intdash Edge Agent Developer Guide

intdash Edge Agent Version 1.23.0

6th edition (October 2022)

aptpod

Rev. 1cc92fc

Table of contents

01 Introduction

This document describes how to use intdash Edge Agent (hereinafter referred to as Agent).

Important:
- This document is for general informational purposes only. Specifications in this document are subject to change without notice and are not guaranteed.
- Screenshots used in descriptions are examples. Some displays or procedures might differ depending on your environment and application version.

Note: Company names, service names, and product names mentioned in this document are generally registered trademarks or trademarks of their respective owners. Trademark symbols "™" and "®" are omitted in the text, figures, and tables.

Attention: This document has been translated using machine translation services and may contain inaccuracies and translation errors. Please also refer to the official version in Japanese.

1.1 What is Agent?

Agent is agent software that sends and receives data to and from the intdash server.

The Agent can stream frequently occurring time series data to the intdash server with low latency. Data that could not be sent due to a failure such as a network line disconnection will be automatically retransmitted. This enables the data to be fully recovered to the intdash server.

1.2 Main features

- Streaming time series data
- Automatic retransmission of lost data
- Filtering and sampling of time series data
- Saving the acquired time series data as a dump file

1.3 System requirements

- Supported platforms
  - Linux on AMD64 architecture
  - Raspbian on Raspberry Pi
  - NVIDIA L4T on NVIDIA Jetson
- Minimum hardware requirements
  - Intel Atom processor E3815, 1.46GHz or higher
- Recommended hardware requirements
  - Multi-core CPU
  - 2 GB or more memory
  - SSD

Important: If the amount of data is large and the CPU load is too high, data may be lost.
The guideline for the amount of data that can be processed without loss is as follows. (The following are approximate values that apply when you are using a device connector with a small processing load. The limit performance will increase or decrease depending on the processing load of the device connector.)

- **When using VTC 1910-S (Intel Atom E3815 1.46GHz)**
  - Sending small data frequently: about 8B(bytes) / unit, 24000 units / second
  - Sending large data infrequently: about 0.98MB / unit, 10 units / second
- **When using Raspberry Pi 4 Model B**
  - Sending small data frequently: about 8B(bytes) / unit, 100000 units / second
  - Sending large data infrequently: about 0.98MB / unit, 80 units / second

If you anticipate a heavy load on the CPU, we recommend that you perform test measurements in advance. After performing test measurements, execute the following command on the edge to check the messages in syslog.

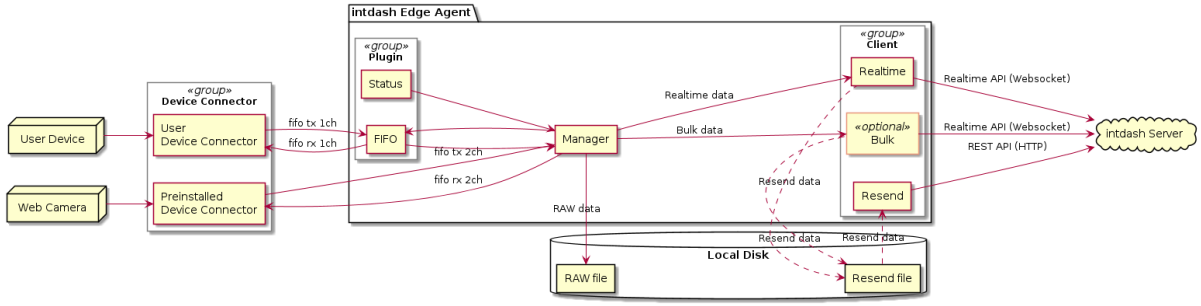
```
$ cat /var/log/syslog | grep -e"ring is full" -e "data buffer is overflow" | grep -v PacketSender
```

If you see a message `ring is full` or `data buffer is overflow` in the syslog (except for the PacketSender message), some data has been lost between the device connector and the agent.

1.4 System configuration

The Agent consists of software modules: Device Connectors, a manager, and clients.

- The Device Connector receives data from the device.
- The manager performs various processes such as filtering and sampling.
- The client sends the time series data to the intdash server.



1.4.1 Device connector

A Device connector is software for connecting various devices and the Agent.

The default Device Connector, the `intdash-edge-logger`, and 6 settings are installed with the Agent.

- v4lh264
- gstreamer_h264
- mjpeg
- nmea
- socketcan
- canopen

Configuration is required to use the Device Connector. See Settings for using the pre-installed Device Connector (p. 12) for instructions on how to configure the Device Connectors above.

Users can connect devices that are not supported by `intdash-edge-logger` by providing a new Device

Connector. See Add your own Device Connector (p. 31) for information on how to add Device Connectors other than `intdash-edge-logger`.

Note: The word "logger" is sometimes used in configuration files, but "logger" refers to the Device Connector.

1.4.2 Plugin

A plugin is an internal software module that interfaces the Device Connector with the Agent. There are the following two types.

FIFO plugin
Sends and receives data between the Agent and a Device Connector via a FIFO (named pipe).

Status plugin
This plugin does not connect to a Device Connector, but collects status information.

Note:
- Plugins need to be specified in `loggers[].details.plugin` in the configuration file. See Device connector settings (p. 57) for more information.
- The Device Connector `intdash-edge-logger` provided by Aptpod does not use the FIFO plugin. Therefore, when using `intdash-edge-logger`, it is not necessary to define `plugin` in the configuration file.

1.4.3 Manager

The manager is the software module at the center of the Agent. It manages the start and stop of other modules, aggregates the data collected from the Device Connectors, processes filtering and sampling, and exports RAW data.

1.4.4 Client

A client in the Agent is a software module responsible for communication between the Agent and the intdash server. A client is responsible for real-time transmission using intdash's Realtime API, transmission in high-efficiency format using intdash REST API, and retransmission.

Realtime client
Sends data in real time using intdash's Realtime API (WebSocket).

Bulk client
Sends data in bulk at regular intervals of several seconds using intdash's Realtime API (WebSocket).

Resend client
Resends the data at regular intervals that the Realtime and Bulk clients were unable to send. Both Realtime API (WebSocket) and REST API can be used to communicate with the intdash server.

Control client
Receives data in real time using intdash's Realtime API (WebSocket).

02 Get started

This section describes how to install the Agent and how to start and stop it.

The installer supports the following environments.

Distribution	Version	Architecture
Ubuntu	22.04(LTS), 20.04(LTS), 18.04(LTS), 16.04(LTS)	amd64, arm64, armhf
Debian	10, 9	amd64
Raspbian	based on Debian 10	armhf

2.1 Installation

In the environment where you want to install the Agent, execute the installer as follows.

For `${DISTRIBUTION}` and `${ARCHITECTURE}` }, select one from the table below.

DISTRIBUTION	ARCHITECTURE
ubuntu	amd64, armhf, arm64
debian	amd64
raspbian	armhf

```
$ sudo apt-get update
$ sudo apt-get install \
    apt-transport-https \
    ca-certificates \
    curl \
    gnupg-agent \
    lsb-release
$ curl -s --compressed \
    "https://repository.aptpod.jp/intdash-edge/linux/${DISTRIBUTION}/gpg" | sudo apt-key add -
$ echo "deb [arch=${ARCHITECTURE}] \
    https://repository.aptpod.jp/intdash-edge/linux/${DISTRIBUTION} \
    $(lsb_release -cs) \
    stable" \
    | sudo tee /etc/apt/sources.list.d/intdash-edge.list
$ sudo apt-get update
$ sudo apt-get install intdash-edge
```

2.2 Files to be installed

When installed the Agent with the default settings, the following files and folders are placed.

```
+ /etc
  + opt
    + intdash                          # (1)
      - manager.conf                   # (2)
      - logger.conf.canopen            # (3)
      - logger.conf.nmea               # (4)
      - logger.conf.gstreamer_h264     # (5)
      - logger.conf.mjpeg              # (6)
```

(continues on next page)

(continued from previous page)

```
      - logger.conf.socketcan           # (7)
      - logger.conf.v4lh264             # (8)

+ /opt
  + vm2m
    + bin                               # (9)
    + etc                               # (10)
    + lib                               # (11)
    + sbin                              # (12)
      - intdash-edge-client             # (13)
      - intdash-edge-logger             # (14)
      - intdash-edge-manager            # (15)
    + share
      + licenses                        # (16)
```

Number	File or directory description
(1)	Directory for storing configuration files
(2)	Sample configuration file for the agent
(3)	Sample configuration file for canopen type device connector
(4)	Sample configuration file for nmea type device connector
(5)	Sample configuration file for gstreamer_h264 type device connector
(6)	Sample configuration file for mjpeg type device connector
(7)	Sample configuration file for socketcan type device connector
(8)	Sample configuration file for v4lh264 type device connector
(9)	Directory for tools
(10)	Directory for storing static configuration files
(11)	Directory for libraries used by Agent (or intdash-edge-logger)
(12)	Directory for the Agent executable and the aptpod device connectors.
(13)	Executable file of a client that sends data over the network.
(14)	Device connector made by aptpod
(15)	Manager executable, the core of the Agent
(16)	Directory where information about open source libraries used by the Agent is stored

2.3 Start and stop the Agent

2.3.1 Start the Agent

To start the Agent, execute the following command.

```
$ sudo \
   LD_LIBRARY_PATH=/opt/vm2m/lib \
   /opt/vm2m/sbin/intdash-edge-manager -C <full-path-to-the-configuration-file>
```

For `full-path-to-the-configuration-file`, you can use the pre-installed configuration file `/etc/opt/intdash/manager.conf`. The pre-installed configuration file uses the environment variables.

Variable name	Description
INTDASH_EDGE_UUID	The UUID to identify this edge. Use the one issued by the intdash server. (Example: f90f2b42-66a5-4a57-8e99-468c36ebb6f2)
INTDASH_EDGE_SECRET	The token for authentication. Use the one issued for this edge by the intdash server. (Example: sEh9ZHPoKX8QYU-v0Noe0ZPzxGBF.....iBn5fn_eFM) See Edge information setting (p. 11) for more information.
INTDASH_EDGE_SERVER	Enter the FQDN of the intdash server. (Example: dummy.intdash.jp) Please note that depending on your environment, the server name used by the edge and the server name used by the web applications may be different. In the case of intdash environments operated by aptpod, the server name for edges is usually <xxxxx>.intdash.jp and the server name for web applications is <xxxxx>.vm2m.jp (the <xxxxx> part is the same).
INTDASH_EDGE_APPDIR	Application data storage location (example: /var/lib)
INTDASH_EDGE_RUNDIR	Temporary file location (example: /var/run)
INTDASH_EDGE_BINDIR	Script file location (example: /opt/vm2m/bin)
INTDASH_EDGE_SBINDIR	Executable file location (example: /opt/vm2m/sbin)
INTDASH_EDGE_LIBDIR	Library location (example: /opt/vm2m/lib)
INTDASH_EDGE_CONFDIR	Configuration file location (example: /etc/opt/intdash)

```
$ sudo \
   LD_LIBRARY_PATH=/opt/vm2m/lib \
   INTDASH_EDGE_UUID=f90f2b42-66a5-4a57-8e99-468c36ebb6f2 \
   INTDASH_EDGE_SECRET=sEh9ZHPoKX8QYU-v0Noe0ZPzxGBF.....iBn5fn_eFM \
   INTDASH_EDGE_SERVER=dummy.intdash.jp \
   INTDASH_EDGE_APPDIR=/var/lib \
   INTDASH_EDGE_RUNDIR=/var/run \
   INTDASH_EDGE_BINDIR=/opt/vm2m/bin \
   INTDASH_EDGE_SBINDIR=/opt/vm2m/sbin \
   INTDASH_EDGE_LIBDIR=/opt/vm2m/lib \
   INTDASH_EDGE_CONFDIR=/etc/opt/intdash \
   /opt/vm2m/sbin/intdash-edge-manager -C /etc/opt/intdash/manager.conf
```

Note: If you set the log level to the environment variable INTDASH_LOG, logs of that level or higher will be output. Available log levels are as follows. If not specified, info or higher logs will be output.

- debug (debug, info, warn, and error logs are output)
- info (info, warn, and error logs are output)
- warn (warn and error logs are output)
- error (error logs are output)
- quiet (no log is output)

In the pre-installed configuration file /etc/opt/intdash/manager.conf, Device Connectors are not configured. Therefore, only the Status plugin will work. The Status plugin gets the system information, network information, and Agent status as shown below and sends them to the intdash server.

Data type	Data ID	Channel	Content
String	s00	255	System information
String	s20	255	Network information
String	s50	255	Agent status

2.3.2 Stop the Agent

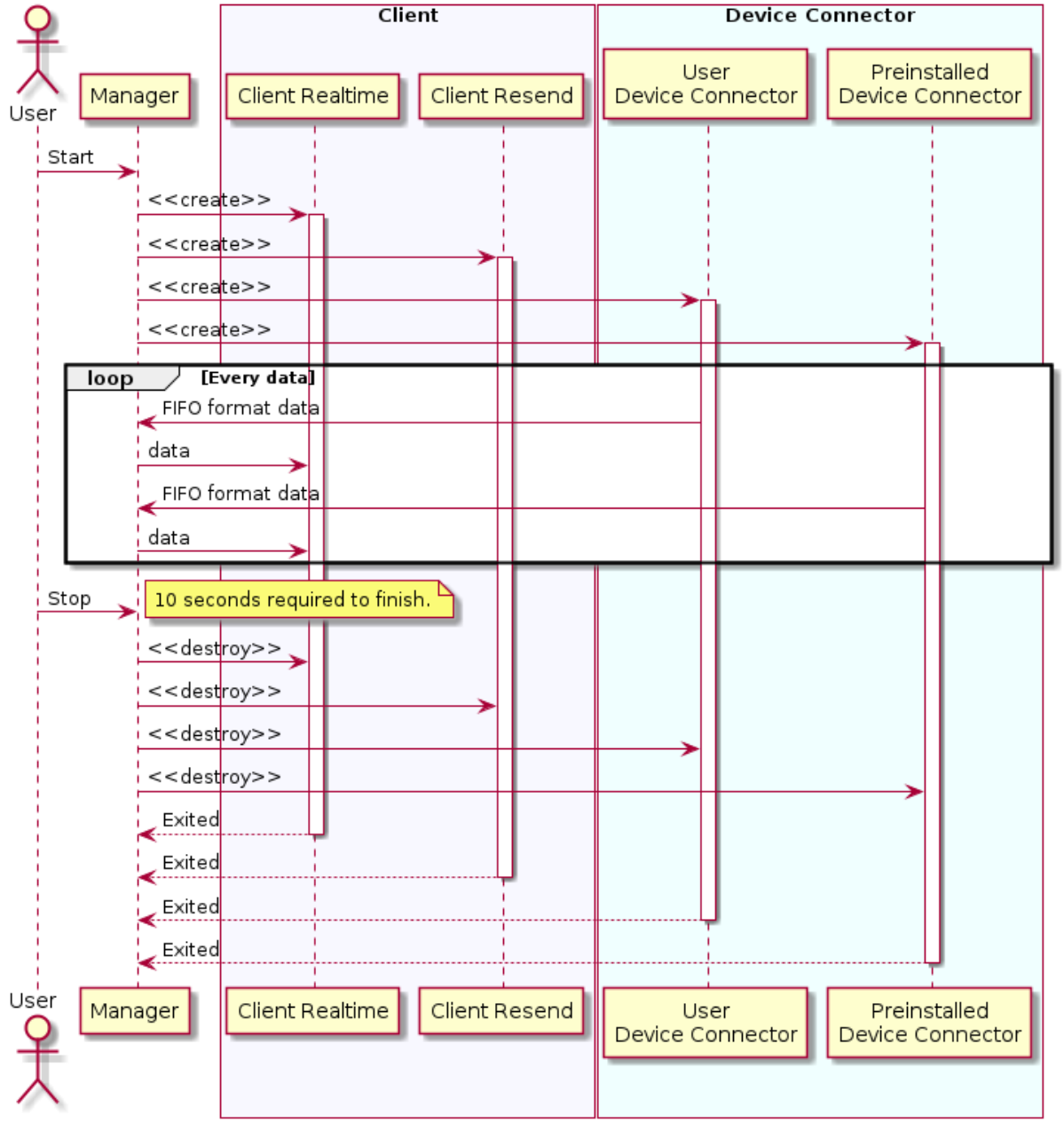
To stop the Agent, do one of the following:

- Send SIGINT
- Execute the following command:

```
$ LD_LIBRARY_PATH=/opt/vm2m/lib /opt/vm2m/sbin/intdash-edge-manager -k
```

> **Attention:** It may take about 10 seconds to stop the Agent.

2.3.3 Reference: Sequence from start to stop of the Agent



03 Change settings

The Agent sends the data received from the Device Connector to the intdash server. During this process, the data can be filtered or resent depending on the status of the network.

The basic configuration is done in the manager configuration file (e.g. `manager.conf`).

To use Device Connectors, one configuration file is required for each Device Connector. The configuration file for the Device Connector to be used must be specified in the manager's configuration file.

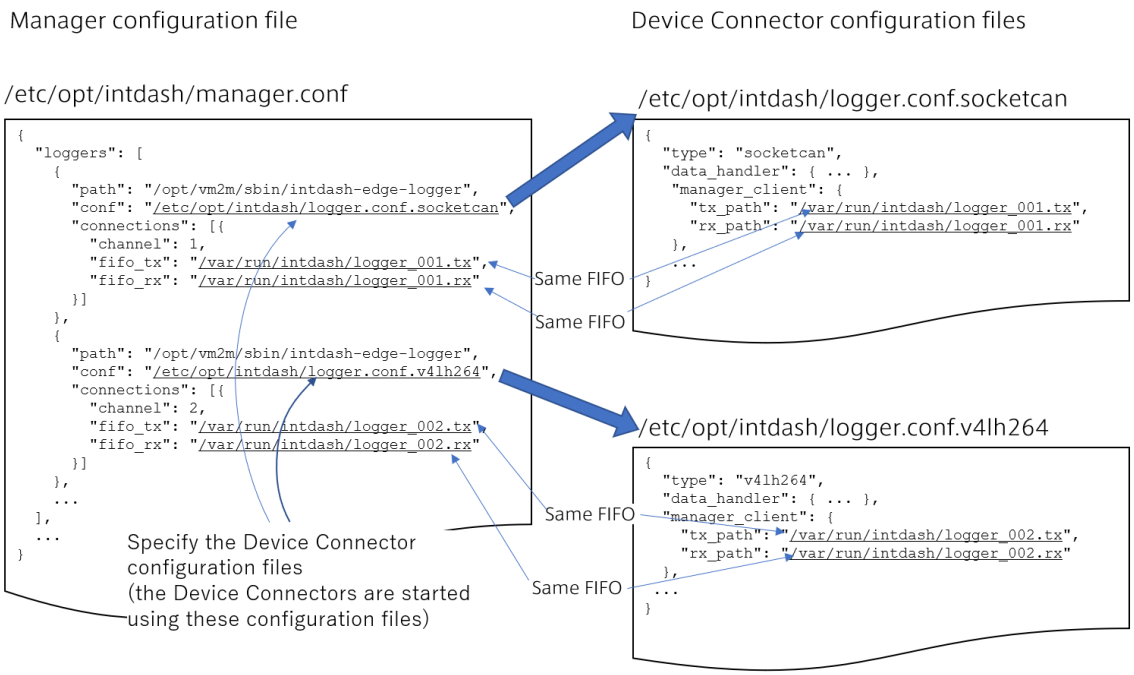


Fig. 1 Example of Manager configuration file and Device Connector configuration files

See All settings for Agent (p. 52) for a list of configurable items in the configuration file. Here are some typical configuration examples.

Note: The "logger" in the configuration file refers to the Device Connector.

3.1 Edge information setting

Set the edge account that the Agent uses to connect to intdash. Each client that communicates with the intdash server must be configured individually. As shown in the example below, specify the edge account UUID and the token in the settings of each client.

> **Note:** Basically, use the same edge account for all clients of one Agent. In the example below, the Realtime and Resend clients use the same edge account.

Edge information setting example

```
{
  "clients": [
    {
      "my_id": "f90f2b42-66a5-4a57-8e99-468c36ebb6f2",                # (1)
      "my_secret": "sEh9ZHPoKX8QYU-v0Noe0ZPzxGBF.....................iBn5fn_eFM", # (2)
      "type": "realtime",
      ...
    },
    {
      "my_id": "f90f2b42-66a5-4a57-8e99-468c36ebb6f2",                # (1)
      "my_secret": "sEh9ZHPoKX8QYU-v0Noe0ZPzxGBF.....................iBn5fn_eFM", # (2)
      "type": "resend",
      ...
    }
  ],
  ...
}
```

Number	Field	Description
(1)	`my_id`	The edge UUID to assign to this client.
(2)	`my_secret`	Client secret for authentication.

> **Attention:** Note when updating from a previous version of Edge Agent.
>
> Edge Agent version 1.19.0 and later uses an edge UUID and a client secret as authentication information. To use a client secret, you need to set `my_secret` and `auth_path` in `clients` of `manager.conf`.
>
> If you want to continue to use the configuration file `manager.conf` from Edge Agent 1.19.0 or earlier, refer to the configuration template `/opt/vm2m/etc/manager.conf` and add `my_secret` and `auth_path`. The value of `auth_path` should be copied from the configuration template `/opt/vm2m/etc/manager.conf`.
>
> Note that authentication with edge tokens can still be used. If you want to use your edge token, set the edge token as `my_token` instead of `my_secret`. (The configuration file `manager.conf` from Edge Agent 1.19.0 and earlier can still be used.)

3.2 Settings for using the pre-installed Device Connector

Settings are required to use the pre-installed Device Connector `intdash-edge-logger`.

To configure the Device Connector, configure both the manager's configuration file (manager.conf) and the Device Connector's configuration file.

By preparing a configuration file for each Device Connector and setting multiple Device Connectors in the manager's configuration file, you can launch `intdash-edge-logger` in multiple processes to collect data from multiple devices.

Note: Acquiring video data from a camera
There are three ways to acquire video data using the pre-installed Device Connectors.

- Acquire video from a camera that can output Motion JPEG and use an mjpeg-type Device Connector.
- Acquire video from a camera that can output H.264 and use a v4lh264-type Device Connector.
- Acquire H.264 output from GStreamer, using a Device Connector of type gstreamer_h264 (e.g., when converting RAW data from a camera to H.264 using GStreamer and then acquiring the converted data).

Choose an appropriate method according to the data format of your camera output.
The data format can be checked by installing the `v4l2-ctl` command.
The following command lists the formats that the camera device `/dev/video0` can output.

```
$ v4l2-ctl -d /dev/video0 --list-formats
```

You can also check the resolution and frame rate that the camera can output with the following command.

```
$ v4l2-ctl -d /dev/video0 --list-formats-ext
```

3.2.1 Manager configuration file example (manager.conf)

In the manager's configuration file, fill in the Device Connector settings in `loggers`.

```
{
  "loggers": [
    {                                                  # (1)
      "path": "/opt/vm2m/sbin/intdash-edge-logger",    # (2)
      "conf": "/etc/opt/intdash/logger.conf.mjpeg",    # (3)
      "connections": [{
        "channel": 1,                                  # (4)
        "fifo_tx": "/var/run/intdash/logger_001.tx",   # (5)
        "fifo_rx": "/var/run/intdash/logger_001.rx"    # (6)
      }]
    },
    {                                                  # (7)
      "path": "/opt/vm2m/sbin/intdash-edge-logger",
      "conf": "/etc/opt/intdash/logger.conf.nmea",
      "connections": [{
        "channel": 2,
        "fifo_tx": "/var/run/intdash/logger_002.tx",
        "fifo_rx": "/var/run/intdash/logger_002.rx"
      }]
    },
    ...
  ],
  ...
}
```

Number	Field	Description
(1)	-	The connection with one Device Connector is represented by one JSON object.
(2)	`path`	Full path of the pre-installed Device Connector.
(3)	`conf`	The configuration file for the Device Connector. Here, the setting `logger.conf.mjpeg` for Motion JPEG is specified as an example.
(4)	`channel`	The channel (0-255) to be used for this Device Connector. A channel number is assigned to the data obtained from the Device Connector. The channel number should not be duplicated with other Device Connectors.
(5)	`fifo_tx`	The FIFO path that this Device Connector uses to send data. The path should not overlap with other Device Connectors.
(6)	`fifo_rx`	The FIFO path that this Device Connector uses to receive data. The path should not overlap with other Device Connectors.
(7)	-	If you want to use more than one Device Connector, configure the second Device Connector from here. Make the settings in the same way as above.

3.2.2 Device connector configuration file example

Setting example of mjpeg type Device Connector

The mjpeg type Device Connector acquires Motion JPEG data from a UVC (USB Video Class) camera that supports Video4Linux. Therefore, the camera needs to be able to output Motion JPEG.

Configuration file `/etc/opt/intdash/logger.conf.mjpeg`

```
{
  "type": "mjpeg",                              # (1)
  "data_handler": {
    "path": "/dev/video0",                      # (2)
    "baudrate": 15,                             # (3)
    "camera_width": 320,                        # (4)
    "camera_height": 240,                       # (5)
    "camera_hwencodedelay_msec": 100            # (6)
  },
  "manager_client": {
    "tx_path": "/var/run/intdash/logger_XXX.tx",   # (7)
    "rx_path": "/var/run/intdash/logger_XXX.rx"    # (8)
  },
  "basetime": "/var/run/intdash/basetime",      # (9)
  "status": "/var/run/intdash/logger_XXX.stat"  # (10)
}
```

Number	Field	Description
(1)	`type`	The type of the Device Connector. For mjpeg type, use "mjpeg".
(2)	`path`	Device path
(3)	`baudrate`	Frame rate (1, 5, 10, 15, or 30) [fps]
(4)	`camera_width`	Frame width (320 or 640)
(5)	`camera_height`	Frame height (240 or 480)
(6)	`camera_hwencodedelay_msec`	Timestamp offset (camera processing time) [msec]. For example, if you set `100`, the timestamp of 100 milliseconds ago will be used assuming that the processing in the camera took 100 milliseconds.
(7)	`tx_path`	The FIFO path used by the Device Connector to send data. Set the same path as the fifo_tx for this Device Connector in manager.conf.
(8)	`rx_path`	The FIFO path used by the Device Connector to receive data. Set the same path as the fifo_rx for this Device Connector in manager.conf.
(9)	`basetime`	The path to the file that the Device Connector uses for time management. Set the same value as manager.basetime in manager.conf.
(10)	`status`	The path to the file that the Device Connector writes the status to.

Setting example of v4lh264 type Device Connector

The v4lh264 type Device Connector acquires H.264 data from a UVC (USB Video Class) camera that supports Video4Linux. Therefore, the camera needs to be able to output H.264.

Configuration file `/etc/opt/intdash/logger.conf.v4lh264`

```
{
 "type": "v4lh264",                          # (1)
 "data_handler": {
   "path": "/dev/video0",                    # (2)
   "baudrate": 15,                           # (3)
   "camera_width": 1920,                     # (4)
   "camera_height": 1080,                    # (5)
   "camera_hwencodedelay_msec": 100          # (6)
 },
 "manager_client": {
   "tx_path": "/var/run/intdash/logger_XXX.tx",   # (7)
   "rx_path": "/var/run/intdash/logger_XXX.rx"    # (8)
 },
 "basetime": "/var/run/intdash/basetime",    # (9)
 "status": "/var/run/intdash/logger_XXX.stat"    # (10)
}
```

Number	Field	Description
(1)	`type`	The type of Device Connector. For v4lh264 type, set to "v4lh264".
(2)	`path`	Device path
(3)	`baudrate`	Frame rate (5, 10, 15, 30) [fps]
(4)	`camera_width`	Frame width (1080, 1920)
(5)	`camera_height`	Frame height (720, 1020)
(6)	`camera_hwencodedelay_msec`	Timestamp offset (camera processing time) [msec]. For example, if you set `100`, the timestamp of 100 milliseconds ago will be used assuming that the processing in the camera took 100 milliseconds.
(7)	`tx_path`	The FIFO path used by the Device Connector to send data. Set the same path as the fifo_tx for this Device Connector in manager.conf.
(8)	`rx_path`	The FIFO path used by the Device Connector to receive data. Set the same path as the fifo_rx for this Device Connector in manager.conf.
(9)	`basetime`	The path to the file that the Device Connector uses for time management. Set the same value as manager.basetime in manager.conf.
(10)	`status`	The path to the file that the pre-installed Device Connector writes the status to.

Example setting of gstreamer_h264 type Device Connector

The gstreamer_h264 type Device Connector acquires H.264 video from GStreamer. Therefore, GStreamer needs to output H.264.

Configuration file `/etc/opt/intdash/logger.conf.gstreamer_h264`

```
{
 "type": "gstreamer_h264",                      # (1)
 "data_handler": {
   "path": "/dev/video0",                       # (2)
   "baudrate": 15,                              # (3)
   "camera_width": 1920,                        # (4)
   "camera_height": 1080,                       # (5)
   "camera_keyperiod": 150,                     # (6)
   "camera_hwencodedelay_msec": 100,            # (7)
   "command": "gst-launch-1.0 -q v4l2src device=$_PATH ! image/jpeg,width=$_WIDTH,height=$_HEIGHT,framerate=$_
↪FPS/1 ! queue ! vaapijpegdec ! queue ! vaapipostproc ! queue ! vaapih264enc rate-control=1 bitrate=3072 max-
↪bframes=0 keyframe-period=$_KEYPERIOD ! fdsink fd=1" # (8)
 },
 "manager_client": {
   "tx_path": "/var/run/intdash/logger_XXX.tx",   # (9)
   "rx_path": "/var/run/intdash/logger_XXX.rx"    # (10)
 },
 "basetime": "/var/run/intdash/basetime",         # (11)
 "status": "/var/run/intdash/logger_XXX.stat"     # (12)
}
```

Number	Field	Description
(1)	`type`	The type of Device Connector. For gstreamer_h264 type, set to "gstreamer_h264".
(2)	`path`	Device path
(3)	`baudrate`	Frame rate (1, 5, 10, 15, 30) [fps]
(4)	`camera_width`	Frame width (1080, 1920)
(5)	`camera_height`	Frame height (720, 1020)
(6)	`camera_keyperiod`	Keyframe interval (set frame rate x 10)
(7)	`camera_hwencodedelay_msec`	Timestamp offset (camera processing time) [msec]. For example, if you set `100`, the timestamp of 100 milliseconds ago will be used assuming that the processing in the camera took 100 milliseconds.
(8)	`command`	GStreamer command. Set the pipeline suitable for the camera and the runtime system so that H.264 data is output from the standard output of the command.
(9)	`tx_path`	The FIFO path used by the Device Connector to send data. Set the same path as the fifo_tx for this Device Connector in manager.conf.
(10)	`rx_path`	The FIFO path used by the Device Connector to receive data. Set the same path as the fifo_rx for this Device Connector in manager.conf.
(11)	`basetime`	The path to the file that the Device Connector uses for time management. Set the same value as manager.basetime in manager.conf.
(12)	`status`	The path to the file that the pre-installed Device Connector writes the status to.

In the `command`, other setting values can be referred to by using the following variables.

$_PATH
: The value set for `path`

$_FPS
: The value set to `baudrate`

$_WIDTH
: The value set to `camera_width`

$_HEIGHT
: The value set to `camera_height`

$_KEYPERIOD
: The value set to `camera_keyperiod`

Setting example of nmea type Device Connector

nmea type Device Connector acquires NMEA data from GPS device

Configuration file `/etc/opt/intdash/logger.conf.nmea`

```
{
 "type": "nmea",                                # (1)
 "data_handler": {
   "path": "/dev/ttyXX",                        # (2)
   "baudrate": 57600                            # (3)
 },
 "manager_client": {
   "tx_path": "/var/run/intdash/logger_XXX.tx",  # (4)
   "rx_path": "/var/run/intdash/logger_XXX.rx"   # (5)
 },
 "basetime": "/var/run/intdash/basetime",       # (6)
 "status": "/var/run/intdash/logger_XXX.stat"   # (7)
}
```

Number	Field	Description
(1)	`type`	The type of Device Connector. For nmea type, set to "nmea".
(2)	`path`	Device path
(3)	`baudrate`	Communication baud rate with GPS module [bps]
(4)	`tx_path`	The FIFO path used by the Device Connector to send data. Set the same path as the fifo_tx for this Device Connector in manager.conf.
(5)	`rx_path`	The FIFO path used by the Device Connector to receive data. Set the same path as the fifo_rx for this Device Connector in manager.conf.
(6)	`basetime`	The path to the file that the Device Connector uses for time management. Set the same value as manager.basetime in manager.conf.
(7)	`status`	The path to the file that the pre-installed Device Connector writes the status to.

Setting example of socketcan type Device Connector

A socketcan type Device Connector gets CAN data from the open source SocketCAN driver.

Configuration file `/etc/opt/intdash/logger.conf.socketcan`

```
{
  "type": "socketcan",                          # (1)
  "data_handler": {
    "path": "can0",                             # (2)
    "baudrate": 500,                            # (3)
    "listenonly": 0                             # (4)
  },
  "manager_client": {
    "tx_path": "/var/run/intdash/logger_XXX.tx",   # (5)
    "rx_path": "/var/run/intdash/logger_XXX.rx"    # (6)
  },
  "basetime": "/var/run/intdash/basetime",      # (7)
  "status": "/var/run/intdash/logger_XXX.stat"  # (8)
}
```

Number	Field	Description
(1)	`type`	The type of Device Connector. For socketcan type, set to "socketcan".
(2)	`path`	Interface name
(3)	`baudrate`	CAN bus baud rate (125, 250, 500, 1000) [Kbps]
(4)	`listenonly`	(int) 0: returns ACK, non-zero: does not return ACK
(5)	`tx_path`	The FIFO path used by the Device Connector to send data. Set the same path as the fifo_tx for this Device Connector in manager.conf.
(6)	`rx_path`	The FIFO path used by the Device Connector to receive data. Set the same path as the fifo_rx for this Device Connector in manager.conf.
(7)	`basetime`	The path to the file that the Device Connector uses for time management. Set the same value as manager.basetime in manager.conf.
(8)	`status`	The path to the file that the pre-installed Device Connector writes the status to.

Setting example of canopen type Device Connector

A canopen type Device Connector acquires CANOpen data from the open source SocketCAN driver.

Configuration file `/etc/opt/intdash/logger.conf.canopen`

```
{
  "type": "canopen",                            # (1)
  "data_handler": {
    "path": "can0",                             # (2)
    "baudrate": 500,                            # (3)
    "listenonly": 0                             # (4)
  },
  "manager_client": {
    "tx_path": "/var/run/intdash/logger_XXX.tx",   # (5)
    "rx_path": "/var/run/intdash/logger_XXX.rx"    # (6)
```

(continues on next page)

(continued from previous page)

```
  },
  "basetime": "/var/run/intdash/basetime",         # (7)
  "status": "/var/run/intdash/logger_XXX.stat"     # (8)
}
```

Number	Field	Description
(1)	`type`	The type of Device Connector. For canopen type, set to "canopen".
(2)	`path`	Interface name
(3)	`baudrate`	CAN bus baud rate (125, 250, 500, 1000) [Kbps]
(4)	`listenonly`	(int) 0: returns ACK, non-zero: does not return ACK
(5)	`tx_path`	The FIFO path used by the Device Connector to send data. Set the same path as the fifo_tx for this Device Connector in manager.conf.
(6)	`rx_path`	The FIFO path used by the Device Connector to receive data. Set the same path as the fifo_rx for this Device Connector in manager.conf.
(7)	`basetime`	The path to the file that the Device Connector uses for time management. Set the same value as manager.basetime in manager.conf.
(8)	`status`	The path to the file that the pre-installed Device Connector writes the status to.

3.3 Settings related to sending and receiving of data

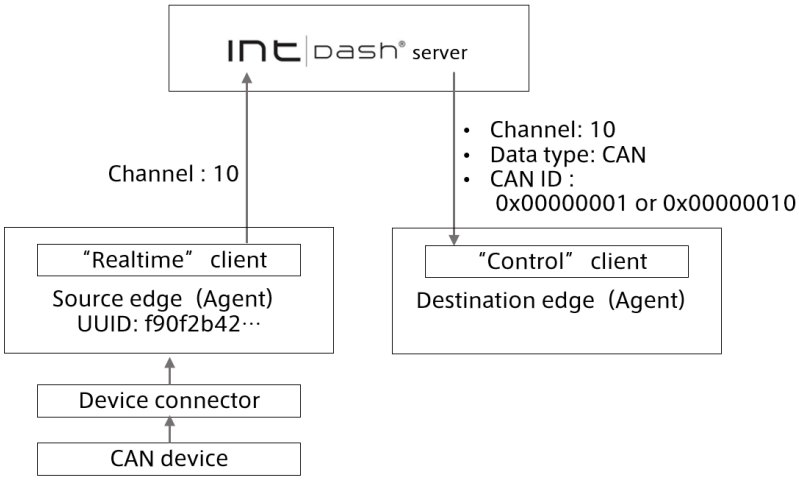
The Agent can receive data from the intdash server and send it to the Device Connector.

The following is an example of settings for sending and receiving data between two agents.

3.3.1 Setting example for sending and receiving CAN data (destination is not specified)

The following example is a configuration for an Agent to send CAN data to another Agent via the intdash server.

In this configuration example, the receiving Agent receives the CAN data sent by the sending Agent on channel 10.



Setting example of CAN data sender edge (destination is not specified)

```
{
  "clients": [{
    ...
    "my_id": "f90f2b42-66a5-4a57-8e99-468c36ebb6f2",                # (1)
    "my_secret": "sEh9ZHPoKX8QYU-v0Noe0ZPzxGBF.....................iBn5fn_eFM", # (2)
    "type": "realtime",                                             # (3)
    ...
  }],
  "loggers": [{                                                     # (4)
    "connections": [{
        "channel": 10,                                              # (5)
        ...
    }],
    ...
  }],
  ...
}
```

Number	Field	Description
(1)	`my_id`	UUID of the sending edge (this edge)
(2)	`my_secret`	Client secret for the sending edge (this edge). Reference: Edge information setting (p. 11).
(3)	`type`	The Realtime client `realtime` is specified for real-time transmission.
(4)	`loggers`	Set the Device Connector that acquires CAN data.
(5)	`channel`	The channel to send data.

Setting example of CAN data receiving edge

```
{
  "clients": [{
    ...
    "my_id": "c35618bf-aa2c-4abc-8a4e-5b157b90c9ef",                # (1)
    "my_secret": "hsNxJhvDNHR2QcXbXl.......................Z0RWKvfPs_neAkjTNSO5", # (2)
    "down_dst_id": "00000000-0000-0000-0000-000000000000",          # (3)
    "ctlr_id": "f90f2b42-66a5-4a57-8e99-468c36ebb6f2",              # (4)
    "ctlr_flts":[                                                   # (5)
      {
        "channel": 10,                                              # (6)
        "dtype": 1,                                                 # (7)
        "ids": [1, 16]                                              # (8)
      }
    ],
    "type": "control",                                              # (9)
    ...
  }],
  "loggers": [{
    "connections": [{
        "channel": 10,                                              # (10)
        ...
    }],
    ...
  }],
  ...
}
```

Number	Field	Description
(1)	`my_id`	UUID of the receiving edge (this edge)
(2)	`my_secret`	Client secret for the receiving edge (this edge). Reference: Edge information setting (p. 11).
(3)	`down_dst_id`	Data addressed to the UUID specified here is received. If `0000...` is specified, the data is received no matter what the destination of the data is.
(4)	`ctlr_id`	UUID of the source edge of the data to be received (only the data sent from the specified edge is received)
(5)	`ctlr_flts`	A filter that specifies the data to receive. Specify by the combination of channel, iSCP data type, and ID. Multiple filters can be set (allow list). (You can filter the data received from the intdash server. For more information on filtering data to be received, refer to the iSCP 1.0 documentation.)
(6)	`channel`	Channel of data to receive (In this example, only the data of channel 10 is received.)
(7)	`dtype`	The type of data to receive. Specify the iSCP data type code in decimal notation. (In this case, 1 represents CAN. Only CAN data is received.)
(8)	`ids`	CAN ID of the data to be received (This example shows that only the data whose CAN ID is `0x00000001` or `0x00000010` is received. In the case of CAN data, if you set the value to an empty array `[]`, data of any ID will be received.)
(9)	`type`	Use the Control client (`control`) to receive data.
(10)	`channel`	The channel on which the Device Connector receives data. In this example, this edge uses channel 10.

Important: In the `dtype` field, specify the iSCP data type code in decimal notation. The iSCP data type codes are as follows.

Data type code (decimal)	Data type
0x01 (1)	CAN
0x02 (2)	NMEA
0x03 (3)	General Sensor
0x04 (4)	Controlpad
0x05 (5)	MAVLink 2 Packet (Communication protocol for Micro Air Vehicles/drones)
0x09 (9)	JPEG
0x0A (10)	String
0x0B (11)	Float (Double precision floating point number)
0x0C (12)	Int (64bit signed integer)
0x0D (13)	H.264
0x0E (14)	Bytes (Byte sequence)
0x0F (15)	PCM (WAVE)
0x10 (16)	AAC (ADTS)
0x7F (127)	Generic (Generic binary data)

Note that the above data type codes for iSCP are different from the data type codes for FIFO data format used between Agent and Device Connector (p. 46).

3.4 Settings for sending timing (filtering on the sender's side)

By setting a filter on the sending agent, you can distribute the data to the Realtime client or Bulk client that sends the data to the intdash server, and adjust the data transmission timing. For more information on filters, see Filter at the sender's side (p. 36).

3.4.1 A setting example in which low frequency data is sent in real time and the rest of the data is sent later.

If the network bandwidth is narrow, sampling can be used to thin out the data so that only some data can be sent by the Realtime client and the rest of the data can be sent by the Resend client when the bandwidth is restored.

A setting example where channel 1 is sampled at 1-second intervals, the sampled data is sent by the Realtime client, and the rest of the data is sent by the Resend client

```
{
  "manager": {
    "filters": [
```

(continues on next page)

(continued from previous page)

```
    {
      "name": "sampling",                                    # (1)
      "channel": "1",                                        # (2)
      "target": "realtime",                                  # (3)
      "setting": [
        {
          "key": "rate",                                     # (4)
          "value": "1000"                                    # (5)
        }
      ]
    }
  ],
  ...
},
"clients": [
  {
    "my_id": "f90f2b42-66a5-4a57-8e99-468c36ebb6f2",         # (6)
    "my_secret": "sEh9ZHPoKX8QYU-v0Noe0ZPzxGBF.....................iBn5fn_eFM", # (7)
    "type": "realtime",                                      # (8)
    "protocol": "mod_websocket.v2",                          # (9)
    ...
  },
  {
    "my_id": "f90f2b42-66a5-4a57-8e99-468c36ebb6f2",         # (6)
    "my_secret": "sEh9ZHPoKX8QYU-v0Noe0ZPzxGBF.....................iBn5fn_eFM", # (7)
    "type": "bulk",                                          # (10)
    "protocol": "mod_websocket.v2",                          # (9)
    "store_cycle": 0,                                        # (11)
    ...
  },
  {
    "my_id": "f90f2b42-66a5-4a57-8e99-468c36ebb6f2",         # (6)
    "my_secret": "sEh9ZHPoKX8QYU-v0Noe0ZPzxGBF.....................iBn5fn_eFM", # (7)
    "type": "resend",                                        # (12)
    "protocol": "mod_http",                                  # (13)
    ...
  }
],
"loggers": [{
  "connections": [{
      "channel": 1,                                          # (14)
      ...
  }],
  ...
}],
...
}
```

Number	Field	Description
(1)	`name`	The type of filter, `sampling`.
(2)	`channel`	The channel to be filtered. In this example, channel 1 is set.
(3)	`target`	How to send the filtered content. Set to `realtime`.
(4)	`key`	Advanced filter settings. Use `rate` to set the sampling period.
(5)	`value`	Advanced filter settings. In this example, the sampling period is set to 1000 milliseconds (1 second). Therefore, this edge sends one piece of data per second.
(6)	`my_id`	The UUID of the sending edge (this edge).
(7)	`my_secret`	Client secret for the sending edge (this edge). Reference: Edge information setting (p. 11).
(8)	`type`	Sending client type, `realtime`.
(9)	`protocol`	Communication protocol of the sending client. Specify `mod_websocket.v2`, as the Realtime and Bulk clients use the Realtime API.
(10)	`type`	Sending client type, `bulk`.
(11)	`store_cycle`	Sending interval for the Bulk client. Set this to 0 to delegate data transmission to the Resend client.
(12)	`type`	Sending client type, `resend`.
(13)	`protocol`	Communication protocol of the sending client. In this example, the Resend client uses the REST API, so specify `mod_http`.
(14)	`channel`	The channel to be used for the acquired data. In this example, this Device Connector (logger) is channel 1.

3.4.2 Setting example to send some data in real time and send other data later

If network bandwidth is tight, you can configure the Realtime client to send only small or infrequent data and the rest of the data to be sent by the Resend client when the bandwidth is restored.

Setting example for sending data other than channel 1 with the Realtime client and delegate the channel 1 data to the Resend client

```
{
  "manager": {
    "filters": [
      {
        "name": "channel",                                          # (1)
        "channel": "1",                                             # (2)
        "target": "realtime",                                       # (3)
        "setting": []                                               # (4)
      }
    ],
    ...
  },
  "clients": [
    {
      "my_id": "f90f2b42-66a5-4a57-8e99-468c36ebb6f2",              # (5)
      "my_secret": "sEh9ZHPoKX8QYU-v0Noe0ZPzxGBF.....................iBn5fn_eFM", # (6)
      "type": "realtime",                                           # (7)
```

(continues on next page)

(continued from previous page)

```
      "protocol": "mod_websocket.v2",                                  # (8)
      ...
    },
    {
      "my_id": "f90f2b42-66a5-4a57-8e99-468c36ebb6f2",                 # (5)
      "my_secret": "sEh9ZHPoKX8QYU-v0Noe0ZPzxGBF.......................iBn5fn_eFM", # (6)
      "type": "bulk",                                                  # (9)
      "protocol": "mod_websocket.v2",                                  # (8)
      "store_cycle": 0,                                                # (10)
      ...
    },
    {
      "my_id": "f90f2b42-66a5-4a57-8e99-468c36ebb6f2",                 # (5)
      "my_secret": "sEh9ZHPoKX8QYU-v0Noe0ZPzxGBF.......................iBn5fn_eFM", # (6)
      "type": "resend",                                                # (11)
      "protocol": "mod_http",                                          # (12)
      ...
    }
  ],
  "loggers": [
    {
      "connections": [{
          "channel": 1,                                                  # (13)
          ...
      }],
      ...
    },
    {
      "connections": [{
          "channel": 2,                                                  # (14)
          ...
      }],
      ...
    }
  ],
  ...
}
```

Number	Field	Description
(1)	`name`	The type of filter, `channel`.
(2)	`channel`	The channel to be filtered. In this example, it is channel 1.
(3)	`target`	How to send the filtered content. Set to `realtime`.
(4)	`setting`	Advanced filter settings. Leave the settings empty.
(5)	`my_id`	The UUID of the sending edge (this edge).
(6)	`my_secret`	Client secret for the sending edge (this edge). Reference: Edge information setting (p. 11).
(7)	`type`	Sending client type, `realtime`.
(8)	`protocol`	Communication protocol of the sending client. Specify `mod_websocket.v2`, as the Realtime and Bulk clients use the Realtime API.
(9)	`type`	Sending client type, `bulk`.
(10)	`store_cycle`	Sending interval of the Bulk client. Set this to 0 to delegate data transmission to the Resend client.
(11)	`type`	Sending client type, `resend`.
(12)	`protocol`	Communication protocol of the sending client. In this example, the Resend client uses the REST API, so specify `mod_http`.
(13)	`channel`	Channel to be assigned to the acquired data. In this example, the first Device Connector (logger) is channel 1.
(14)	`channel`	Channel to be assigned to the acquired data. In this example, the second Device Connector (logger) is channel 2.

3.4.3 Setting example to save all data as RAW data without sending

If there is no network connection or the bandwidth is too narrow to send data, you can give up sending data to the intdash server and dump all the data to your local storage. The data dumped to the local storage will need to be manually uploaded to the intdash server later.

Setting example not to send data to the server

```
{
  "manager": {
    ...
  },
  "clients": [],                                          # (1)
  "loggers": [
    {
      "connections": [{
          "channel": 1,                                   # (2)
          ...
      }],
      ...
    },
    {
      "connections": [{
          "channel": 2,                                   # (3)
          ...
      }],
```

(continues on next page)

(continued from previous page)

```
    ...
  }
 ],
 ...
}
```

Number	Field	Description
(1)	`clients`	Client settings. Leave the sending client settings empty.
(2)	`channel`	Channel to be assigned to the acquired data. In this example, the first Device Connector (logger) is channel 1.
(3)	`channel`	Channel to be assigned to the acquired data. In this example, the second Device Connector (logger) is channel 2.

3.4.4 Setting example to store all data for Resend client

If the network is unstable and the bandwidth fluctuates greatly, you can give up the real-time transmission, store all the data, and send with the Resend client in one batch when the bandwidth is restored. This retransmission process by the Resend client is automatic and you do not need to manually upload the data to the intdash server.

Setting example to delegate the data of all channels to the Resend client without sending in real time

```
{
  "manager": {
    "filters": [
      {
        "name": "channel",                                          # (1)
        "channel": "-1",                                            # (2)
        "target": "realtime",                                       # (3)
        "setting": []                                               # (4)
      }
    ],
    ...
  },
  "clients": [
    {
      "my_id": "f90f2b42-66a5-4a57-8e99-468c36ebb6f2",              # (5)
      "my_secret": "sEh9ZHPoKX8QYU-v0Noe0ZPzxGBF.......................iBn5fn_eFM", # (6)
      "type": "realtime",                                           # (7)
      "protocol": "mod_websocket.v2",                               # (8)
      ...
    },
    {
      "my_id": "f90f2b42-66a5-4a57-8e99-468c36ebb6f2",              # (5)
      "my_secret": "sEh9ZHPoKX8QYU-v0Noe0ZPzxGBF.......................iBn5fn_eFM", # (6)
      "type": "bulk",                                               # (9)
      "protocol": "mod_websocket.v2",                               # (8)
      "store_cycle": 0,                                             # (10)
      ...
    },
    {
```

(continues on next page)

(continued from previous page)

```
      "my_id": "f90f2b42-66a5-4a57-8e99-468c36ebb6f2",              # (5)
      "my_secret": "sEh9ZHPoKX8QYU-v0Noe0ZPzxGBF.....................iBn5fn_eFM", # (6)
      "type": "resend",                                             # (11)
      "protocol": "mod_http",                                       # (12)
      ...
    }
  ],
  "loggers": [
    {
      "connections": [{
          "channel": 1,                                               # (13)
          ...
      }],
      ...
    },
    {
      "connections": [{
          "channel": 2,                                               # (14)
          ...
      }],
      ...
    }
  ],
  ...
}
```

Number	Field	Description
(1)	`name`	The type of filter, `channel`.
(2)	`channel`	The channel to be filtered. Set to "-1" which means that all channels are filtered.
(3)	`target`	How to send the filtered content. Set to `realtime`.
(4)	`setting`	Advanced filter settings. Leave the settings empty.
(5)	`my_id`	The UUID of the sending edge (this edge).
(6)	`my_secret`	Client secret for the sending edge (this edge). Reference: Edge information setting (p. 11).
(7)	`type`	Sending client type, `realtime`.
(8)	`protocol`	Communication protocol of the sending client. Specify `mod_websocket.v2`, as the Realtime and Bulk clients use the Realtime API.
(9)	`type`	Sending client type, `bulk`.
(10)	`store_cycle`	Sending interval of the Bulk client. Set this to 0 to delegate data transmission to the Resend client.
(11)	`type`	Sending client type, `resend`.
(12)	`protocol`	Communication protocol of the sending client. In this example, the Resend client uses the REST API, so specify `mod_http`.
(13)	`channel`	The channel to be used for the acquired data. In this example, this Device Connector (logger) is channel 1.
(14)	`channel`	Channel to be assigned to the acquired data. In this example, this Device Connector (logger) is channel 2.

3.5 Settings related to saving RAW data

For more information on RAW data, see RAW data (p. 42).

The following settings can be made for RAW data.

3.5.1 Preventing any data from being saved as RAW data

To prevent any data from being saved as RAW data, set `manager.rawdir` to null.

Example (No RAW data saved):

```
{
  "manager": {
    ...
    "rawdir": null,
    ...
  },
  ...
}
```

3.5.2 Preventing the storage of RAW data for a specific channel

Add `loggers[].connections[].disable_raw` to the Device Connector settings and set the value to `1`.

Example (Disabling RAW data storage for channel 0):

```
{
  "loggers": [{
    "connections": [{
      ...
      "disable_raw": 1,
      "channel": 0,
      ...
    }],
    ...
  }],
  ...
}
```

04 Add your own Device Connector

By developing a Device Connector that reads data from a device and passes it to the Agent, you can send the data retrieved from any device to the intdash server. The transfer of data between the Device Connector and the Agent is done using FIFO.

4.1 Configure to use your own Device Connector

The Agent and the Device Connector are connected by a FIFO. When data from the device is sent to the Agent, the Device Connector writes the data to the FIFO in a given format and the Agent reads it. Conversely, when the Agent sends data to the device, the Agent writes the data to the FIFO in a given format and the Device Connector reads it.

The FIFO is provided by the plug-in. To add the FIFO plug-in, add the following items to the configuration file. If you start the Agent with the FIFO plug-in added, a FIFO file for communication between the Device Connector and the plug-in will be generated.

> **Note:** However, the Device Connector `intdash-edge-logger` provided by Aptpod does not use the FIFO plugin. Therefore, when using `intdash-edge-logger`, it is not necessary to include `plugin` in the configuration file.

Example (Adding a FIFO for channel 2):

```
{
  ...
  loggers: [{
      "connections": [{
        "channel": 2,
        "fifo_tx": "/var/run/intdash/logger_002.tx",
        "fifo_rx": "/var/run/intdash/logger_002.rx"
      }],
      "details": {
        "plugin": "fifo"
      },
  }],
  ...
}
```

> **Note:** The "logger" in the configuration file refers to the Device Connector.

When you start the Agent with this setting, two FIFO files `/var/run/intdash/logger_002.tx` and `/var/run/intdash/logger_002.rx` are created for communication between the Agent and the Device Connector. The Device Connector must write data to `/var/run/intdash/logger_002.tx` when sending data to the Agent. The Device Connector must read data from `/var/run/intdash/logger_002.rx` when receiving data from the Agent.

> **Note:** When the Agent sends data to the intdash server, the `channel` set in the configuration file is used as the channel number.

4.2 Write to FIFO from Device Connector

To send data from the Device Connector to the Agent, the Device Connector needs to write the data to the FIFO generated by the Agent.

4.2.1 Writing data

The data to be written to the FIFO must follow a pre-defined format. Check FIFO data format used between Agent and Device Connector (p. 46) for the format.

4.3 Read data from FIFO

To set up downstream, you need to add a Control client to the `clients` section of the configuration file. The data that the Device Connector can read from the FIFO is the data on the channels that are configured for downstream in the Control client and that are configured for the Device Connector.

Example (Device Connector settings):

```
{
  ...
  loggers: [{
      "connections": [{
        "channel": 2,
        "channel_rx": -1,
        "receive_basetime": true,
        "fifo_tx": "/var/run/intdash/logger_002.tx",
        "fifo_rx": "/var/run/intdash/logger_002.rx"
      }],
      "details": {
        "plugin":"fifo"
      },
  }],
  ...
}
```

The format of data to be read from the FIFO is the same as the format for writing (FIFO data format used between Agent and Device Connector (p. 46)).

The data received by one Device Connector is limited to the data on one channel. If `channel_rx` does not exist or -1 is set, the channel number set in `channel` will be received. If `channel_rx` is set to a value between 0 and 255, the channel number set in `channel_rx` will be received.

Whenever a downstream connection is made to the intdash server (including a reconnection), the base time is received. If the Device Connector does not want to receive the base time data, specify `false` for `receive_basetime`.

Example (Settings for Control client):

```
{
  "clients": [{
    ...
    "my_id": "c35618bf-aa2c-4abc-8a4e-5b157b90c9ef",
    "my_secret": "hsNxJhvDNHR2QcXbXl.......................Z0RWKvfPs_neAkjTNSO5",
    "down_dst_id": "00000000-0000-0000-0000-000000000000",
    "ctlr_id": "9defc535-4640-4c5e-934a-bb435a89a64f",
    "ctlr_flts": [
      {
```

(continues on next page)

(continued from previous page)

```
        "channel": 2,
        "dtype": 10,
        "ids": ["string_a", "string_b"]
      },
      {
        "channel": 3,
        "dtype": 14,
        "ids": ["data_a", "data_b"]
      }
    ],
    "type": "control",
    ...
  }],
  ...
}
```

In this Control client configuration example, the following two types of data sent from the edge UUID `9defc535-4640-4c5e-934a-bb435a89a64f` are received via the downstream.

- Channel number: 2, iSCP data type: String (10), data ID: "string_a" or "string_b"
- Channel number: 3, iSCP data type: Bytes (14), data ID: "data_a" or "data_b"

When the Device Connector and Control client are configured as above, the Device Connector receives the data sent from the edge UUID `9defc535-4640-4c5e-934a-bb435a89a64f`, channel number: 2, iSCP data type: String (10), data ID: "string_a" or "string_b". Since only channel 2 is specified in the Device Connector settings, data with channel number 3 cannot be received.

4.4 Automatic startup and termination of Device Connectors

The Device Connector can be started and shut down at any time. If you want the Device Connector to start and end together with the Agent, you can do it in the following ways.

4.4.1 Automatically start the Device Connector when the Agent starts

You can configure the Device Connector to start when the Agent starts.

Specify the Device Connector startup command in `path` of the configuration file, and specify the Device Connector configuration file in `conf`. With this setting, when the Agent starts, the command to start the Device Connector is executed in the form of `path -C conf`.

```
"loggers": [{
  "path": "/opt/vm2m/sbin/test-logger",
  "conf": "/etc/opt/intdash/test-logger.conf",
  "connections": [{
    "channel": 3,
    ...
  }],
  ...
}]
```

4.4.2 Terminate the Device Connector with a signal from the Agent

When exiting the Agent, the Agent sends a SIGTERM to the Device Connector. Make sure that your Device Connector detects the signal and performs the termination process.

05 Contact Us

If you have any questions or problems, please contact us using the contact information below.

aptpod, Inc.

- Email address (customer support) VM2M-support@aptpod.co.jp
- Website https://www.aptpod.co.jp/en/

When making inquiries, please let us know the following.

- intdash Edge version
- All config files you are using
  - Manager configuration file (manager.conf) (Please delete the tokens contained in the file before sending.)
  - Device connector configuration files (e.g. logger.conf.xxx)

06 Appendix

6.1 Base time

The base time is information that represents the start time of measurement. The Agent gets the base time as needed and sends it to intdash.

> **Note:** "Measurement" refers to a collection of time series data sent from a particular edge. For details, refer to the separate document on iSCP v1.

There are two types of base times.

Base time by EdgeRTC
: The real-time clock (RTC) base time of the system running the Agent.

Base time by NTP
: The base time by the clock synchronized with the NTP server. The Status plugin must be enabled to use the NTP base time (The Status plugin is enabled in the default configuration file).

Base time type	Timing to determine the base time	Timing to send the base time to intdash
EdgeRTC	When the Agent starts	At the start of data transmission
NTP	After the Status plugin has communicated with the NTP server	Fixed cycle (1-minute interval for 10 minutes after startup, then 10-minute interval)

6.2 Filter at the sender's side

The time series data collected by the manager through the Device Connector is filtered by the manager and distributed to each client such as the Realtime client and the Bulk client, and sent to the intdash server.

Filters give you the flexibility to switch between data transmission methods.

6.2.1 Filtering process on the sending side

For each time series data, the manager decides whether to send the data using each client (pass) or not (drop).

- Unless otherwise specified, time series data is sent to the intdash server via the Realtime client.
- Data that was not sent by the Realtime client because it was "dropped" by the filter for the Realtime client is sent by the Bulk client.
- Data that is "dropped" by the filter for the Bulk client will not be sent to the intdash server.

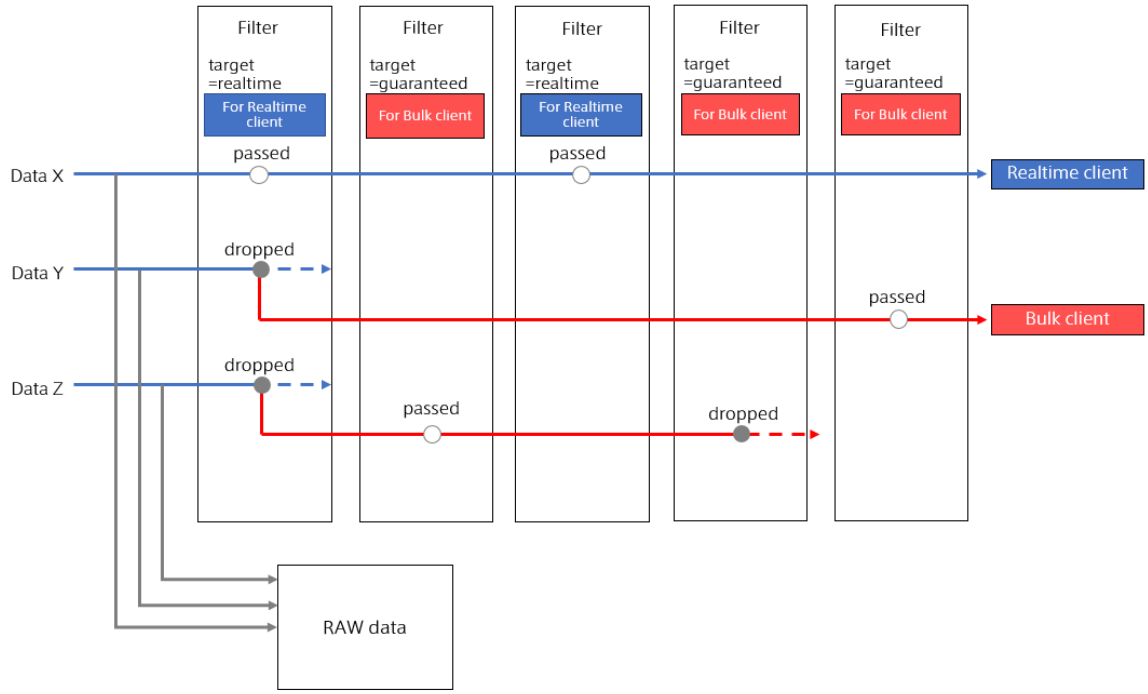
In either case, the time series data is dumped as RAW data by the manager.

Realtime client	Bulk client	How the data is handled
Passed	-	Data is sent in real time to the intdash server and saved on the intdash server.
Dropped	Passed	Data is sent to the intdash server by batch transmission at regular intervals and saved on the intdash server.
Dropped	Dropped	Data is not sent to the intdash server in real time. It is not saved on the intdash server. It is saved as RAW data (p. 42) in the local storage on the edge device.

6.2.2 Filter configuration

Filtering is performed by multiple filters. Multiple filters are evaluated sequentially to determine which client should send each time series data.

For example, if the following filter configuration has been set up, data is sent as follows.



- Data X does not correspond to any filter and is sent by the Realtime client.
- Data Y is dropped by the Realtime client by the first filter and is sent by the Bulk client.
- Data Z is not sent because the first filter drops it from the Realtime client and the fourth filter also drops it from the Bulk client.

6.2.3 Common filter settings

The settings for the filter are made in `manager.filters[]` of the configuration file. The contents of the objects stored in the "filters" array should be as follows.

Key	Type	Description
`name`	string	See the table for each filter in Filter type (p. 38).
`channel`	string	Set the channel to which the filter is applied. You can set "-1", "0" to "255". In case of "-1", all channels are targeted.
`target`	string	Sets the client to which the filter is applied. (`realtime|guaranteed|both`)
`setting`	object[]	Filter setting. See the description of each filter in Filter type (p. 38).

For `target`, specify one of the following.

- `realtime`: Filter for the Realtime client.
- `guaranteed`: Filter for the Bulk client.
- `both`: Filter for both the Realtime client and the Bulk client.

6.2.4 Filter type

There are the following types of filters.

Some filters apply only to data of a specific data type, while others apply to all data types.

Filter name	Data types to which you can apply filters	Description
sampling filter (p. 38)	CAN, NMEA, Motion JPEG, String, Float, Int, Bytes only	Sampling filter. Thins out the data.
can_id filter (p. 40)	CAN only	Allow-list type CAN ID filter. Passes the data with the specified CAN ID.
can_mask filter (p. 40)	CAN only	Block-list type CAN ID filter. Drops the data with the specified CAN ID.
channel filter (p. 41)	Any	Block-list type channel filter. Drops the data for the specified channel.
duplicate filter (p. 41)	Any	Data replication filter. Duplicate the same data to the Realtime and Bulk clients.

sampling filter

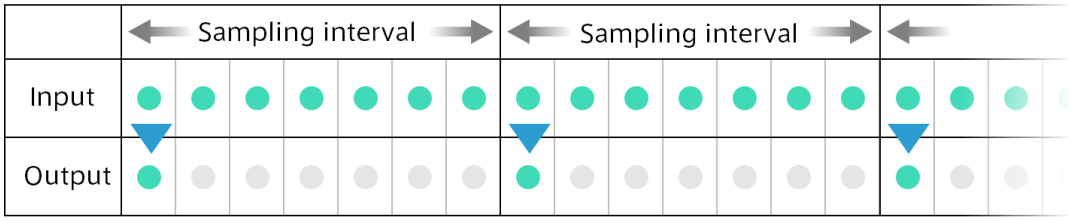
When a sampling filter is set for a channel, data is extracted so that there is one data point for each data ID within the time range specified as the sampling interval.

In each sampling interval, data points are processed as follows.

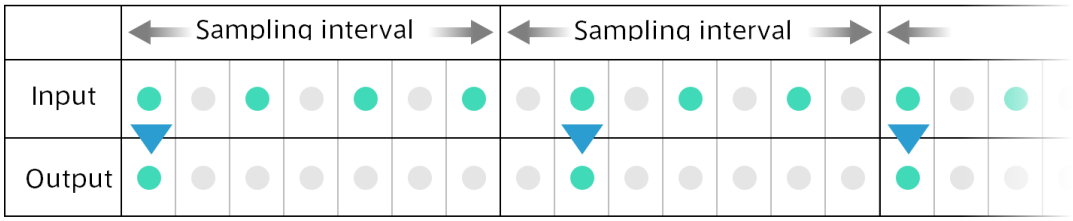
- The first data point of each data ID will pass.
- The second and subsequent data points with the same data ID will be discarded.
- If there is no distinction by data ID, as in MotionJPEG, the first data point will pass.

The following figure shows how CAN data with three different frequencies (ID 1, 2, and 3) are filtered by a sampling filter. In each sampling interval, the first data point of the data with ID 1 passes the filter. Similarly, in each sampling interval, the first data point with ID 2 and the first data point with ID 3 pass the filter.

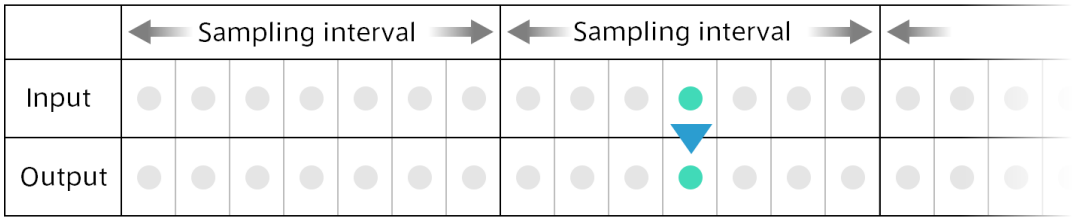
CAN ID : 1



CAN ID : 2



CAN ID : 3



Contents of the "setting" object of the sampling filter

Key	Value
rate	Sampling interval [msec]

Example (Sampling real-time data of channel 1 with a sampling interval of 1000 [msec]):

```
{
  "manager": {
    ...
    "filters": [{
      "name": "sampling",
      "channel": "1",
      "target": "realtime",
      "setting": [{"key": "rate", "value": "1000"}]
    }]
  },
  ...
}
```

can_id filter

An allow-list type filter that passes data with a specific CAN ID and drops other data.

- Passes only the data with the specified CAN ID.
  - Multiple CAN IDs can be set.
  - Drops data that does not have a specified CAN ID
  - When setting the extended CAN ID, enter a number with the first bit set to 1, such as `0x80000000`.

Contents of "setting" object of can_id filter

Key	Value
id	CAN ID to pass

Example (Only CAN IDs 0x00000010 and 0x00000020 in Realtime channel 2 pass):

```
{
  "manager": {
    ...
    "filters": [{
      "name": "can_id",
      "channel": "2",
      "target": "realtime",
      "setting": [
        {"key": "id", "value": "16"},
        {"key": "id", "value": "32"}
      ]
    }]
  },
  ...
}
```

can_mask filter

A block list type filter that drops data of a specific CAN ID.

- Drops the data with the specified CAN ID.
  - Multiple CAN IDs can be set.
  - Passes data that does not have a specified CAN ID
  - When setting the extended CAN ID, enter a number with the first bit set to 1, such as `0x80000000`.

Contents of the filter "setting" object

Key	Value
id	CAN ID to be dropped

Example (Data with extended CAN ID 0x00000010 in Bulk channel 1 are dropped):

```
{
  "manager": {
    ...
    "filters": [{
      "name": "can_mask",
      "channel": "1",
      "target": "guaranteed",
      "setting": [
```

(continues on next page)

(continued from previous page)

```
        {"key": "id", "value": "2147483664"}
      ]
    }]
  },
  ...
}
```

2147483664 is a decimal notation with the first bit of 0x00000010 set to 1.

channel filter

A block list type filter that drops a specific channel.

- Drops the data of the specified channel

Contents of the filter "setting" object

- None

Example (Data in Realtime channel 1 are dropped):

```
{
  "manager": {
    ...
    "filters": [{
      "name": "channel",
      "channel": "1",
      "target": "realtime",
      "setting": []
    }]
  },
  ...
}
```

duplicate filter

Duplicate the data.

- Duplicate the data and send the same data to the Realtime and Bulk clients

Contents of the filter "setting" object

- None

Example (copy Realtime channel 1 data and send it to the Bulk client):

```
{
  "manager": {
    ...
    "filters": [{
      "name": "channel",
      "channel": "1",
      "target": "realtime",
      "setting": []
    }]
  },
  ...
}
```

6.3 RAW data

All the data input to the Agent from the Device Connectors can be saved as RAW data in the dump file.

6.3.1 Saving and automatic deletion of RAW data

It is possible to set whether to save RAW data. By default, the data is saved.

To prevent your disk from filling up by storing your RAW data, you can automatically delete your RAW data.

```
{
  "manager": {
    "rawdir": "/var/lib/intdash/raw",
    "raw_autodelete": true,
    "raw_autodelete_th": 85,
    ...
  },
  ...
}
```

If `raw_autodelete` is set to true, the RAW data will be automatically deleted when the usage of the partition storing the RAW data reaches a certain level. The auto-delete feature deletes older RAW data first, until the partition containing `rawdir` uses less than `raw_autodelete_th` of disk space.

6.3.2 Destination and file structure of the RAW data

A directory for each measurement is created in the RAW data storage directory, and the dump files are saved in that directory.

```
+ /opt/vm2m/var/lib/intdash
  + raw
    + 1562549837.123456789     # (1)
      - 001_000.raw            # (2)
      - 002_000.raw            # (3)
    + 1562549837.999999999
      - 001_000.raw
      - 001_001.raw
```

Number	Description
(1)	Measurement directory
(2)	Dump of data from channel 1 Device Connector
(3)	Dump of data from channel 2 Device Connector

Measurement directory

A new measurement directory is created for each measurement. The directory name is the Edge RTC base time of the measurement in Unix time down to nanoseconds. (Example: The directory name 1562549837.123456789 means the base time 2019-07-08T01:37:17.123456789 UTC.)

Dump file

A dump file is created for each Device Connector. The format of the file name is as follows:

`XXX_NNN.raw`

- XXX: Device connector channel number (decimal number: 000-255)
- NNN: Counter number (decimal number: 000-999)

If the size of the dump file is 512MB or more, a new file will be created with the next counter number.

Dump file format

The dump file is a direct dump of data in FIFO data format used between Agent and Device Connector (p. 46).

Sample (NMEA)

```
       0      1             4             8             12     13     14            18
       +------+---- - - ----+---- - - ----+---- - - ----+------+------+---- - - ----+------+------+---//-+
Data1  | Type | Length      | Time Sec    | Time Nano   |DType |SeqNo | NMMEA Size  | NMEA String
       +------+---- - - ----+---- - - ----+---- - - ----+------+------+---- - - ----+------+------+---//-+
Data2  | Type | Length      | Time Sec    | Time Nano   |DType |SeqNo | NMMEA Size  | NMEA String
       +------+---- - - ----+---- - - ----+---- - - ----+------+------+---- - - ----+------+------+---//-+
Data3  | Type | Length      | Time Sec    | Time Nano   |DType |SeqNo | NMMEA Size  | NMEA String
       +------+---- - - ----+---- - - ----+---- - - ----+------+------+---- - - ----+------+------+---//-+
Data4  | Type | Length      | Time Sec    | Time Nano   |DType |SeqNo | NMMEA Size  | NMEA String
       +------+---- - - ----+---- - - ----+---- - - ----+------+------+---- - - ----+------+------+---//-+
```

6.3.3 Tool for analyzing RAW data

Tools for analyzing RAW data are installed with the Agent.

Usage example:

```
/opt/vm2m/bin/rawutil -P hexdump /opt/vm2m/var/lib/intdash/raw/1562551394.826226991/000_000.raw
```

```
timestamp              datatype        size    value
        0.099532866            16          72    00: 24 47 4e 52 4d 43 2c 30 32 30 33 31 34 2e 38 30 2c 41 ...
        0.104995269            16          39    00: 24 47 4e 56 54 47 2c 2c 54 2c 2c 4d 2c 30 2e 34 32 31 ...
        0.118923030            16          82    00: 24 47 4e 47 47 41 2c 30 32 30 33 31 34 2e 38 30 2c 33 ...
        0.128784514            16          62    00: 24 47 4e 47 53 41 2c 41 2c 33 2c 31 38 2c 30 31 2c 31 ...
```

For more information on how to use the tool, see the `rawutil` help.

```
/opt/vm2m/bin/rawutil -h
```

6.4 Retransmission data

Data that is tried to be sent by a Realtime or Bulk client but fails is retained as retransmission data. The time-series data held as retransmission data is sequentially retransmitted by the Resend client.

6.4.1 Automatic stop when disk space is low

If the disk usage exceeds the threshold, the Agent stops automatically. The conditions are as follows ("[]" represents the key in the configuration file.):

- `[manager.required_space]` < Usage of the partition where retransmission data is stored (%)
- `[manager.required_space_raw]` < Usage of the partition containing `[manager.rawdir]` (%)

6.4.2 Save destination and file structure

A measurement directory is created under the retransmission data directory, and the retransmission section file is saved in it.

```
+ /opt/vm2m/var/lib/intdash
  + meas
    + <SERVER_NAME>                              # (1)
      + CCCCCCCCC_TTTTTTTTTT.NNNNNNNNN           # (2)
        - .meas.<MEAS_UUID>                      # (3)
        - .meta                                  # (4)
        - SSSSSSSSSS_TTTTTTTTTT.NNNNNNNNNF.EXT   # (5)
        - SSSSSSSSSS_TTTTTTTTTT.NNNNNNNNNF.EXT
    + <SERVER_NAME>
      + CCCCCCCCC_TTTTTTTTTT.NNNNNNNNN
        - .meas.<MEAS_UUID>
        - .meta
        - SSSSSSSSSS_TTTTTTTTTT.NNNNNNNNNF.EXT
        - SSSSSSSSSS_TTTTTTTTTT.NNNNNNNNNF.EXT
```

Number	Description
(1)	Server directory
(2)	Measurement directory
(3)	UUID of measurement
(4)	Measurement metadata
(5)	Retransmission section

Server directory

The server directory stores the measurement data to be sent to this server. The directory name is the name of the server.

Measurement directory

A new measurement directory is created for each measurement. The format of the directory name is as follows:

`CCCCCCCCC_TTTTTTTTTT.NNNNNNNNN`

- `CCCCCCCCC`: Number of measurements
- `TTTTTTTTTT.NNNNNNNNN`: EdgeRTC base time of the measurement (Unix timestamp down to nanoseconds)

Measurement UUID file

The measurement UUID file is an empty file that has the measurement UUID in the filename. This file is created to make it easier for users to find measurements. The file name is `.meas.<first 8 digits of measurement UUID>`.

Measurement metadata file

The following information about measurement is stored in the measurement metadata file.

- Serial number (Serial number of the last section)
- Unit count (Total number of units)
- Number of retransmission section files
- Retransmission file size (Total retransmission file size)
- Measurement UUID flag (Indicates whether the measurement UUID was obtained)
- End flag (Indicates whether the edge notified the server that the measurement is finished.)
- Measurement tag flag (Indicates whether the measurement tag was sent to the server)
- Measurement count (Counter number given to the measurements created in this terminal)
- EdgeRTC base time
- Measurement duration

The file name is `.meta`.

Retransmission section file

A retransmission section file is a file that dumps only the units in a particular intdash Section.

The file name format is `SSSSSSSSSS_TTTTTTTTTT.NNNNNNNNNF.EXT`.

- S: Section serial number
- T: Relative time (seconds) from EdgeRTC
- N: Relative time from Edge RTC (nanoseconds)
- F: Flag (B: Includes base time, None: The base time is not included.)
- EXT: Extension (bin: retransmission data, store: data for the Bulk client, store.bin: retransmission data from the Bulk client)

Example:

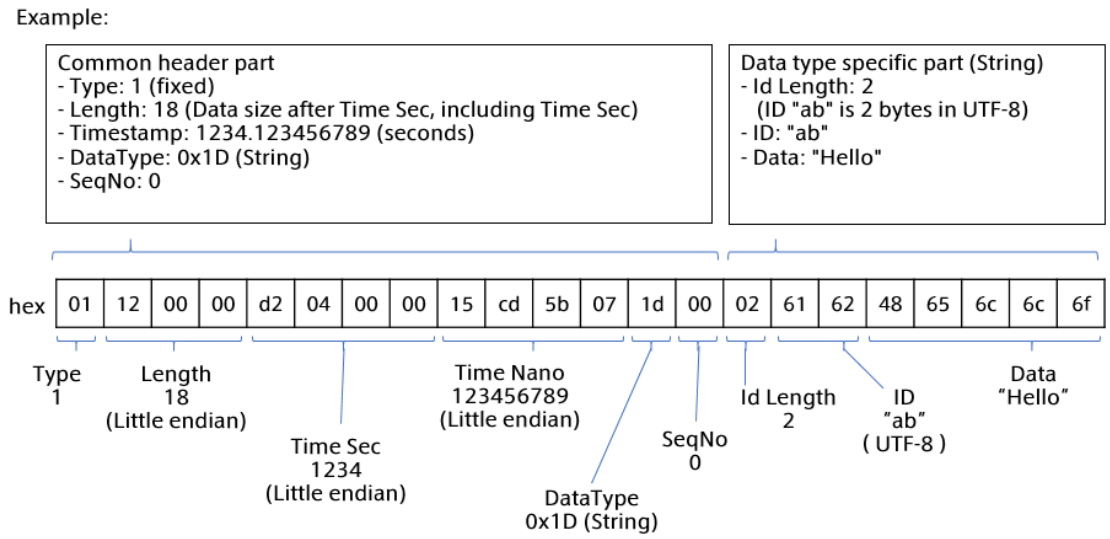
```
0000000000_0000000013.517448529B.bin
0000000001_0000000014.002924135.bin
0000000002_0000000015.000175599.store
0000000003_0000000016.002819513.store.bin
```

6.5 FIFO data format used between Agent and Device Connector

The data format for the Agent FIFO consists of a common header part and a data-type specific part.

- See Common header (p. 46) for the common header part.
- See Data type-specific part (p. 47) for the specific parts of each data type.

The following is an example of the primitive string type "Hello".



Important: The data format and the data type used by the FIFO between the Agent and the Device Connector is different from those of iSCP v1.

6.5.1 Common header

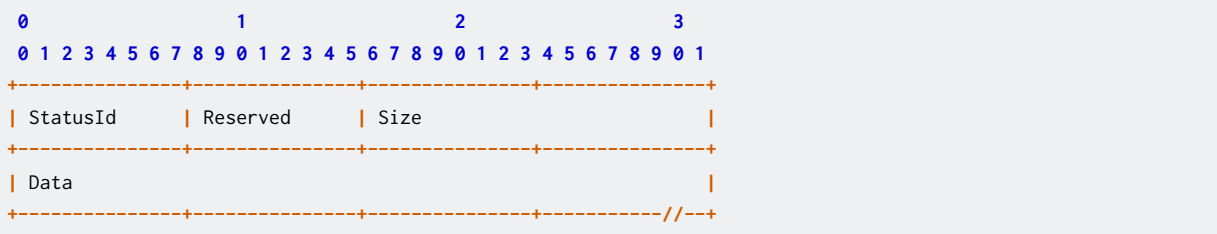
```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
| Type          | Length                                        |
+---------------+---------------+---------------+---------------+
| Time Sec                                                      |
+---------------+---------------+---------------+---------------+
| Time Nano                                                     |
+---------------+---------------+---------------+---------------+
| DataType      | SeqNo         | Data                          |
+---------------+---------------+---------------+-----------//--+
```

Field name	Byte length	Endian	Signed	Value	Description
Type	1	—	No	1	Message type (fixed at 1)
Length	3	LE	No	10-16777216	Size after Time Sec (including Time Sec)
Time Sec	4	LE	No	0-4294967295	Monotonically increasing system time (seconds)[*1]
Time Nano	4	LE	No	0-999999999	Monotonically increasing system time (nanoseconds)
DataType	1	—	No	—	FIFO data type codes (see data type table below)
SeqNo	1	—	No	0-255	Sequential number (can be fixed at zero.)
Data	0-16777208	—	—	—	Data (see section for each FIFO data type)

Data types

6.5.2 Data type-specific part

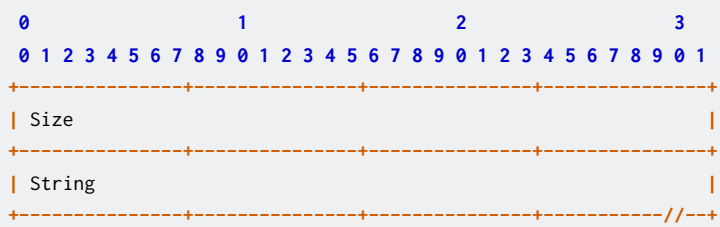
Status (data type: 0x03)



[*1] In programming languages that can use POSIX, you can get this data by specifying `CLOCK_MONOTONIC_RAW` in `clock_gettime()` function.

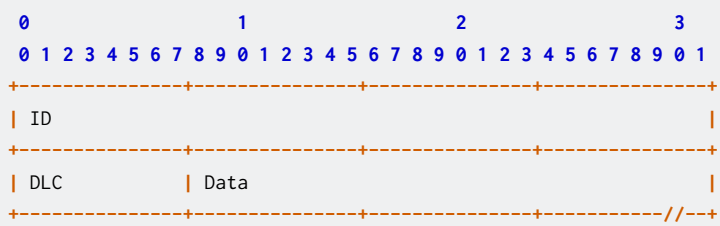
Field name	Byte length	Endian	Signed	Value	Description
StatusId	1	—	No	0x90	Status type (fixed at 0x90)
Size	2	LE	No	0-32767	Data size
Data	0-32767	—	—	—	JSON string

NMEA (data type: 0x10)



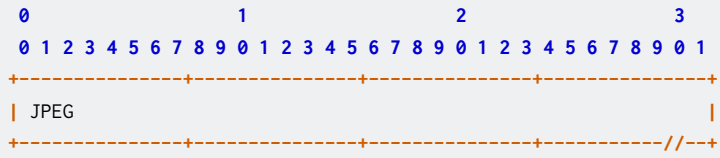
Field name	Byte length	Endian	Signed	Value	Description
Size	4	LE	No	0-4294967295	Data size
Data	0-4294967295	—	—	—	NMEA string

CAN / CAN-FD (data type: 0x11)



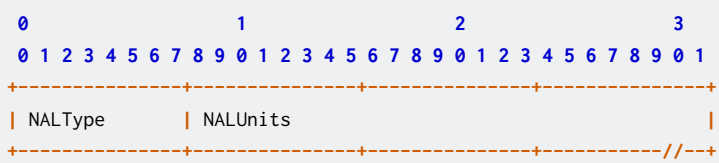
Field name	Byte length	Endian	Signed	Value	Description
ID	4	LE	No	0-4294967295	CAN ID (set the first bit to 1 in the case of extended CAN)
DLC	1	—	No	0-255	Data size
Data	0-255	—	—	—	data

JPEG (data type: 0x12)



Field name	Byte length	Endian	Signed	Value	Description
JPEG	0-16777216	—	—	—	JPEG (ISO / IEC 10918-1, Annex B) binary data

H.264 (data type: 0x1C)

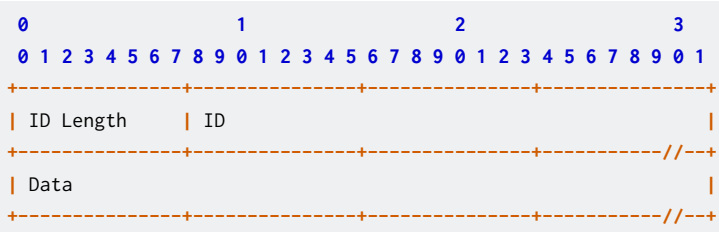


Field name	Byte length	Endian	Signed	Value	Description
NALType	1	—	—	—	NALType (see NAL Type below)
NALUnits	—	—	—	—	NALUnits

NALType

NALType	Data to be stored in NAL Units
0x00	Concatenation of NAL Units in 1 frame[*2] which are IDR slices (nal_unit_type == 5)[*3]
0x01	Concatenation of NAL Units in 1 frame which are non-IDR slices (nal_unit_type == 1)
0x08	Concatenation of NAL Units[*4] required for H.264 decoding. It is assumed that one is generated in advance for each NALType 0x00.

String (Data type: 0x1D) Primitive string type



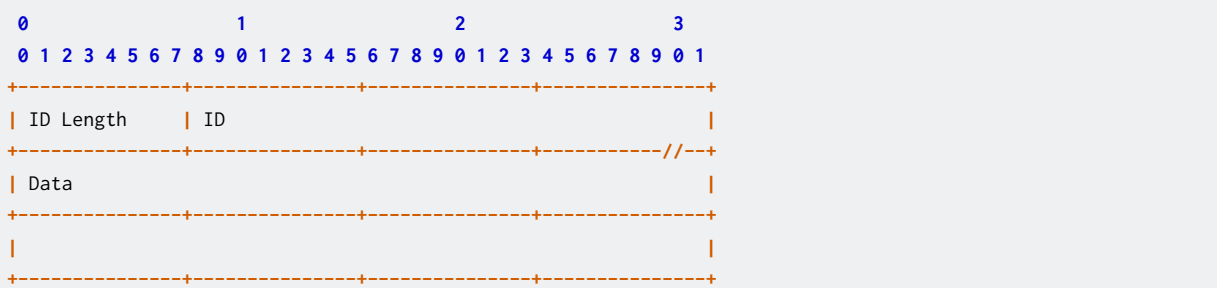
Field name	Byte length	Endian	Signed	Value	Description
ID Length	1	—	—	0-255	ID length
ID	0-255	—	—	0-16777215	UTF-8 encoded ID
Data	—	—	—	—	String

[*2] The sequence of NAL units of the same `nal_unit_type` 1-5 (Coded slice), starting from the unit where `first_mb_in_slice_header` in `slice_header` is 0, ending with the unit before the next 0.

[*3] According to "Byte stream format (Annex B)" in "ITU-T Rec. H.264 ¦ ISO/IEC 14496-10 Advanced Video Coding", concatenate as follows: start code prefix + NAL unit + start code prefix + NAL unit ... start code prefix + NAL unit

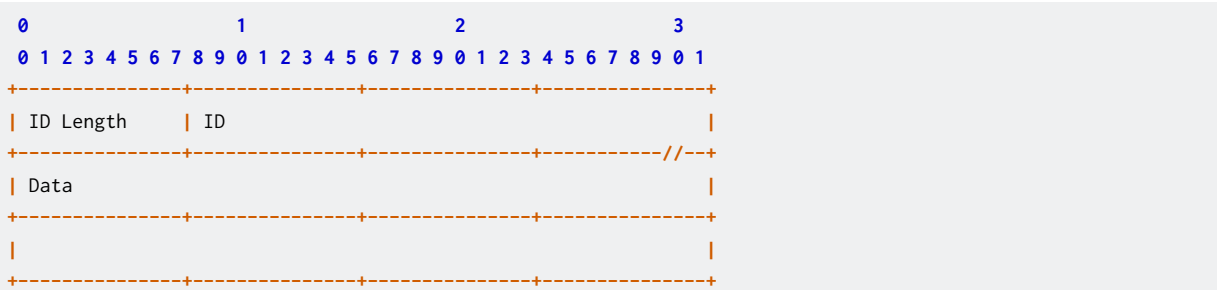
[*4] The NAL units required for H.264 decoding are SPS (`nal_unit_type == 7`) and PPS (`nal_unit_type == 8`).

Float (data type: 0x01E) Primitive Float64 type



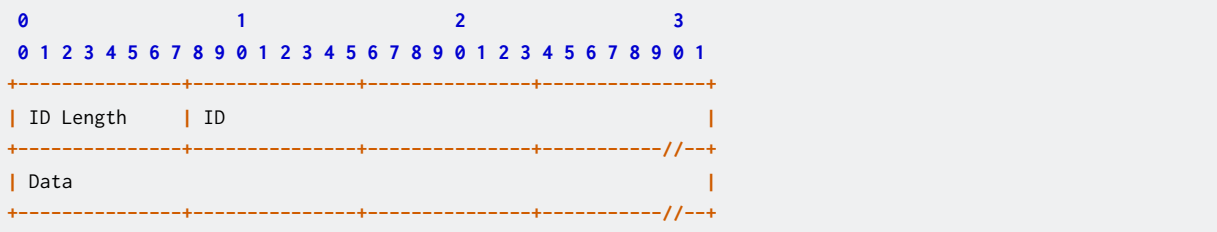
Field name	Byte length	Endian	Signed	Value	Description
ID Length	1	—	—	0-255	ID length
ID	0-255	—	—	0-16777215	UTF-8 encoded ID
Data	8	LE	—	—	Byte string containing double-precision floating point number (based on IEEE 754)

Int (data type: 0x1F) Primitive Int64 type



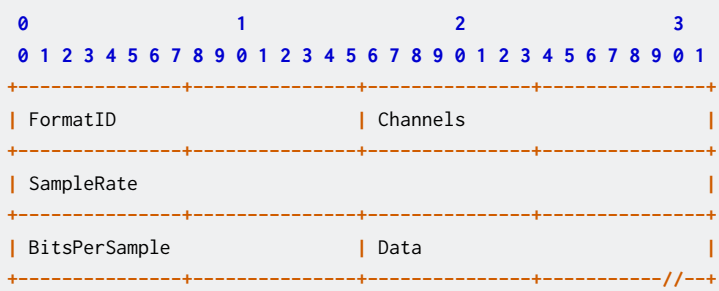
Field name	Byte length	Endian	Signed	Value	Description
ID Length	1	—	—	0-255	ID length
ID	0-255	—	—	0-16777215	UTF-8 encoded ID
Data	8	LE	Yes	—	Int64

Bytes (data type: 0x20) Primitive byte array type



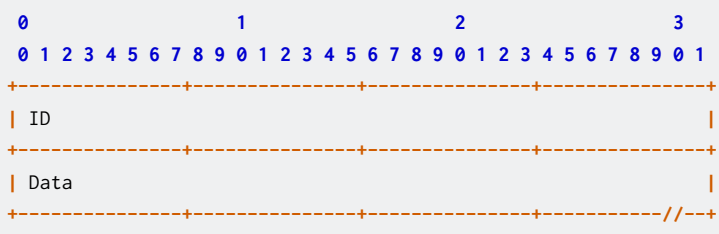
Field name	Byte length	Endian	Signed	Value	Description
ID Length	1	—	—	0-255	ID length
ID	0-255	—	—	0-16777215	UTF-8 encoded ID
Data	—	—	—	—	Binary data

PCM (data type: 0x22)



Field name	Byte length	Endian	Signed	Value	Description
FormatID	2	LE	No	0-65535	Format ID defined in WAVE[5]
Channels	2	LE	No	0-65535	Number of audio channels stored
SampleRate	4	LE	No	0-4294967295	Sampling frequency [Hz]
BitsPerSample	2	LE	No	0-65535	Bit rate [bit/sample]
Data	—	—	—	—	Waveform information conforming to WAVE[5]

Generic (data type: 0x7F)



Field name	Byte length	Endian	Signed	Value	Description
ID	4	LE	No	0-4294967295	Numeric ID
Data	0-65531	—	—	—	Arbitrary data

[5] WAVE (RIFF waveform Audio Format), a container format for audio data

6.6 All settings for Agent

This section describes all items in the Agent configuration file.

The configuration file has the following structure. Refer to the following sections for details.

> **Note:** The "logger" in the configuration file refers to the Device Connector.

Key	Type	Description
manager	object	Manager settings (p. 53)
clients	object[]	Client settings (p. 54)
loggers	object[]	Device connector settings (p. 57)

Example

```
{
  "manager": {
    ...
  },
  "clients": [{
    ...
  }],
  "loggers": [{
    ...
  }]
}
```

6.6.1 Replacing settings with environment variables

The string type setting values in the configuration file can be given by the environment variables. By giving the settings in the environment variables, you can flexibly change the settings at startup without rewriting the configuration file.

For example, to give a value to `clients.my_secret` using an environment variable, use the variable in the configuration file as follows. In the example below, the variable `$SECRET` is used.

```
{
  ...
  "clients": [
    {
      ...
      "my_secret": "$SECRET",
    }],
   ...
}
```

Then, set the environment variable with a prefix `INTDASH_EDGE_` added to the above variable name. In this example, set a variable `INTDASH_EDGE_SECRET`.

When intdash-edge-manager is started using this configuration file, `$SECRET` in the configuration file will be expanded to the value of the environment variable `INTDASH_EDGE_SECRET`.

> **Note:** Only string-type values can be replaced by environment variables.

6.6.2 Manager settings

Settings related to the manager are set in the `manager` field of the configuration file as an object. The items are as follows.

Key	Type	Default value	Description
`workdirs`	string[]	`["/opt/vm2m/var/lib/intdash/meas", "/opt/vm2m/var/run/intdash"]`	Directory path to be created at startup
`basetime`	string	`"/opt/vm2m/var/run/intdash/basetime"`	Path of the file that stores the base time information
`meas_root`	string	`"/opt/vm2m/var/lib/intdash/meas"`	Directory path to store the retransmission data
`rawdir`	string	`"/opt/vm2m/var/lib/intdash/raw"`	Directory path to save RAW data
`raw_autodelete`	bool	`true`	Automatic deletion of RAW data
`raw_autodelete_th`	number	`85`	Threshold for automatically deleting RAW data [%]
`required_space`	number	`90`	Threshold at which the Agent stops measurement [%]. The measurement is automatically stopped when the free space ratio of the partition that stores the retransmission data becomes larger than this value. If you set it to 100, it will not stop automatically.
`required_space_raw`	number	`90`	Threshold at which the Agent stops measurement [%]. The measurement is automatically stopped when the free space ratio of the partition that stores RAW becomes larger than this value. If you set it to 100, it will not stop automatically.
`stat`	string	`"/opt/vm2m/var/run/intdash/manager.stat"`	Path to the file that records the status of the Manager
`process_stat`	string	`"/opt/vm2m/var/run/intdash/process.stat"`	Path of the file that records the status of the Process
`wwan_stat`	string	`"/opt/vm2m/var/run/intdash/wwan.stat"`	Path of the file that records the WWAN status

continues on next page

Table 1 – continued from previous page

Key	Type	Default value	Description
logger_stat	string	"/opt/vm2m/var/run/intdash/logger_%03hhu.stat"	Format of the file path to record the status of the Device Connector. The first format specifier (%03hhu in the default value) is replaced with the channel number.
system_stat	string	"/opt/vm2m/var/run/intdash/system.stat"	Path to the file that records the status of the System
filters	object[]	[]	Filter settings. See Filter at the sender's side (p. 36) for more information.

6.6.3 Client settings

The settings for clients are in the clients field of the configuration file as an array of objects. The items are as follows.

Key	Type	Default value	Description
protocol	string	""	Name of communication library to use (mod_websocket.v2|mod_http)
type	string	""	Operating mode of the client (realtime|bulk|resend|control). If the protocol is mod_http, only resend can be selected.
mode	number	1	Whether to persist data to the server (0: do not persist, 1: persist). This setting is only used if type is realtime|bulk|resend and protocol is mod_websocket.v2.
my_id	string	""	Edge UUID used to connect to intdash
my_secret	string	""	Client secret used to connect to intdash
auth_path	string	"/opt/vm2m/var/lib/intdash/.auth"	The path to the file where the connection information is saved. You must have write access to this file.
my_token	string	""	Edge token used to connect to intdash
dst_id	string[]	[]	UUID of the destination edge. Used when "type" is realtime|bulk|resend. (optional)

continues on next page

Table 2 – continued from previous page

Key	Type	Default value	Description
down_dst_id	string	""	Only data with the destination UUID specified here will be received from the server. However, if 00000000-0000-0000-0000-000000000000 is specified, data for any destination will be received. If the specified string cannot be parsed as a UUID (including the default "" case), the UUID set in my_id is assumed and only data for my_id is received from the server. Used when "type" is "control".
ctlr_id	string	""	UUID of the source edge. Used when "type" is "control".
ctlr_flts	object[]	[]	Filters to be applied to data received from the server. See ctlr_flts.channel, ctlr_flts.dtype, ctlr_flts.ids for filter content.
ctlr_flts.channel	number	0	Channel to receive data from the server. Used when "type" is "control".
ctlr_flts.dtype	number	0	iSCP data type code of data to be received from the server. Used when "type" is "control".
ctlr_flts.ids	number[] or string[] (Data IDs for CAN and Generic are numbers, so use number[]; data IDs for String, Float, Int, and Bytes are strings, so use string[].)	[]	Data ID of the data to be received from the server (optional). Used when "type" is "control". In the case of CAN data, if you set the value to an empty array [], data of any ID will be received. For data types other than CAN, setting [] will cause nothing to be received.
unit_flush_cycle	number	5	Flush interval [msec]. Used when "type" is "realtime".
resend_cycle	number	1000	Retransmission cycle [msec]. Used when "type" is "resend".

continues on next page

Table 2 – continued from previous page

Key	Type	Default value	Description
store_flushtime	number	3000	Flush interval [msec]. Used when "type" is "bulk".
store_flushsize	number	10000	Flush interval [number of units]. Used when "type" is "bulk".
store_cycle	number	1000	Transmission interval [msec]. Used when "type" is "bulk".
http_client_count	number	1	Number of simultaneous retransmissions. Used when the protocol is "mod_http".
connection.host	string	""	The hostname + domain name (FQDN) of the intdash server used by the edge (e.g., dummy.intdash.jp). Please note that depending on your environment, the server name used by the edge and the server name used by the web applications may be different. In the case of intdash environments operated by aptpod, the server name for edges is usually <xxxxx>.intdash.jp and the server name for web applications is <xxxxx>.vm2m.jp (the <xxxxx> part is the same).
connection.path	string	"/"	Path to the server resource. (If the protocol is mod_websocket*, use /api/v1/ws/measurements. If the protocol is mod_http, use /api/v1/measurements.)
connection.ssl	string	"secure"	Security settings for SSL connections (none|lax|secure).
connection.port	number	443	The port of the intdash server to which the edge connects.
connection.cert	string	"" (Use a certificate installed in the OS.)	Server certificate file path (optional).
connection.client_cert	string	""	Certificate when using a client certificate (optional).
connection.client_key	string	""	Private key when using a client certificate (optional).

continues on next page

Table 2 – continued from previous page

Key	Type	Default value	Description
user_agent	string	"IntDash-Edge/ unknown (Unknown; Unknown)"	User agent (optional).

6.6.4 Device connector settings

The settings for Device Connectors are in the loggers field of the configuration file as an array of objects. The items are as follows.

Key	Type	Default value	Description
path	string	""	Full path of the Device Connector you want to start automatically.
conf	string	""	String to be passed as the second argument when the Device Connector is started automatically. The first argument is -c.
details.plugin	string	""	Name of the plugin to use (fifo|status).
details.plugin_dir	string	"/opt/vm2m/lib/ plugins"	Directory where the executable file of the plug-in is stored.
details.plugin_arg	string	{}	JSON object for plugin settings. Settings differ depending on the plug-in.
details.plugin_with_process	bool	true	Whether to start the Device Connector automatically when using the plug-in.
connections[].channel	number	-1	Channel (0-255) to set for the Device Connector.
connections[].channel_rx	number	-1 (If set to -1, the same value as the "channel" is used)	Channel used to send data to the Device Connector when using downstream
connections[]. receive_basetime	bool	false	Whether base time data is sent to the Device Connector when using downstream.
connections[].fifo_tx	string	"/opt/vm2m/var/ run/intdash/ logger_%03hhu.tx"	FIFO file path used for communication. The first format specifier (%03hhu in the default value) is replaced with the channel number.
connections[].fifo_rx	string	"/opt/vm2m/var/ run/intdash/ logger_%03hhu.rx"	FIFO file path used for communication. The first format specifier (% 03hhu in the default value) is replaced with the channel number,

continues on next page

Table 3 – continued from previous page

Key	Type	Default value	Description
connections[].disable_raw	number	0	Whether to save the data of this channel as RAW data (0: save, non-zero: do not save).

6.7 Agent logs

The Agent outputs log messages to the standard output.

By redirecting the standard output, log messages can be saved to a file. For example, if you start the Agent as follows, log messages will be output to /var/run/intdash/intdash.log (${CONF_PATH} is the full path of the configuration file.

```
/opt/vm2m/sbin/intdash-edge-manager -C ${CONF_PATH} >/var/run/intdash/intdash.log 2>&1
```

6.7.1 Log message format

Log messages are output in a format similar to the following example.

```
01/13 01:51:49 intdash-edge-manager(1532): INFO  : procedure(): CREATED
01/13 01:51:49 intdash-edge-manager(1532): INFO  : start(): Manager thread STARTING
01/13 01:51:49 intdash-edge-manager(1532): INFO  : procedure(): STARTED
01/13 01:51:49 intdash-edge-manager(1532): INFO  : base_proc(): Manager thread STARTED
01/13 01:51:49 intdash-edge-manager(1532): INFO  : proc(): Basetime (RTC monotonic) : 58469.155277440
01/13 01:51:49 intdash-edge-manager(1532): INFO  : proc(): Basetime (RTC realtime) : 1610502709.225218700
01/13 01:51:49 intdash-edge-manager(1532): INFO  : start(): RawDataHandler thread STARTING
01/13 01:51:49 intdash-edge-manager(1532): INFO  : start(): DataSaver thread STARTING
 ^              ^                           ^       ^
(1)            (2)                         (3)     (4)
```

Number	Description
(1)	Date and time of occurrence
(2)	Module name (process number)
(3)	Prefix (INFO: information about normal operation, WARN: information about recoverable errors, ERR: information about non-recoverable errors)
(4)	Function name and log content

6.7.2 Log messages about the Agent application

The main log messages related to the Agent application are as follows.

Creating a FIFO to connect to the device connector

```
INFO  : procDataReader(): DeviceProcess create pipe : /opt/vm2m/var/run/intdash/logger_000.tx
```

The path in the log message is the path of the FIFO created by intdash-edge-manager.

Opening FIFO

```
INFO  : procDataReader(): DeviceProcess open pipe : /opt/vm2m/var/run/intdash/logger_000.tx
```

The path in the log message is the path of the FIFO opened by intdash-edge-manager.

Closing FIFO

```
INFO  : procDataReader(): close pipe:/opt/vm2m/var/run/intdash/logger_004.tx
```

The path in the log message is the path of the FIFO that was closed by intdash-edge-manager.

6.7.3 Log messages about real-time transmission via WebSocket

The following are the main log messages related to real-time transmission via WebSocket.

Establishing a connection with the server

```
INFO  : callbackPacketEstablish(): REALTIME Upstream request succeed
```

Sending data of one section

This message indicates that the data of one section has been sent. At this point, processing on the server side has not been completed.

```
INFO  : callbackSndSectionEnd(): REALTIME 46 units and 1 basetimes with ID:2
                                          ^          ^                 ^
                                         (1)        (2)               (3)
```

Number	Description
(1)	Number of units in the section (excluding base time)
(2)	Number of base time units included in the section
(3)	Serial number given to the section

Completing transmission of a single section

This message indicates that the processing of one section has been completed on the server side. When an ACK is received from the server, this message is output.

```
INFO  : callbackRcvPacket(): REALTIME(c4a0e287) ACK 2 (45/45 units)
                                                ^           ^  ^  ^
                                               (1)         (2)(3)(4)
```

Number	Description
(1)	Measurement ID
(2)	Serial number given to the section
(3)	Number of units in this section
(4)	Total number of units sent in this measurement

Saving data of one section as a file for retransmission

```
INFO  : saveSection(): Store   2 units to /opt/vm2m/var/lib/intdash/meas/<SERVER_NAME>/<MEASUREMENT>/<SECTION>
↪.bin
                              ^          ^
                             (1)        (2)
```

Number	Description
(1)	Number of units in the section
(2)	Path of the retransmission section file

Closing the connection to the server

```
INFO  : callbackPacketEstablish(): REALTIME Upstream closed
```

6.7.4 Log messages about retransmissions via HTTP

The main log messages related to retransmissions via HTTP are as follows

Completing retransmission of one section

```
INFO  : postSection(): RESEND 32187 units and ? basetimes with ID:80891
                              ^                                  ^
                             (1)                                (2)
```

Number	Description
(1)	Number of units in the section
(2)	Serial number given to the section