Issues

- **Startup Script:** `fledge start` does not check if the Core microservice has started correctly, hence it may report that “Fledge started.” when the process has died. As a workaround, check with `fledge status` the presence of the Fledge microservices.
- **Snap Execution on Raspbian:** there is an issue on Raspbian when the Fledge snap package is used. It is an issue with the snap environment, it looks for a shared object to preload on Raspbian, but the object is not available. As a workaround, a superuser should comment a line in the file `/etc/ld.so.preload`. Add a # at the beginning of this line: `/usr/lib/arm-linux-gnueabi/hf/libarmmmem.so`. Save the file and you will be able to immediately use the snap.



- **OMF Translator North Plugin for Fledge Statistics:** in this version the statistics collected by Fledge are not sent automatically to the PI System via the OMF Translator plugin, as it is supposed to be. The issue will be fixed in a future release.
- **Snap installed in an environment with an existing version of PostgreSQL:** the Fledge snap does not check if another version of PostgreSQL is available on the machine. The result may be a conflict between the tailored version of PostgreSQL installed with the snap and the version of PostgreSQL generally available on the machine. You can check if PostgreSQL is installed using the command `sudo dpkg -l | grep 'postgres'`. All packages should be removed with `sudo dpkg --purge <package>`.







## DOWNLOADS

### 19.1 Packages

Packages for a number of different Linux platforms are available for both Intel and Arm architectures via the Dianomic web site's download page.

- 

### 19.2 Download/Clone from GitHub

Fledge and the Fledge tools are on GitHub. You can view and download them here:

- **Fledge:** This is the main project for the Fledge platform.
- **Fledge GUI:** This is an experimental GUI that connects to the Fledge REST API to configure and administer the platform and to retrieve the data buffered in it.

There are many south, north, and filter plugins available on github:







## GLOSSARY

The following are a set of definitions for terms used within the Fledge documentation and code, these are designed to be an aid to understanding some of the principles behind Fledge and improve the comprehension of the documentation by ensuring all readers have a common understanding of the terms used. If you feel any terms are missing or not fully explained please raise an issue against, or contribute to, the documentation in the .

**Asset** A representation of a set of device or set of values about a device or entity that is being monitored and possibly controlled by Fledge. It may also be used to represent a subset of a device. These values are a collection of *Datapoints* that are the actual values. An asset contains a unique name that is used to reference the data about the asset. An asset is an abstract concept and has no real implementation with the fledge code, instead a *reading* is used to represent the state of an asset at a point in term. The phase asset is used to represent a time series collection of 0 or more *readings*.

**Control Service** An optional microservice that is used by the control features of Fledge to route control messages from the various sources of control and send them to the *south service* which implements the control path for the *assets* under control.

**Core Service** The *service* within Fledge that is responsible for the oversight of all the other services. It provides configuration management, monitoring, registration and routing services. It is also responsible for the public API into the Fledge system and the execution of periodic tasks such as *purge*, statistics and backup.

**Datapoint** A datapoint is a container for data, each datapoint represents a value that is known about an asset and has a name for that value and the value itself. Values may be one of many types; simpler scalar values, alpha numeric strings, arrays of scalar values, images, arbitrary binary objects or a collection of datapoints.

**Filter** A combination of a *Filter Plugin* and the configuration that makes that filter perform the processing that is required of it.

**Filter Plugin** A filter plugin is a *plugin* that implements an operation on one or more *reading* as it passes through the Fledge system. This processing may add, remove or augment the data as it passes through Fledge. Filters are arrange as linear *pipelines* in either the *south service* as data is ingested into Fledge or the *north services* and *tasks* as data is passed upstream to the systems that receive data from Fledge.

**Microservice** A microservice is a small service that implements parts of the Fledge functionality. They are also referred to as *services*.

**Notification Delivery Plugin** A notification delivery plugin is used by the *notification service* to delivery notifications when a *notification rule* triggers. A notification delivery plugin may send notification data to external systems, trigger internal Fledge operations or create *reading* data within the Fledge *storage service*.

**Notification Rule Plugin** A notification rule plugin is used by the notification service to determine if a notification should be sent. The rule plugin receives *reading* data from the Fledge *storage service*, evaluates a rule against that data and returns a triggered or cleared state to the notification service.

**Notification Service** An optional *service* within Fledge that is responsible for the execution and delivery of notifications when events occurs in the data that is being ingested into Fledge.



**North** An abstract term for any service or system to which Fledge sends data that it has ingested. Fledge may also receive control messages from the north as well as from other locations.

**North Plugin** A *plugin* that implements the connection to an upstream system. North plugins are responsible to both implement the communication to the north systems and also the translation from internal data representations to the representation used in the external system.

**North Service** A *service* responsible for connections upstream from Fledge. These are usually systems that will receive data that Fledge has ingested and/or processed. There may also be control data flows that operate from the north systems into the Fledge system.

**North Task** A *task* that is run to send data to upstream systems from Fledge. It is very similar in operation and concept to a *north service*, but differs from a north service in that it does not always run, it is scheduled using a time based schedule and is designed for situations where connection to the upstream system is not always available or desirable.

**Pipeline** A linear collection of zero or more *filters* connected between with the *south plugin* that ingests data and the *storage service*, or between the *storage service* and the *north plugin* as data exits Fledge to be sent to upstream systems.

**Plugin** A dynamically loadable code fragment that is used to enhance the capabilities of Fledge. These plugins may implement a *south* interface to devices and systems, a *north* interface to systems that receive data from Fledge, a *storage plugin* used to buffer *readings*, a *filter plugin* used to process data, a *notification rule* or *notification delivery* plugin. Plugins have well defined interfaces, they can be written by third parties without recourse to modifying the Fledge services and are shipped externally to Fledge to allow for diverse installations of Fledge. Plugins are the major route by which Fledge is customized for individual use cases.

**Purge** The process by which *readings* are removed from the *storage service*.

**Reading** A reading is the presentation of an *asset* at a point in time. It contains the asset name, two timestamps and the collection of *datapoints* that represent the state of the asset at that point in time. A reading has two timestamps to allow for the time to be recorded when Fledge first read the data and also for the device itself to give a time that it sets for when the data was created. Not all devices are capable of reporting timestamps and hence this second timestamp may be the same as the first.

**Service** Fledge is implemented as a set of services, each of which runs constantly and implements a subset of the system functionality. There are a small set of fixed services, such as the *core service* or *storage service*, optional services for enhanced functionality, such as the *notification service* and *control service*. There are also a set of non-fixed services of various types used to interact with downstream or *south* devices and upstream or *north* systems.

**South** An abstract term for any device or service from which Fledge ingests data or over which Fledge exerts control.

**South Service** A *service* responsible for communication with a device or service from which Fledge is ingesting data. Each south service connects to a single device and can collect data from that device and optionally send control signals to that device. A south service may represent one or more *assets*.

**South Plugin** A south plugin is a *plugin* that implements the interface to a device or system from which Fledge is collecting data and optionally to which Fledge is sending control signals.

**Storage Service** A *microservice* that implements either permanent or transient storage services used to both buffer *readings* within Fledge and also to store Fledge's configuration information. The storage services use either one or two *storage plugins* to store the configuration data and the *readings* data.

**Storage Plugin** A *plugin* that implements the storage requirements of the Fledge *storage service*. A plugin may implement the storage of both configuration and *readings* or it may just implement *readings* storage. In this latter case Fledge will use two storage plugins, one to store the configuration and the other to store the readings.

**Task** A task implements functionality that only runs for specific times within Fledge. It is used to initiate periodic operations that are not required to be always running. Amongst the tasks that form part of Fledge are the *purge task*, *north tasks*, backup and statistics gathering tasks.



## INDEX

### A

Asset, [533](#)

### C

Control Service, [533](#)

Core Service, [533](#)

### D

Datapoint, [533](#)

### F

Filter, [533](#)

Filter Plugin, [533](#)

### M

Microservice, [533](#)

### N

North, [534](#)

North Plugin, [534](#)

North Service, [534](#)

North Task, [534](#)

Notification Delivery Plugin, [533](#)

Notification Rule Plugin, [533](#)

Notification Service, [533](#)

### P

Pipeline, [534](#)

Plugin, [534](#)

Purge, [534](#)

### R

Reading, [534](#)

### S

Service, [534](#)

South, [534](#)

South Plugin, [534](#)

South Service, [534](#)

Storage Plugin, [534](#)

Storage Service, [534](#)

### T

Task, [534](#)