intdash Edge Agent デベロッパーガイド

intdash Edge Agent Version 1.17.2

第 2 版（2021 年 3 月）

aptpod

3687f72

目次

01 はじめに

本ドキュメントでは、intdash Edge Agent（以降、Agent と呼びます）の使用方法を解説します。

> **注釈:** 本マニュアルに記載されている会社名、サービス名、製品名等は、一般に、各社の登録商標または商標です。本文および図表中には、「™」、「®」は明記していません。

1.1 Agent とは

Agent とは、intdash サーバーとの間でデータの送受信を行うエージェントソフトウェアです。

Agent は、高頻度に発生する時系列データを、低遅延で intdash サーバーへストリーミング送信することができます。ネットワーク回線の切断などの障害により送信できなかったデータは、自動的に再送信します。これにより、データは intdash サーバーに完全に回収されます。

1.2 主要機能

- 時系列データのストリーミング
- 伝送欠損データの自動的な再送信
- 時系列データのフィルタリングとサンプリング
- 取得した時系列データをダンプファイルとして保存

1.3 動作要件

- 対応プラットフォーム
  - AMD64 アーキテクチャー上の Linux
  - Raspberry Pi 上の Raspbian
  - NVIDIA Jetson 上の NVIDIA L4T
- 最低ハードウェア要件
  - Intel Atom プロセッサー E3815、1.46GHz 相当以上
- 推奨ハードウェア要件
  - マルチコア CPU
  - SSD

重要: データ量が多く、CPU の負荷が高くなりすぎると、データが欠損することがあります。
欠損なく処理可能なデータ量の目安は以下の通りです。(以下の値は、処理負荷の小さなデバイスコネクターを使った場合のおおよその目安です。デバイスコネクターの処理の負荷によって限界性能は増減します。)

- **VTC 1910-S (Intel Atom E3815 1.46GHz) 使用時**
  - 小さなデータを高頻度で送信するケース: 8B (バイト) のユニットを、24000 ユニット／秒程度
  - 大きなデータを低頻度で送信するケース: 0.98MB のユニットを、10 ユニット／秒程度
- **Raspberry Pi 4 Model B 使用時**
  - 小さなデータを高頻度で送信するケース: 8B (バイト) のユニットを、100000 ユニット／秒程度
  - 大きなデータを低頻度で送信するケース: 0.98MB のユニットを、80 ユニット／秒程度

CPU に高い負荷がかかることが予想される場合は、事前にテスト計測を行うことをお勧めします。テスト計測を行ったあと、エッジ上で以下のコマンドを実行して、syslog 内のメッセージを確認してください。

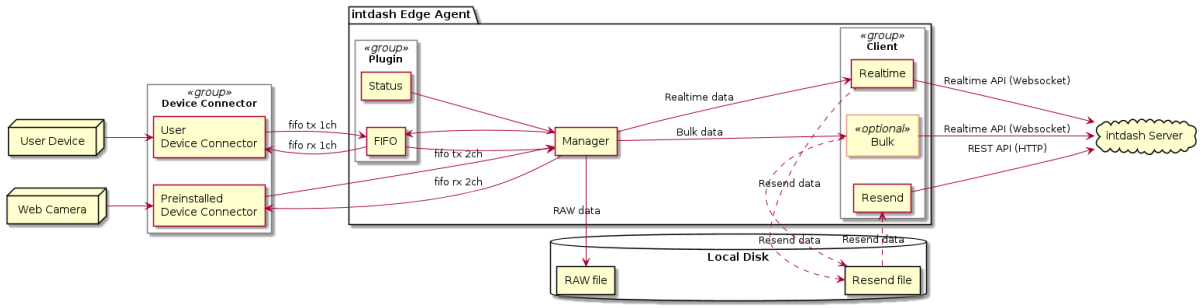
```
$ cat /var/log/syslog | grep -e"ring is full" -e "data buffer is overflow" | grep -v PacketSender
```

syslog の中に、`ring is full` または `data buffer is overflow` というメッセージがある場合 (ただし PacketSender に関するメッセージは除外)、デバイスコネクターとエージェントの間でデータの取りこぼしが発生しています。

1.4 システム構成

Agent は、デバイスコネクター、マネージャー、クライアントなどのソフトウェアモジュールから構成されます。

- デバイスコネクターは、デバイスからデータを受け取ります。
- マネージャーは、フィルタリングやサンプリングなどの各種処理を行います。
- クライアントは、時系列データを intdash サーバーに送信します。



1.4.1 デバイスコネクター

デバイスコネクターは、各種デバイスと Agent を接続するためのソフトウェアです。

デフォルトのデバイスコネクター `intdash-edge-logger` と、これを使用するための 6 種の設定が Agent と同時にインストールされます。

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

デバイスコネクターを使用するためには、設定が必要です。上記のデバイスコネクターを使用するための設定方法については、プリインストールされたデバイスコネクターを使用するための設定 (p. 13) を参照してください。

ユーザーは、新たにデバイスコネクターを用意することで、`intdash-edge-logger` が対応していないデバイスも接続することができます。`intdash-edge-logger` 以外のデバイスコネクターを追加する方法については、独自のデバイスコネクターを追加する (p. 34) を参照してください。

> **注釈:** 設定ファイル内で「logger」という言葉が使用されることがありますが、「logger」はデバイスコネクターを指します。

1.4.2 プラグイン

プラグインは、デバイスコネクターと Agent のインターフェイスとなる内部ソフトウェアモジュールです。以下の 2 種類があります。

FIFO プラグイン
: FIFO（名前付きパイプ）経由で、Agent とデバイスコネクターとの間でデータを送受信します。

Status プラグイン
: デバイスコネクターとは接続せず、ステータス情報を収集します。

> **注釈:**
> - プラグインは設定ファイルの `loggers[].details.plugin` で指定します。詳細については デバイスコネクターに関する設定 (p. 62) を参照してください。
> - アプトポッドが提供するデバイスコネクター `intdash-edge-logger` は、FIFO プラグインを使用しません。そのため、`intdash-edge-logger` を使う場合は、設定ファイルに `plugin` の記載は不要です。

1.4.3 マネージャー

マネージャーは、Agent の各種処理の中枢を担うソフトウェアモジュールです。他のモジュールの起動と停止の管理や、デバイスコネクターから集めたデータの集約、フィルタリングやサンプリングなどの処理、RAW データの書き出しなどを行います。

1.4.4 クライアント

Agent におけるクライアントは、Agent と intdash サーバーとの間の通信を担うソフトウェアモジュールです。intdash の Realtime API を使用したリアルタイム送信や、REST API を利用した高効率形式での送信、伝送欠損の再送などの機能を担います。

Realtime クライアント
: intdash の Realtime API（WebSocket）を使用してリアルタイムにデータを送信します。

Bulk クライアント
: intdash の Realtime API（WebSocket）を使用して数秒単位の一定周期でデータを一括送信します。

Resend クライアント
: Realtime クライアントや Bulk クライアントが送信できなかったデータを、一定周期で再送します。intdash サーバーとの通信には、Realtime API（WebSocket）、REST API どちらも使用することができます。

Control クライアント
: intdash の Realtime API（WebSocket）を使用してリアルタイムにデータを受信します。

02 使用を開始する

Agent のインストール方法と、起動や停止の手順について解説します。

インストーラーは以下の環境に対応しています。

ディストリビューション	バージョン	アーキテクチャー
Ubuntu	18.04 (LTS), 16.04 (LTS)	amd64, armhf, arm64
Debian	10	amd64
Raspbian	based on Debian Buster	armhf

2.1 インストール

インストール先の環境で、以下のようにインストーラーを実行してください。

> `${DISTRIBUTION}` と `${ARCHITECTURE}` には、それぞれ下の表からいずれかを選択してください。

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 インストールされるファイル

デフォルト設定でインストールすると、以下のファイルとフォルダーが配置されます。

```
+ /etc
  + opt
    + intdash                           # (1)
      - manager.conf                    # (2)
      - logger.conf.canopen             # (3)
      - logger.conf.nmea                # (4)
      - logger.conf.gstreamer_h264      # (5)
      - logger.conf.mjpeg               # (6)
      - logger.conf.socketcan           # (7)
      - logger.conf.v4lh264             # (8)

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

番号	ファイルまたはディレクトリの説明
(1)	設定ファイルを格納するディレクトリ
(2)	Agent のサンプル設定ファイル
(3)	canopen タイプのデバイスコネクター用のサンプル設定ファイル
(4)	nmea タイプのデバイスコネクター用のサンプル設定ファイル
(5)	gstreamer_h264 タイプのデバイスコネクター用のサンプル設定ファイル
(6)	mjpeg タイプのデバイスコネクター用のサンプル設定ファイル
(7)	socketcan タイプのデバイスコネクター用のサンプル設定ファイル
(8)	v4lh264 タイプのデバイスコネクター用のサンプル設定ファイル
(9)	ツールを格納するディレクトリ
(10)	Agent（または intdash-edge-logger）が使用するライブラリを格納するディレクトリ
(11)	Agent の実行ファイルとアプトポッド製デバイスコネクターを格納するディレクトリ
(12)	Agent ネットワークでのデータ送信を行うクライアントの実行ファイル
(13)	アプトポッド製デバイスコネクター
(14)	Agent のコアであるマネージャーの実行ファイル
(15)	Agent が使用するオープンソースライブラリの情報を格納するディレクトリ

2.3 Agent の起動と終了

2.3.1 Agent の起動

Agent を起動するには以下のコマンドを実行します。

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

full-path-to-the-configuration-file には、プリインストールされた設定ファイル /etc/opt/intdash/manager.conf を使用することができます。プリインストールされた設定ファイルは環境変数から設定を読み取るため、環境変数を設定する必要があります。

変数名	説明
INTDASH_EDGE_UUID	このエッジを識別するための UUID です。intdash サーバーから発行されたものを使用してください。（例：f90f2b42-66a5-4a57-8e99-468c36ebb6f2 ）
INTDASH_EDGE_TOKEN	認証用のトークンです。intdash サーバーからこのエッジ用に発行されたものを使用してください。（例：sEh9ZHPoKX8QYU-v0Noe0ZPzxGBF.. .....iBn5fn_eFM ）
INTDASH_EDGE_SERVER	intdash サーバーの FQDN を指定してください。（例：dummy.intdash. jp ）
INTDASH_EDGE_APPDIR	アプリケーションデータ保存先（例：/var/lib ）
INTDASH_EDGE_RUNDIR	一時ファイル保存先（例：/var/run ）
INTDASH_EDGE_BINDIR	スクリプト保存先（例：/opt/vm2m/bin ）
INTDASH_EDGE_SBINDIR	実行ファイル保存先（例：/opt/vm2m/sbin ）
INTDASH_EDGE_LIBDIR	ライブラリ保存先（例：/opt/vm2m/lib ）
INTDASH_EDGE_CONFDIR	設定ファイル保存先（例：/etc/opt/intdash ）

```
$ sudo \
   LD_LIBRARY_PATH=/opt/vm2m/lib \
   INTDASH_EDGE_UUID=f90f2b42-66a5-4a57-8e99-468c36ebb6f2 \
   INTDASH_EDGE_TOKEN=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
```

> **注釈:** `INTDASH_EDGE_UUID` および `INTDASH_EDGE_TOKEN` 、`INTDASH_EDGE_SERVER` は必ず使用する環境の値に書き換えてください。

プリインストールされた設定ファイル `/etc/opt/intdash/manager.conf` では、デバイスコネクターは設定されておらず、Status プラグインのみが動作します。Status プラグインは、以下のようにシステム情報、ネットワーク情報、Agent のステータスを取得し、intdash サーバーへ送信します。

データタイプ	データ ID	チャンネル	内容
String	s00	255	システム情報
String	s20	255	ネットワーク情報
String	s50	255	Agent のステータス

2.3.2 Agent の終了

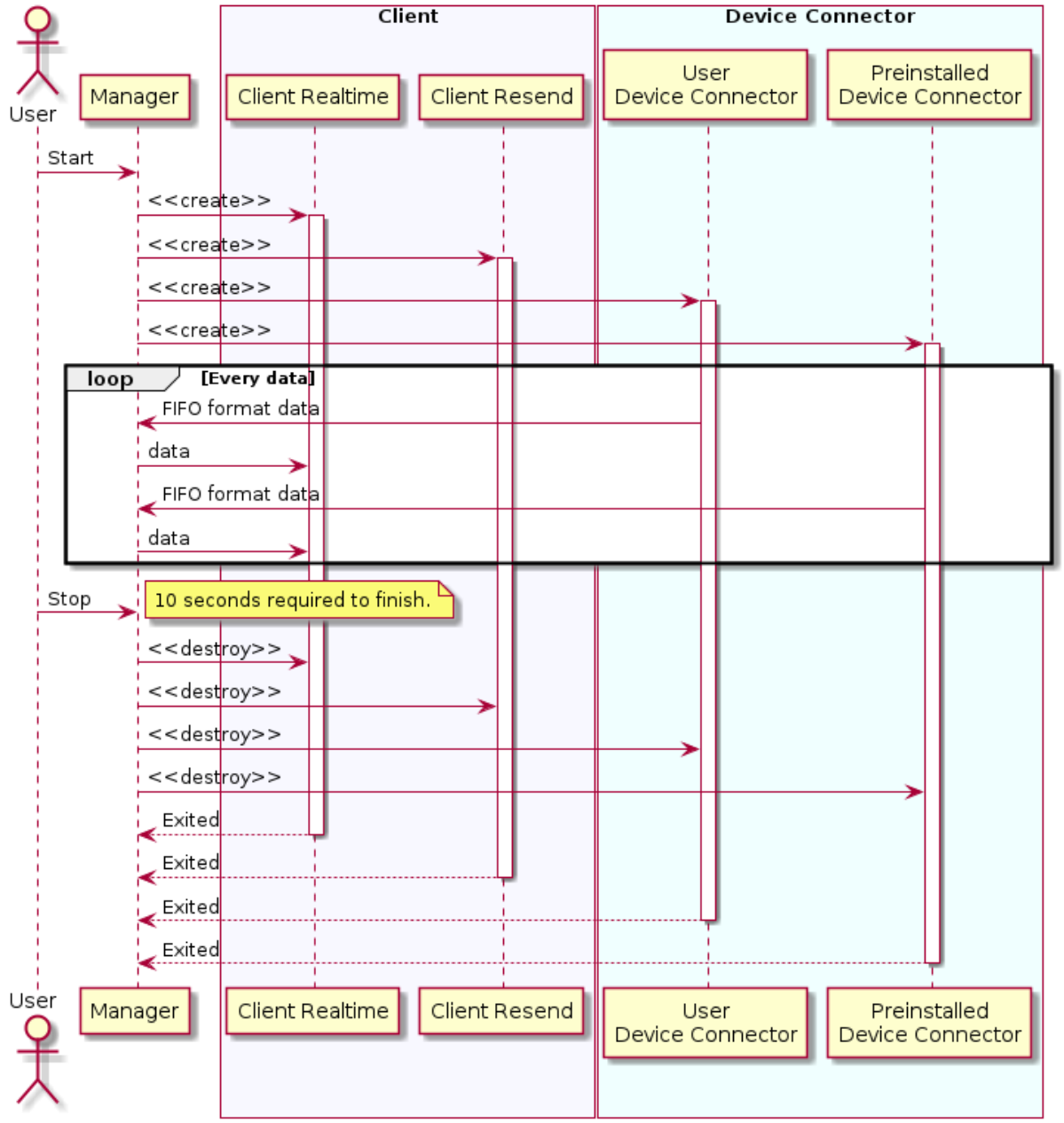
Agent を終了するには以下のいずれかを行います。

- SIGINT を送信する
- 以下のコマンドを実行する:

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

> **注意:** Agent の終了処理には 10 秒程度の時間がかかる場合があります。

2.3.3 参考：Agent の起動から終了までのシーケンス



03 設定を変更する

Agent は、デバイスコネクターから受け取ったデータを intdash サーバーに送信します。その際、データをフィルタリングしたり、ネットワークの状態に応じてデータを再送したりできます。

基本の設定は、マネージャーの設定ファイル（例：`manager.conf` ）で行います。

デバイスコネクターを使用するには、1 つのデバイスコネクターにつき 1 つのデバイスコネクター用設定ファイルが必要です。使用するデバイスコネクターの設定ファイルは、マネージャーの設定ファイル内で指定します。

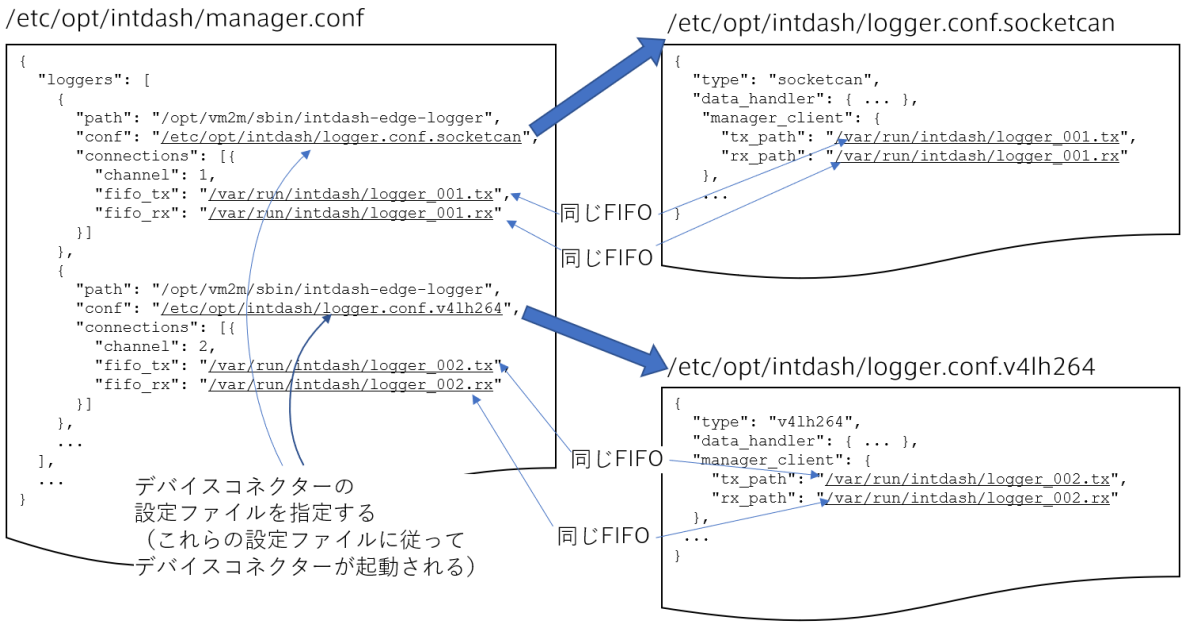


図 1　マネージャーの設定ファイルとデバイスコネクターの設定ファイルの例

設定ファイルで指定できる項目の一覧については、Agent の全設定 (p. 57) を参照してください。ここでは、代表的な設定例を紹介します。

> **注釈:** 設定ファイル内の「logger」はデバイスコネクターを指します。

3.1 エッジ情報の設定

この Agent が intdash に接続する際に使用するエッジアカウントを設定します。設定は、intdash サーバーとの通信を担うクライアントごとに行います。以下の例のように、各クライアントの設定で、使用するエッジアカウントを UUID で指定し、そのトークンを指定してください。

> **注釈:** 基本的に、1 つの Agent のクライアントには同じエッジアカウントを使用してください。以下の例では、Realtime クライアントと Resend クライアントに同じエッジアカウントを設定しています。

エッジ情報の設定例

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

番号	フィールド	説明
(1)	my_id	このクライアントに割り当てるエッジ UUID。
(2)	my_token	認証用のエッジトークン。

3.2 プリインストールされたデバイスコネクターを使用するための設定

プリインストールされたデバイスコネクター intdash-edge-logger を使用するには設定が必要です。

デバイスコネクターの設定は、マネージャーの設定ファイル（manager.conf）と、デバイスコネクターの設定ファイルの両方で行います。

起動するデバイスコネクターごとに設定ファイルを用意し、マネージャーの設定ファイルでも複数のデバイスコネクターを設定することで、intdash-edge-logger を複数のプロセスとしてそれぞれ起動し、複数のデバイスからデータを収集することができます。

3.2.1 マネージャーの設定ファイル例（manager.conf）

マネージャーの設定ファイルでは、loggers に、使用するデバイスコネクターの設定を書き込みます。

```
{
  "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"
      }]
    },
    ...
  ],
  ...
}
```

番号	フィールド	説明
(1)	-	1 つのデバイスコネクターとの接続を、1 つの JSON オブジェクトで表現します。
(2)	`path`	プリインストールされたデバイスコネクターのフルパス。
(3)	`conf`	起動するデバイスコネクターの設定ファイル。ここでは例として Motion JPEG 用の設定 `logger.conf.mjpeg` を指定しています。
(4)	`channel`	このデバイスコネクターに設定するチャンネル（0-255）。デバイスコネクターから取得されたデータには、チャンネル番号が付加されます。チャンネル番号は他のデバイスコネクターと重複しないようにしてください。
(5)	`fifo_tx`	このデバイスコネクターが送信に使用する FIFO のパス。パスは他のデバイスコネクターと重複しないようにしてください。
(6)	`fifo_rx`	このデバイスコネクターが受信に使用する FIFO のパス。パスは他のデバイスコネクターと重複しないようにしてください。
(7)	-	2 つ以上のデバイスコネクターを使用する場合は、ここから 2 つ目のデバイスコネクターの設定を行います。以下同様に設定します。

3.2.2 デバイスコネクターの設定ファイル例

mjpeg タイプのデバイスコネクター設定例

mjpeg タイプのデバイスコネクターは、Video4Linux に対応した UVC（USB Video Class）カメラから Motion JPEG データを取得します。

設定ファイル `/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)
}
```

番号	フィールド	説明
(1)	type	デバイスコネクターのタイプ。mjpeg タイプの場合は"mjpeg"とします。
(2)	path	デバイスパス
(3)	baudrate	フレームレート（1、5、10、15、30 のいずれか）[fps]
(4)	camera_width	画面幅（320 または 640）
(5)	camera_height	画面高さ（240 または 480）
(6)	camera_hwencodedelay_msec	データ打刻のオフセット（カメラ処理時間）[msec]。例えば、100 と設定すると、カメラ内での処理に 100 ミリ秒かかったもの想定して、100 ミリ秒前のタイムスタンプを付与します。
(7)	tx_path	デバイスコネクターが送信に使用する FIFO のパス。manager.conf でこのデバイスコネクター用に設定した fifo_tx と同じパスを設定してください。
(8)	rx_path	デバイスコネクターが受信に使用する FIFO のパス。manager.conf でこのデバイスコネクター用に設定した fifo_rx と同じパスを設定してください。
(9)	basetime	デバイスコネクターが時刻管理に使用するファイルのパス。manager.conf で設定した manager.basetime と同じ値を設定してください。
(10)	status	デバイスコネクターが状態を書き出すファイルのパス。

v4lh264 タイプのデバイスコネクター設定例

v4lh264 タイプのデバイスコネクターは、Video4Linux に対応した UVC（USB Video Class）カメラから H.264 データを取得します。

設定ファイル `/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)
}
```

番号	フィールド	説明
(1)	type	デバイスコネクターのタイプ。v4lh264 タイプの場合は"v4lh264" とします。
(2)	path	デバイスパス
(3)	baudrate	フレームレート（5、10、15、30）[fps]
(4)	camera_width	画面幅（1080、1920）
(5)	camera_height	画面高さ（720、1020）
(6)	camera_hwencodedelay_msec	データ打刻のオフセット（カメラ処理時間）[msec]。例えば、100 と設定すると、カメラ内での処理に 100 ミリ秒かかったもの想定して、100 ミリ秒前のタイムスタンプを付与します。
(7)	tx_path	デバイスコネクターが送信に使用する FIFO のパス。manager.conf でこのデバイスコネクター用に設定した fifo_tx と同じパスを設定してください。
(8)	rx_path	デバイスコネクターが受信に使用する FIFO のパス。manager.conf でこのデバイスコネクター用に設定した fifo_rx と同じパスを設定してください。
(9)	basetime	デバイスコネクターが時刻管理に使用するファイルのパス。manager.conf で設定した manager.basetime と同じ値を設定してください。
(10)	status	プリインストールされたデバイスコネクターが状態を書き出すファイルのパス。

gstreamer_h264 タイプのデバイスコネクター設定例

gstreamer_h264 タイプのデバイスコネクターは、GStreamer から H.264 動画を取得します。

設定ファイル /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)
 },
```

(次のページに続く)

(前のページからの続き)

```
 "basetime": "/var/run/intdash/basetime",         # (11)
 "status": "/var/run/intdash/logger_XXX.stat"     # (12)
}
```

番号	フィールド	説明
(1)	type	デバイスコネクターのタイプ。gstreamer_h264 タイプの場合は、"gstreamer_h264"とします。
(2)	path	デバイスパス
(3)	baudrate	フレームレート（1, 5, 10, 15, 30）[fps]
(4)	camera_width	画面幅（1080, 1920）
(5)	camera_height	画面高さ（720, 1020）
(6)	camera_keyperiod	キーフレーム間隔（フレームレート ×10 を設定する）
(7)	camera_hwencodedelay_msec	データ打刻のオフセット（カメラ処理時間）[msec]。例えば、100 と設定すると、カメラ内での処理に 100 ミリ秒かかったもの想定して、100 ミリ秒前のタイムスタンプを付与します。
(8)	command	GStreamer のコマンド。使用するカメラおよび実行環境に適したパイプラインを設定し、標準出力から H.264 のデータが出力されるようにしてください。
(9)	tx_path	デバイスコネクターが送信に使用する FIFO のパス。manager.conf でこのデバイスコネクター用に設定した fifo_tx と同じパスを設定してください。
(10)	rx_path	デバイスコネクターが受信に使用する FIFO のパス。manager.conf でこのデバイスコネクター用に設定した fifo_rx と同じパスを設定してください。
(11)	basetime	デバイスコネクターが時刻管理に使用するファイルのパス。manager.conf で設定した manager.basetime と同じ値を設定してください。
(12)	status	プリインストールされたデバイスコネクターが状態を書き出すファイルのパス。

command の設定値では、以下の変数を使用することにより他の設定値を参照することができます。

$_PATH

path に設定した値

$_FPS

baudrate に設定した値

$_WIDTH

camera_width に設定した値

$_HEIGHT

camera_height に設定した値

$_KEYPERIOD

`camera_keyperiod` に設定した値

nmea タイプのデバイスコネクター設定例

nmea タイプのデバイスコネクターは、GPS デバイスから NMEA データを取得します

設定ファイル `/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)
}
```

番号	フィールド	説明
(1)	`type`	デバイスコネクターのタイプ。nmea タイプの場合は"nmea"とします。
(2)	`path`	デバイスパス
(3)	`baudrate`	GPS モジュールとの通信ボーレート [bps]
(4)	`tx_path`	デバイスコネクターが送信に使用する FIFO のパス。manager.conf でこのデバイスコネクター用に設定した fifo_tx と同じパスを設定してください。
(5)	`rx_path`	デバイスコネクターが受信に使用する FIFO のパス。manager.conf でこのデバイスコネクター用に設定した fifo_rx と同じパスを設定してください。
(6)	`basetime`	デバイスコネクターが時刻管理に使用するファイルのパス。manager.conf で設定した manager.basetime と同じ値を設定してください。
(7)	`status`	プリインストールされたデバイスコネクターが状態を書き出すファイルのパス。

socketcan タイプのデバイスコネクターの設定例

socketcan タイプのデバイスコネクターは、オープンソースの SocketCAN ドライバーから CAN データを取得します。

設定ファイル `/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)
}
```

番号	フィールド	説明
(1)	`type`	デバイスコネクターのタイプ。socketcan タイプの場合は"socketcan"とします。
(2)	`path`	インターフェイス名
(3)	`baudrate`	CAN バスのボーレート（125、250、500、1000）[Kbps]
(4)	`listenonly`	(int) 0:ACK を返す、0 以外:ACK を返さない
(5)	`tx_path`	デバイスコネクターが送信に使用する FIFO のパス。manager.conf でこのデバイスコネクター用に設定した fifo_tx と同じパスを設定してください。
(6)	`rx_path`	デバイスコネクターが受信に使用する FIFO のパス。manager.conf でこのデバイスコネクター用に設定した fifo_rx と同じパスを設定してください。
(7)	`basetime`	デバイスコネクターが時刻管理に使用するファイルのパス。manager.conf で設定した manager.basetime と同じ値を設定してください。
(8)	`status`	プリインストールされたデバイスコネクターが状態を書き出すファイルのパス。

canopen タイプのデバイスコネクター設定例

canopen タイプのデバイスコネクターは、オープンソースの SocketCAN ドライバーから CANOpen データを取得します。

設定ファイル `/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)
  },
  "basetime": "/var/run/intdash/basetime",        # (7)
  "status": "/var/run/intdash/logger_XXX.stat"    # (8)
}
```

番号	フィールド	説明
(1)	`type`	デバイスコネクターのタイプ。canopen タイプの場合は"canopen"とします。
(2)	`path`	インターフェイス名
(3)	`baudrate`	CAN バスのボーレート（125、250、500、1000）[Kbps]
(4)	`listenonly`	(int) 0:ACK を返す、0 以外:ACK を返さない
(5)	`tx_path`	デバイスコネクターが送信に使用する FIFO のパス。manager.conf でこのデバイスコネクター用に設定した fifo_tx と同じパスを設定してください。
(6)	`rx_path`	デバイスコネクターが受信に使用する FIFO のパス。manager.conf でこのデバイスコネクター用に設定した fifo_rx と同じパスを設定してください。
(7)	`basetime`	デバイスコネクターが時刻管理に使用するファイルのパス。manager.conf で設定した manager.basetime と同じ値を設定してください。
(8)	`status`	プリインストールされたデバイスコネクターが状態を書き出すファイルのパス。

3.3 データ送受信に関する設定

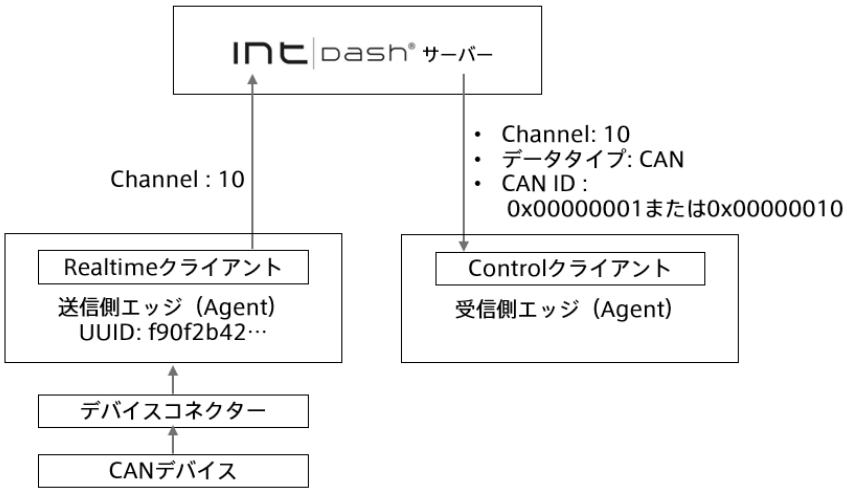
Agent は、intdash サーバーから受信したデータをデバイスコネクターに送ることもできます。

以下に、 2 つの Agent を使用してデータを送受信する設定の例を挙げます。

3.3.1 CAN データを送受信するための設定例（宛先指定なし）

以下の例は、送信側の Agent が intdash サーバーを介して受信側の Agent に CAN データを送信するための設定です。

この設定例では、送信側の Agent がチャンネル 10 で送信した CAN データを受信側の Agent が受信します。



CAN データ送信側エッジの設定例（宛先指定なし）

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

番号	フィールド	説明
(1)	`my_id`	送信側エッジ（このエッジ）の UUID
(2)	`my_token`	送信側エッジ（このエッジ）の認証用トークン
(3)	`type`	リアルタイム送信を行うため Realtime クライアント `realtime` が指定されています。
(4)	`loggers`	CAN データを取得するデバイスコネクターを設定します。
(5)	`channel`	送信するデータのチャンネル。

CAN データ受信側エッジの設定例

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

番号	フィールド	説明
(1)	`my_id`	ここで指定した UUID 宛てのデータを受信します。0000... を指定した場合、すべの宛先のデータを受信します。
(2)	`my_token`	受信側エッジ（このエッジ）の認証用トークン
(3)	`ctlr_id`	受信するデータの、送信元エッジの UUID（指定したエッジから送信されたデータのみを受信します。）
(4)	`ctlr_flts`	受信するデータを指定するフィルター。チャンネル、データタイプ、ID の組み合わせで指定を行います。複数のフィルターを設定可能（許可リスト）。(intdash サーバーから受信するデータをフィルタリングすることができます。受信データのフィルタリングについては、iSCPv1 に関する資料を参照してください。)
(5)	`channel`	受信するデータのチャンネル (この例ではチャンネル 10 のデータのみを受信します。)
(6)	`dtype`	受信するデータのタイプ。数値がタイプを表します（1 は CAN を表します。CAN データのみを受信します。)
(7)	`ids`	受信するデータの CAN ID（CAN ID が `0x00000001` または `0x00000010` のデータのみを受信することを表します。(配列を空にすることで、全データ ID を受信対象とすることも可能です。)
(8)	`type`	受信を行うため Control クライアント `control` とします。
(9)	`channel`	デバイスコネクターが受信するデータのチャンネル。この例ではチャンネル 10 とします。

重要: dtype では、iSCP のデータタイプコードを十進数で指定します。iSCP のデータタイプコードは以下のとおりです。

データタイプコード (十進数)	データタイプ名
0x01 (1)	CAN
0x02 (2)	NMEA
0x03 (3)	General Sensor（汎用センサー）
0x04 (4)	Controlpad（汎用コントロールパッド）
0x05 (5)	MAVLink 2 Packet（Micro Air Vehicle/ドローン用の通信プロトコル）
0x09 (9)	JPEG
0x0A (10)	String（文字列）
0x0B (11)	Float（倍精度浮動小数点数）
0x0C (12)	Int（64bit 符号付き整数）
0x0D (13)	H.264
0x0E (14)	Bytes（バイト列）
0x0F (15)	PCM（WAVE）
0x10 (16)	AAC（ADTS）
0x7F (127)	Generic（汎用バイナリデータ）

上記の iSCP のデータタイプコードは、Agent とデバイスコネクターの間で使われる FIFO のデータフォーマット (p. 50) のデータタイプコードとは異なりますのでご注意ください。

3.4 送信タイミングに関する設定（送信側でのフィルタリング）

送信側 Agent でフィルターを設定することにより、intdash サーバーにデータを送信する Realtime クライアントや Bulk クライアントにデータを振り分け、データの送信タイミングを調整することができます。フィルターの詳細については、送信側でのフィルタリング (p. 39) を参照してください。

- 間引いて低頻度にしたデータをリアルタイムに送信し、残りのデータは後で送信する場合の設定例 (p. 26)
- 一部のデータをリアルタイム送信し、その他のデータは後で送信する設定の例 (p. 28)
- 送信は行わずに全てのデータを RAW データとして保存する場合の設定例 (p. 30)
- 全てのデータを Resend クライアント用として蓄積する場合の設定例 (p. 31)

3.4.1 間引いて低頻度にしたデータをリアルタイムに送信し、残りのデータは後で送信する場合の設定例

ネットワーク帯域が狭い場合、サンプリングによりデータを間引いて一部のデータのみを Realtime クライアントで送信し、残りのデータは帯域が回復した際に Resend クライアントで送信することができます。

チャンネル 1 のデータを 1 秒間隔でサンプリングし、サンプリングしたデータを Realtime クライアントで送信し、残りのデータは Resend クライアントで送信する設定例

```
{
  "manager": {
    "filters": [
      {
        "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_token": "sEh9ZHPoKX8QYU-v0Noe0ZPzxGBF.........iBn5fn_eFM", # (7)
      "type": "realtime",                                            # (8)
      "protocol": "mod_websocket.v2",                                # (9)
      ...
    },
    {
      "my_id": "f90f2b42-66a5-4a57-8e99-468c36ebb6f2",               # (6)
      "my_token": "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_token": "sEh9ZHPoKX8QYU-v0Noe0ZPzxGBF.........iBn5fn_eFM", # (7)
      "type": "resend",                                              # (12)
      "protocol": "mod_http",                                        # (13)
      ...
    }
```

(次のページに続く)

(前のページからの続き)

```
  ],
  "loggers": [{
    "connections": [{
        "channel": 1,                                          # (14)
        ...
    }],
    ...
  }],
  ...
}
```

番号	フィールド	説明
(1)	name	フィルターの種類。sampling とします。
(2)	channel	フィルター対象のチャンネル。この例ではチャンネル 1 を設定しています。
(3)	target	フィルター対象の送信方法。realtime とします。
(4)	key	フィルターの詳細設定。サンプリング周期を設定するため rate とします。
(5)	value	フィルターの詳細設定。サンプリング周期を設定します。この例では 1000 ミリ秒（1 秒）とします。1 秒に 1 個のデータを送信します。
(6)	my_id	送信側エッジ（このエッジ）の UUID。
(7)	my_token	送信側エッジ（このエッジ）の認証用トークン。
(8)	type	送信クライアントの種類。realtime とします。
(9)	protocol	送信クライアントの通信方法。Realtime クライアントと Bulk クライアントは Realtime API を使用するため、mod_websocket.v2 とします。
(10)	type	送信クライアントの種類。bulk とします。
(11)	store_cycle	Bulk クライアントの送信間隔。Bulk クライアントからは送信せずに Resend クライアントに委ねるため、0 とします。
(12)	type	送信クライアントの種類。resend とします。
(13)	protocol	送信クライアントの通信方法。この例では Resend クライアントは REST API を使用するため、mod_http とします。
(14)	channel	取得するデータに付加するチャンネル。この例では、このデバイスコネクター（logger）がチャンネル 1 です。

3.4.2 一部のデータをリアルタイム送信し、その他のデータは後で送信する設定の例

ネットワーク帯域が狭い場合、小さいまたは頻度の少ないデータのみを Realtime クライアントで送信し、残りのデータは帯域が回復した際に Resend クライアントで送信することができます。

チャンネル 1 以外のデータを Realtime クライアントで送信し、チャンネル 1 のデータは Resend クライアントに委ねるための設定例

```
{
  "manager": {
    "filters": [
      {
        "name": "channel",                                          # (1)
        "channel": "1",                                             # (2)
        "target": "realtime",                                       # (3)
        "setting": []                                               # (4)
      }
    ],
    ...
  },
  "clients": [
    {
      "my_id": "f90f2b42-66a5-4a57-8e99-468c36ebb6f2",              # (5)
      "my_token": "sEh9ZHPoKX8QYU-v0Noe0ZPzxGBF.........iBn5fn_eFM", # (6)
      "type": "realtime",                                           # (7)
      "protocol": "mod_websocket.v2",                               # (8)
      ...
    },
    {
      "my_id": "f90f2b42-66a5-4a57-8e99-468c36ebb6f2",              # (5)
      "my_token": "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_token": "sEh9ZHPoKX8QYU-v0Noe0ZPzxGBF.........iBn5fn_eFM", # (6)
      "type": "resend",                                             # (11)
      "protocol": "mod_http",                                       # (12)
      ...
    }
  ],
  "loggers": [
    {
      "connections": [{
          "channel": 1,                                               # (13)
          ...
```

(次のページに続く)

(前のページからの続き)

```
      }],
      ...
    },
    {
      "connections": [{
          "channel": 2,                                          # (14)
          ...
      }],
      ...
    }
  ],
  ...
}
```

番号	フィールド	説明
(1)	name	フィルターの種類。channel とします。
(2)	channel	フィルター対象のチャンネル。この例ではチャンネル 1 とします。
(3)	target	フィルター対象の送信方法。realtime とします。
(4)	setting	フィルターの詳細設定。設定内容は空にします。
(5)	my_id	送信側エッジ（このエッジ）の UUID。
(6)	my_token	送信側エッジ（このエッジ）の認証用トークン。
(7)	type	送信クライアントの種類。realtime とします。
(8)	protocol	送信クライアントの通信方法。Realtime クライアントと Bulk クライアントは Realtime API を使用するため、mod_websocket.v2 とします。
(9)	type	送信クライアントの種類。bulk とします。
(10)	store_cycle	Bulk クライアントの送信間隔。Bulk クライアントは送信せずに Resend クライアントに委ねるため、0 とします。
(11)	type	送信クライアントの種類。resend とします。
(12)	protocol	送信クライアントの通信方法。この例では Resend クライアントは REST API を使用するため、mod_http とします。
(13)	channel	取得するデータに付加するチャンネル。この例では、1 つ目のデバイスコネクター（logger）がチャンネル 1 です。
(14)	channel	取得するデータに付加するチャンネル。この例では、2 つ目のデバイスコネクター（logger）がチャンネル 2 です。

3.4.3 送信は行わずに全てのデータを RAW データとして保存する場合の設定例

ネットワークの接続がまったく無いか、または帯域が狭すぎてデータを送信できない場合、intdash サーバーへのデータの送信はあきらめ、一旦すべてのデータをローカルストレージにダンプすることができます。ローカルストレージにダンプされたデータは、後で手動で intdash サーバーにアップロードする必要があります。

データをサーバーに送信しない設定の例

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

番号	フィールド	説明
(1)	clients	クライアントの設定。送信クライアントが設定されていない状態にします。
(2)	channel	取得するデータに付加するチャンネル。この例では、1 つ目のデバイスコネクター（logger）がチャンネル 1 です。
(3)	channel	取得するデータに付加するチャンネル。この例では、2 つ目のデバイスコネクター（logger）がチャンネル 2 です。

3.4.4 全てのデータを Resend クライアント用として蓄積する場合の設定例

ネットワークが不安定で、帯域に大きな変動がある場合、intdash サーバーへのリアルタイムでの転送はあきらめ、全て Resend クライアント用のデータとして蓄積しておき、帯域が回復した際に一気に送信することができます。この再送処理は自動的に行われるため、手動で intdash サーバーにアップロードする必要はありません。

全てのチャンネルのデータを、リアルタイム送信せずに Resend クライアントに委ねるための設定例

```
{
  "manager": {
    "filters": [
      {
        "name": "channel",                                          # (1)
        "channel": "-1",                                            # (2)
        "target": "realtime",                                       # (3)
        "setting": []                                               # (4)
      }
    ],
    ...
  },
  "clients": [
    {
      "my_id": "f90f2b42-66a5-4a57-8e99-468c36ebb6f2",              # (5)
      "my_token": "sEh9ZHPoKX8QYU-v0Noe0ZPzxGBF.........iBn5fn_eFM", # (6)
      "type": "realtime",                                           # (7)
      "protocol": "mod_websocket.v2",                               # (8)
      ...
    },
    {
      "my_id": "f90f2b42-66a5-4a57-8e99-468c36ebb6f2",              # (5)
      "my_token": "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_token": "sEh9ZHPoKX8QYU-v0Noe0ZPzxGBF.........iBn5fn_eFM", # (6)
      "type": "resend",                                             # (11)
      "protocol": "mod_http",                                       # (12)
      ...
    }
  ],
  "loggers": [
    {
      "connections": [{
          "channel": 1,                                               # (13)
```

(次のページに続く)

(前のページからの続き)

```
        ...
      }],
      ...
    },
    {
      "connections": [{
          "channel": 2,                                    # (14)
          ...
      }],
      ...
    }
  ],
  ...
}
```

番号	フィールド	説明
(1)	name	フィルターの種類。channel とします。
(2)	channel	フィルター対象のチャンネル。全てのチャンネルを対象とする-1 を設定します。
(3)	target	フィルター対象の送信方法。realtime とします。
(4)	setting	フィルターの詳細設定。設定内容は空にしてください。
(5)	my_id	送信側エッジ（このエッジ）の UUID。
(6)	my_token	送信側エッジ（このエッジ）の認証用トークン。
(7)	type	送信クライアントの種類。realtime とします。
(8)	protocol	送信クライアントの通信方法。Realtime クライアントと Bulk クライアントは Realtime API を使用するため、mod_websocket.v2 とします。
(9)	type	送信クライアントの種類。bulk とします。
(10)	store_cycle	Bulk クライアントの送信間隔。Bulk クライアントは送信せずに Resend クライアントに委ねるため、0 とします。
(11)	type	送信クライアントの種類。resend とします。
(12)	protocol	送信クライアントの通信方法。この例では Resend クライアントは REST API を使用するため、mod_http とします。
(13)	channel	取得するデータに付加するチャンネル。この例では、このデバイスコネクター（logger）がチャンネル 1 です。
(14)	channel	取得するデータに付加するチャンネル。この例では、このデバイスコネクター（logger）がチャンネル 2 です。

3.5 RAW データの保存に関する設定

RAW データの詳細については、RAW データ (p. 45) を参照してください。

RAW データの保存については、以下の設定ができます。

3.5.1 どのデータも RAW データとして保存しない

どのデータも RAW データとして保存しないように設定するには、`manager.rawdir` を null にします。

例（RAW データを保存しない）:

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

3.5.2 特定のチャンネルの RAW データを保存しない

対象のチャンネルのデバイスコネクターの設定に `loggers[].connections[].disable_raw` を追加し、値に `1` を設定します。

例（チャンネル 0 の RAW データ保存を無効にする）:

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

04 独自のデバイスコネクターを追加する

デバイスからデータを読み取って Agent に渡すデバイスコネクターを開発することで、任意のデバイスから取得したデータを intdash サーバーに送信することができます。デバイスコネクターと Agent の間のデータの受け渡しは FIFO で行います。

4.1 独自のデバイスコネクターを使用するための設定

Agent とデバイスコネクターは FIFO により接続されています。デバイスからのデータが Agent に送られてくるとき、デバイスコネクターは所定のフォーマットで FIFO にデータを書き込み、Agent がそれを読み出します。反対に、Agent からデバイスにデータを送るときには、Agent は所定のフォーマットで FIFO にデータを書き込み、デバイスコネクターがそれを読み出します。

FIFO はプラグインにより提供されます。Agent に FIFO プラグインを追加し、デバイスコネクターと接続するには、設定ファイルに以下の項目を追加します。FIFO プラグインが追加された状態で Agent を起動すると、デバイスコネクターとプラグイン間の通信用の FIFO ファイルが生成されます。

> **注釈:** ただし、アプトポッドが提供するデバイスコネクター `intdash-edge-logger` は、FIFO プラグインを使用しません。そのため、`intdash-edge-logger` を使う場合は、設定ファイルに `plugin` の記載は不要です。

例（チャンネル 2 の FIFO を追加する）：

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

> **注釈:** 設定ファイル内の「logger」はデバイスコネクターを指します。

この設定を使用して Agent を起動すると、Agent とデバイスコネクターの間の通信のために `/var/run/intdash/logger_002.tx` と `/var/run/intdash/logger_002.rx` という FIFO ファイルが作成されます。デバイスコネクターは、Agent にデータを送信する場合は `/var/run/intdash/logger_002.tx` に書き込み、Agent からデータを受信する場合は `/var/run/intdash/logger_002.rx` から読み込む必要があります。

注釈: Agent が intdash サーバーにデータを送信する際には、設定ファイルに記載された `channel` がチャンネル番号として使用されます。

4.2 デバイスコネクターからの FIFO への書き込み

デバイスコネクターから Agent にデータを送るには、Agent により生成された FIFO にデータを書き込む必要があります。

4.2.1 データの書き込み

FIFO への書き込みは、所定のフォーマットに従った形式である必要があります。フォーマットについては、Agent とデバイスコネクターの間で使われる FIFO のデータフォーマット (p. 50) を確認してください。

4.3 FIFO からの読み出し

ダウンストリームを行うためには、設定ファイルの `clients` の項目に Control クライアントを追加する必要があります。そのため、デバイスコネクターが FIFO から読み込めるデータは、Control クライアントでダウンストリーム対象としたデータ、かつ、デバイスコネクターに設定されているチャンネルのデータになります。

例（デバイスコネクターに関する設定）：

```
{
  ...
  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"
      },
  }],
  ...
}
```

FIFO からの読み込めるデータは、書き込みの際の Agent とデバイスコネクターの間で使われる FIFO のデータフォーマット (p. 50) フォーマットと同様です。

1 つのデバイスコネクターが受信するデータは、1 つのチャンネルのデータに限定されます。`channel_rx` がない場合や -1 が設定されている場合は、`channel` に設定されたチャンネル番号が受信対象となります。`channel_rx` に 0〜255 までの値が設定されている場合は、`channel_rx` に設定したチャンネル番号が受信対象となります。

ダウンストリームでは intdash サーバーと接続（再接続を含む）した際に、必ず基準時刻のデータを受信します。デバイスコネクターが基準時刻のデータを受信したくない場合は、`receive_basetime` に `false` を指定して

ください。

例（Control クライアントに関する設定）：

```
{
  "clients": [{
    ...
    "my_id": "00000000-0000-0000-0000-000000000000",
    "my_token": "hsNxJhvDNHR2QcXbXl..........Z0RWKvfPs_neAkjTNSO5",
    "ctlr_id": "9defc535-4640-4c5e-934a-bb435a89a64f",
    "ctlr_flts": [
      {
        "channel": 2,
        "dtype": 10,
        "ids": ["string_a", "string_b"]
      },
      {
        "channel": 3,
        "dtype": 14,
        "ids": []
      }
    ],
    "type": "control",
    ...
  }],
  ...
}
```

この Control クライアントの設定例では、エッジの UUID `9defc535-4640-4c5e-934a-bb435a89a64f` から送信された、以下 2 種類のデータをダウンストリームします。

- チャンネル番号：2、データタイプ：String（10）、データ ID："string_a" または "string_b"
- チャンネル番号：3、データタイプ：Bytes（14）、データ ID：全て

上記のようにデバイスコネクターと Control クライアントを設定した場合、デバイスコネクターが FIFO から受信できるデータは、エッジの UUID `9defc535-4640-4c5e-934a-bb435a89a64f` から送信された、チャンネル番号：2、データタイプ：String（10）、データ ID："string_a" または "string_b" のデータのみです。デバイスコネクターの設定でチャンネル 2 のみが指定されているため、チャンネル番号 3 のデータは受信できません。

4.4 デバイスコネクターの自動起動と自動終了

デバイスコネクターは、任意のタイミングで起動や終了をさせることができます。デバイスコネクターの起動と終了を Agent と連動させたい場合は、以下の方法が可能です。

4.4.1 Agent 起動時に自動的にデバイスコネクターを起動する

Agent が起動したときにデバイスコネクターが起動されるように設定することができます。

設定ファイルの `path` にデバイスコネクターの起動コマンドを指定し、`conf` にデバイスコネクターの設定ファイルを指定してください。この設定により、Agent が起動したときに、`path -C conf` の形式でデバイスコネクターの起動コマンドが実行されます。

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

4.4.2 Agent からのシグナルによりデバイスコネクターを終了する

Agent を終了するとき、Agent はデバイスコネクターに SIGTERM を送信します。デバイスコネクターはシグナルを検知して終了処理を行うようにしてください。

05 お問い合わせ

ご不明な点、不都合などございましたら、以下の連絡先にお問い合わせください。

株式会社アプトポッド

- サポート窓口メールアドレス VM2M-support@aptpod.co.jp
- ウェブサイト https://www.aptpod.co.jp

お問い合わせの際には、以下をお知らせください。

- intdash Edge のバージョン
- 使用している設定ファイルすべて
  - マネージャーの設定ファイル（manager.conf）（ファイル内に含まれるトークンは削除してからお送りください。）
  - デバイスコネクターの設定ファイル（例：logger.conf.xxx）

06 付録

6.1 基準時刻

基準時刻は計測の開始時刻を表す情報です。Agent は、基準時刻を適宜取得し、intdash に送信します。

> **注釈:**「計測」とは、特定のエッジから送信された時系列データのまとまりを指します。詳細については別途 iSCPv1 に関する資料を参照してください。

基準時刻には、以下の 2 つがあります。

EdgeRTC による基準時刻
: Agent を実行しているシステムのリアルタイムクロック（RTC）による基準時刻です。

NTP による基準時刻
: NTP サーバーと同期した時計による基準時刻です。NTP 基準時刻を使用するには、Status プラグインが有効になっている必要があります（インストール直後の設定ファイルでは Status プラグインが有効になっています）。

基準時刻の種類	基準時刻が決定するタイミング	基準時刻を intdash に送信するタイミング
EdgeRTC	Agent 起動時	データ送信開始時
NTP	Status プラグインが NTP サーバーと疎通した時点以降	一定周期（起動後 10 分間は 1 分間隔、そのあとは 10 分間隔）

6.2 送信側でのフィルタリング

デバイスコネクターを通じてマネージャーに集められた時系列データは、マネージャーでのフィルタリングによって、Realtime や Bulk などの各クライアントに振り分けられて intdash サーバーに送信されます。

フィルターを使用することで、データの送信方法を柔軟に切り替えることができます。

6.2.1 送信側でのフィルタリングの流れ

マネージャーは、それぞれの時系列データに対して、各クライアントを使って送信する（パス）か、もしくは送信しない（ドロップ）かを決定します。

- 特に設定がない場合は、時系列データは Realtime クライアント経由で intdash サーバーに送信されます。
- Realtime クライアント用のフィルターで「ドロップ」となったため Realtime クライアントで送信されなかったデータは、Bulk クライアントにより intdash サーバーに送信されます。
- Bulk クライアント用のフィルターでも「ドロップ」となったデータは intdash サーバーに送信されません。

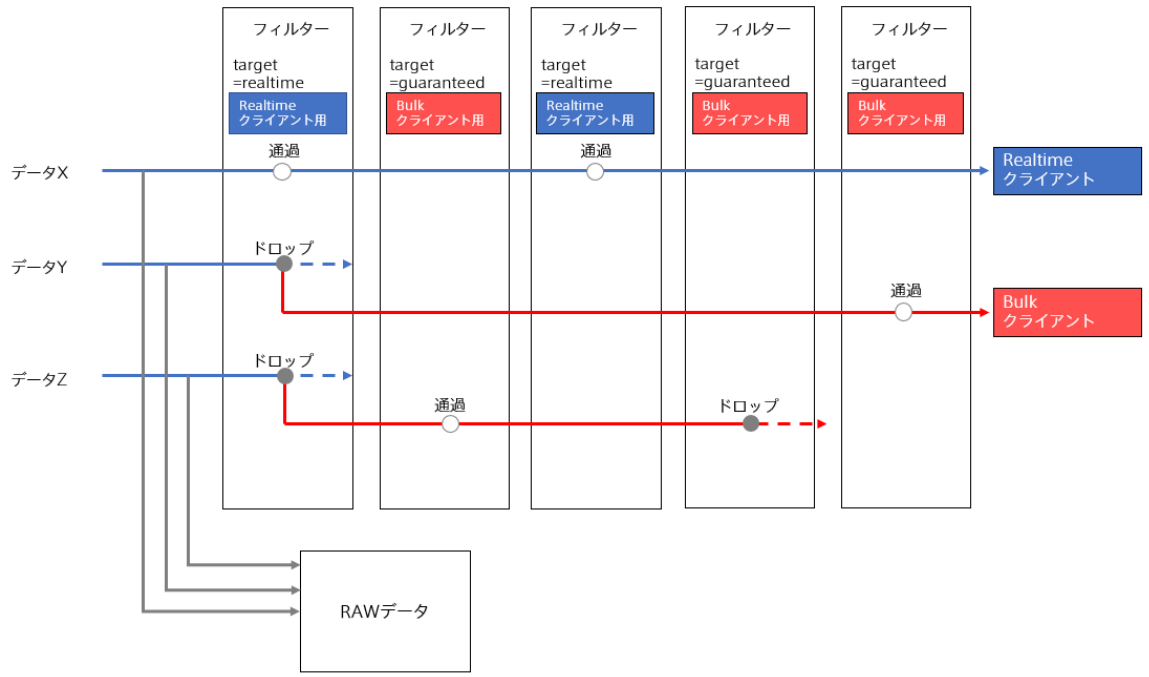
いずれの場合も、時系列データはマネージャーにより RAW データとしてダンプされます。

Realtime クライアント	Bulk クライアント	データの取り扱われ方
パス	-	intdash サーバーへリアルタイムに送信される。intdash サーバーに過去のデータとしても保存される。
ドロップ	パス	一定周期の一括伝送で intdash サーバーへ送信される。intdash サーバーに過去のデータとしても保存される。
ドロップ	ドロップ	intdash サーバーへリアルタイムに送信されない。intdash サーバーには過去のデータとしても保存されない。RAW データ (p. 45) としてエッジ側のローカルストレージに保存される。

6.2.2 フィルターの構成

フィルタリングは、複数のフィルターにより行います。複数のフィルターは順次評価され、各時系列データがどのクライアントを使って送信されるべきかを判定します。

たとえば、以下のようなフィルター構成の場合、以下のようになります。



- データ X はどのフィルターにも相当しないため Realtime クライアントにて送信されます。
- データ Y は 1 番目のフィルターにより Realtime クライアントからドロップするため、Bulk クライアントにて送信されます。
- データ Z は 1 番目のフィルターにより Realtime クライアントからドロップし、4 番目のフィルターにより Bulk クライアントからもドロップするため、送信されません。

6.2.3 フィルターの共通設定

フィルターに関する設定は、設定ファイルの `manager.filters[]` に記載します。filters フィールドの配列に格納されたオブジェクトの中身は以下のようになります。

キー	型	説明
`name`	string	フィルターの種類 (p. 41) の各フィルターの表を参照してください。
`channel`	string	フィルターを適用するチャンネルを設定します。設定できるのは"-1"、"0"～"255"です。"-1"の場合、全チャンネルが対象となります。
`target`	string	フィルターを適用するクライアントを設定します。(`realtime|guaranteed|both`)
`setting`	object[]	フィルターの設定です。フィルターの種類 (p. 41) の各フィルターの説明を参照してください。

`target` には、以下いずれかを指定します。

- `realtime`：Realtime クライアントが送るデータにフィルターを適用します
- `guaranteed`：Bulk クライアントが送るデータにフィルターを適用します
- `both`：Realtime クライアントと Bulk クライアント両方のデータにフィルターを適用します

6.2.4 フィルターの種類

フィルターには以下の種類があります。

特定のデータタイプのデータのみに適用できるフィルターと、全てのデータタイプに適用できるフィルターがあります。

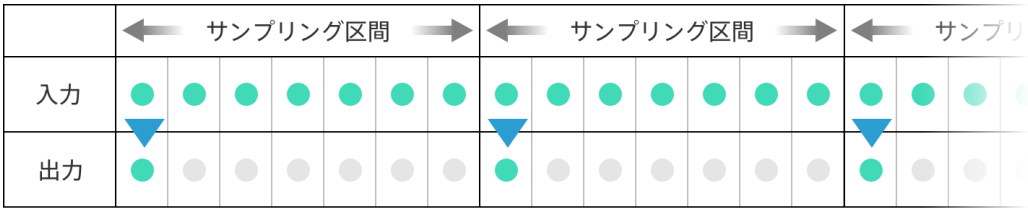
フィルター名	フィルターを適用できるデータタイプ	説明
sampling フィルター (p. 42)	NMEA、CAN、Motion JPEG のみ	サンプリングフィルター。データを間引きます。
can_id フィルター (p. 43)	CAN のみ	許可リスト型の CAN ID フィルター。指定された CAN ID のデータをパスします。
can_mask フィルター (p. 44)	CAN のみ	ブロックリスト型の CAN ID フィルター。指定された CAN ID のデータをドロップします。
channel フィルター (p. 44)	Any	ブロックリスト型のチャンネルフィルター。指定されたチャンネルのデータをドロップします。
duplicate フィルター (p. 45)	Any	データ複製フィルター。Realtime クライアントと Bulk クライアントに同じデータを複製します。

sampling フィルター

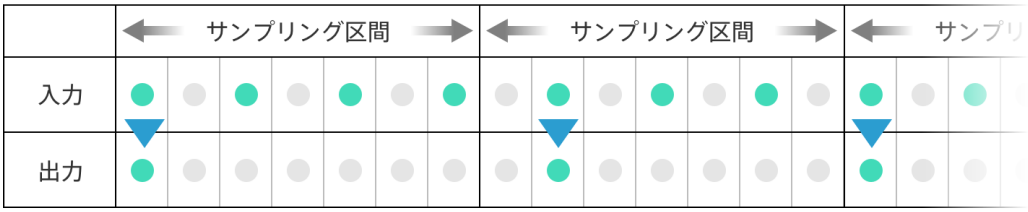
一定期間の出力が 1 データになるようにデータを間引きます。

- 任意の時間範囲（サンプリング間隔）を設定し、1 つの CAN ID でのその時間範囲の出力が 1 つのデータだけになるようにします。
  - 設定された時間範囲内で初めて来たデータはパスします。
  - 設定された時間範囲内で 2 度目以降に来たデータは破棄します。

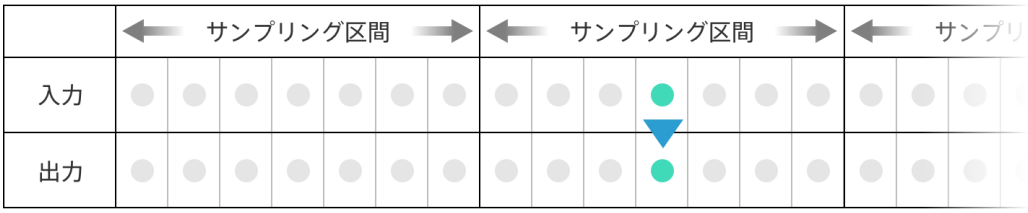
CAN ID : 1



CAN ID : 2



CAN ID : 3



sampling フィルターの setting オブジェクトの内容

key	value
rate	サンプリング間隔 [msec]

例（Realtime の 1 チャンネルを、1000[msec] の時間範囲を設定してサンプリング）：

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

(次のページに続く)

(前のページからの続き)

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

can_id フィルター

特定の CAN ID のデータをパスし、それ以外のデータをドロップする、許可リスト型のフィルターです。

- 設定された CAN ID のデータのみパスします。
  - 複数の CAN ID を設定できます。
  - 設定されていない CAN ID はドロップします。
  - 拡張 CAN ID を設定する場合は、`0x80000000` のように先頭ビットを立てた数値で入力してください。

can_id フィルターの setting オブジェクトの内容

key	value
id	通過させる CAN ID

例（Realtime のチャンネル 2 のデータのうち、CAN ID 0x00000010 と 0x00000020 だけパスする）:

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

can_mask フィルター

特定の CAN ID のデータをドロップする、ブロックリスト型のフィルターです。

- 設定された CAN ID をドロップします
  - 複数の CAN ID を設定できます。
  - 設定されていない CAN ID はパスします。
  - 拡張 CAN ID を設定する場合は、`0x80000000` のように先頭ビットを立てた数値で入力してください。

フィルターの setting オブジェクトの内容

key	value
id	ドロップする CAN ID

例（Bulk のチャンネル 1 の拡張 CAN ID 0x00000010 をドロップする）：

```
{
  "manager": {
    ...
    "filters": [{
      "name": "can_mask",
      "channel": "1",
      "target": "guaranteed",
      "setting": [
        {"key": "id", "value": "2147483664"}
      ]
    }]
  },
  ...
}
```

`2147483664` は、0x00000010 の先頭ビットを立てて、十進表記したものです。

channel フィルター

特定の channel をドロップするブロックリスト型のフィルターです。

- 設定された channel のデータをドロップします

フィルターの setting オブジェクトの内容

- 不要

例（Realtime のチャンネル 1 をドロップする）：

```
{
  "manager": {
    ...
    "filters": [{
      "name": "channel",
      "channel": "1",
```

(次のページに続く)

(前のページからの続き)

```
      "target": "realtime",
      "setting": []
    }]
  },
  ...
}
```

duplicate フィルター

データを複製します。

- データを複製し、Realtime と Bulk に同じデータを送信します

フィルターの setting オブジェクトの内容

- 不要

例（Realtime のチャンネル 1 のデータをコピーして Bulk にも送る）：

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

6.3 RAW データ

デバイスコネクターから Agent に入力された全データを、RAW データとしてダンプファイルに保存することができます。

6.3.1 RAW データの保存と自動削除

RAW データを保存するかは設定可能です。デフォルトでは保存する設定になっています。

RAW データを保存し続けるとディスクが一杯になる可能性があるため、自動的に RAW データを削除する設定があります。

```
{
  "manager": {
    "rawdir": "/var/lib/intdash/raw",
    "raw_autodelete": true,
```

(次のページに続く)

(前のページからの続き)

```
    "raw_autodelete_th": 85,
    ...
  },
  ...
}
```

raw_autodelete を true に設定すると、RAW データを保存するパーティションが一定の割合に達した場合に RAW データを自動削除します。自動削除は、RAW データ保存ディレクトリ rawdir が存在するパーティションのディスク使用率が raw_autodelete_th よりも小さくなるまで、古い RAW データから順に削除します。

6.3.2 RAW データの保存先とファイル構成

RAW データ保存ディレクトリに、計測ごとのディレクトリが作成され、その中にダンプファイルが保存されます。

```
+ /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
```

番号	説明
(1)	計測ディレクトリ
(2)	チャンネル 1 のデバイスコネクターが送信したデータのダンプ
(3)	チャンネル 2 のデバイスコネクターが送信したデータのダンプ

計測ディレクトリ

計測ディレクトリは計測ごとに新規作成されます。ディレクトリ名は、計測の EdgeRTC 基準時刻を Unix 時間でナノ秒まで表記したものです。(例：ディレクトリ名 1562549837.123456789 は、基準時刻 2019/7/8 1:37:17.123456789 UTC を意味します。)

ダンプファイル

ダンプファイルは、デバイスコネクターごとのデータのダンプファイルです。ファイル名は以下のようになります。

XXX_NNN.raw

- XXX：デバイスコネクターのチャンネル番号（10 進数：000-255）
- NNN：カウンター番号（10 進数：000-999）

ダンプファイルのサイズが 512MB 以上になると、次のカウンター番号で新しいファイルが作られます。

ダンプファイルのフォーマット

ダンプファイルは Agent とデバイスコネクターの間で使われる FIFO のデータフォーマット (p. 50) のデータをそのままファイルにダンプしたものです。

サンプル（NMEA）

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

6.3.3 RAW データを解析するためのツール

Agent をインストールすると、RAW データを解析するためのツールもインストールされます。

使用例：

```
/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 ...
```

ツールの詳しい使用方法については、`rawutil` のヘルプを参照してください。

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

6.4 再送データ

Realtime クライアントまたは Bulk クライアントによって送信が試みられ、intdash サーバーに正しく送信できなかったデータは、再送データとして保持されます。再送データとして保持された時系列データは、Resend クライアントによる再送処理によって順次再送されます。

6.4.1 ディスクの空き容量が少なくなった場合の自動停止

ディスクの使用量が閾値を超えた場合、Agent は自動停止します。Agent は以下のいずれかの条件を満たすときに自動停止します。([] は設定ファイルのキーを表します。)

- [manager.required_space] < 再送データ保存先パーティションの使用率（%）
- [manager.required_space_raw] < [manager.rawdir] のパーティションの使用率（%）

6.4.2 保存先とファイル構成

再送データ保存ディレクトリ以下に、計測ディレクトリが作成され、その中に再送セクションファイルを保存されます。

```
+ /opt/vm2m/var/lib/intdash
  + meas
    + <SERVER_NAME>                              # (1)
      + CCCCCCCCC_TTTTTTTTTT.NNNNNNNNN           # (2)
        - .S2E-<MEAS_UUID>                       # (3)
        - .meta                                  # (4)
        - SSSSSSSSSS_TTTTTTTTTT.NNNNNNNNNF.EXT   # (5)
        - SSSSSSSSSS_TTTTTTTTTT.NNNNNNNNNF.EXT
    + <SERVER_NAME>
      + CCCCCCCCC_TTTTTTTTTT.NNNNNNNNN
        - .S2E-<MEAS_UUID>
        - .meta
        - SSSSSSSSSS_TTTTTTTTTT.NNNNNNNNNF.EXT
        - SSSSSSSSSS_TTTTTTTTTT.NNNNNNNNNF.EXT
```

番号	説明
(1)	サーバーディレクトリ
(2)	計測ディレクトリ
(3)	計測 UUID
(4)	計測メタ情報
(5)	再送セクション

サーバーディレクトリ

サーバーディレクトリには、このサーバーへ送信する計測のデータが保存されます。ディレクトリ名は、送信対象サーバーの名前です。

計測ディレクトリ

計測ディレクトリは、計測ごとに新規作成されます。ディレクトリ名は、下記のとおりです。

`CCCCCCCCC_TTTTTTTTTT.NNNNNNNNN`

- `CCCCCCCCC`：計測回数
- `TTTTTTTTTT.NNNNNNNNN`：計測の EdgeRTC 基準時間（ナノ秒までの Unix Timestamp）

計測 UUID ファイル

計測 UUID ファイルは、計測 UUID をファイル名に含む空のファイルです。ユーザーが計測を見つけやすくするために作成されます。ファイル名は、`.S2E-`<計測 `UUID` 先頭 `6` 桁> です。

計測メタ情報ファイル

計測メタ情報ファイルには、計測に関する以下の情報が保存されています。

- シリアルナンバー（最後のセクションのシリアル番号）
- ユニットカウント（総ユニット数）
- 再送セクションファイル数
- 再送ファイルサイズ（総再送ファイルサイズ）
- 計測 UUID フラグ（計測 UUID を取得したか）
- 終了フラグ（計測終了をサーバーに通知したか）
- 計測タグフラグ（計測タグをサーバーに通知したか）
- 計測カウント（このターミナルで作成された計測のカウンター番号）
- EdgeRTC 基準時刻
- 計測時間

ファイル名は、`.meta` です。

再送セクションファイル

再送セクションファイルは、intdash の Section の Unit のみをダンプしたファイルです。

ファイル名は、`SSSSSSSSSS_TTTTTTTTTT.NNNNNNNNNF.EXT` です。

- S : セクションのシリアルナンバー
- T : EdgeRTC からの相対時間（秒）
- N : EdgeRTC からの相対時間（ナノ秒）
- F : フラグ（B:基準時刻を含んでいる、なし:左記以外）
- EXT : 拡張子（bin: 再送データ、store: ストア送信データ、store.bin: ストア送信の再送データ）

例：

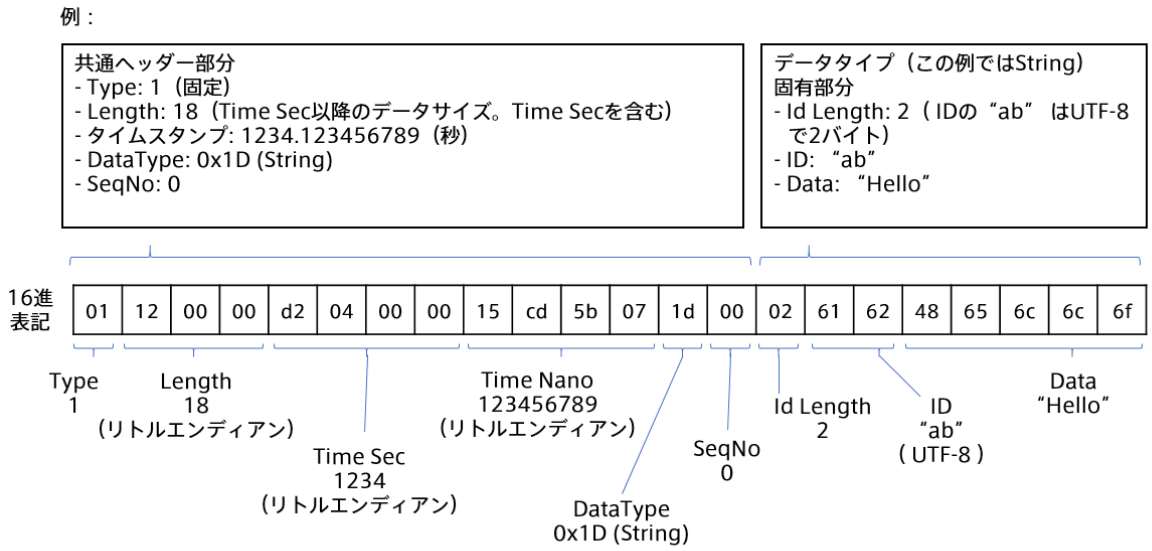
```
0000000000_0000000013.517448529B.bin
0000000001_0000000014.002924135.bin
0000000002_0000000015.000175599.store
0000000003_0000000016.002819513.store.bin
```

6.5 Agent とデバイスコネクターの間で使われる FIFO のデータフォーマット

Agent の FIFO が読み取るデータのフォーマットは、共通ヘッダー部分と、データタイプごとの固有部分から構成されます。

- 共通ヘッダー部分については、共通ヘッダー (p. 50) を参照してください。
- データタイプごとの固有部分については、データタイプごとの固有部分 (p. 52) を参照してください。

以下は、プリミティブ文字列型の「Hello」という文字列のデータの例です。



重要: Agent とデバイスコネクターの間の FIFO で使われるデータフォーマットやデータタイプと、iSCPv1 のデータフォーマットやデータタイプは異なります。

6.5.1 共通ヘッダー

```
 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                          |
+---------------+---------------+---------------+----------//---+
```

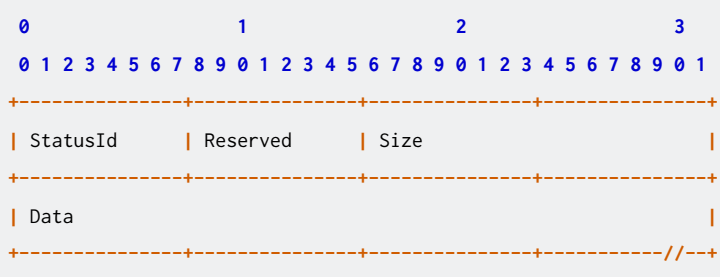
フィールド名	バイト長	エンディアン	符号	値の範囲	説明
Type	1	—	なし	1	メッセージの種類（1で固定）
Length	3	LE	なし	10-16777216	Time Sec 以降のサイズ（Time Sec を含む）
Time Sec	4	LE	なし	0-4294967295	システムの単調増加する時間（秒）[*1]
Time Nano	4	LE	なし	0-999999999	システムの単調増加する時間（ナノ秒）
DataType	1	—	なし	—	データタイプ（下記、データタイプの表を参照）
SeqNo	1	—	なし	0-255	シーケンシャル番号（0固定でもデータ送信可能）
Data	0-16777208	—	—	—	データ（データタイプ毎の項を参照してください）

データタイプ

[*1] POSIX を使用できるプログラミング言語では、`clock_gettime()` 関数に `CLOCK_MONOTONIC_RAW` を指定することで取得できます。

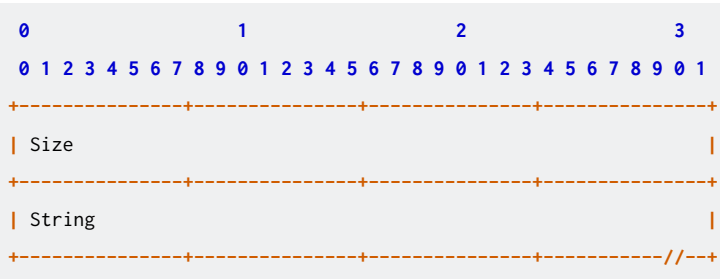
6.5.2 データタイプごとの固有部分

Status（データタイプ: 0x03）



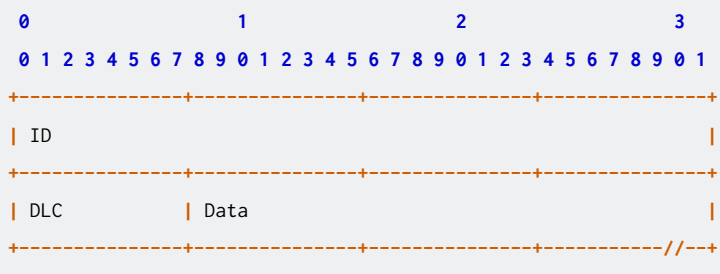
フィールド名	バイト長	エンディアン	符号	値	説明
StatusId	1	—	なし	0x90	ステータスの種類（0x90 で固定）
Size	2	LE	なし	0-32767	データサイズ
Data	0-32767	—	—	—	JSON 文字列

NMEA（データタイプ: 0x10）



フィールド名	バイト長	エンディアン	符号	値	説明
Size	4	LE	なし	0-4294967295	データサイズ
Data	0-4294967295	—	—	—	NMEA 文字列

CAN/CAN-FD（データタイプ: 0x11）



フィールド名	バイト長	エンディアン	符号	値	説明
ID	4	LE	なし	0-4294967295	CAN ID（拡張 CAN の場合は先頭ビットを立てる）
DLC	1	—	なし	0-255	データサイズ
Data	0-255	—	—	—	データ

JPEG（データタイプ: 0x12）

```
 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
+---------------+---------------+---------------+---------------+
| JPEG                                                          |
+---------------+---------------+---------------+-----------//--+
```

フィールド名	バイト長	エンディアン	符号	値	説明
JPEG	0-16777216	—	—	—	JPEG (ISO/IEC 10918-1, Annex B) のバイナリデータ

H.264（データタイプ: 0x1C）

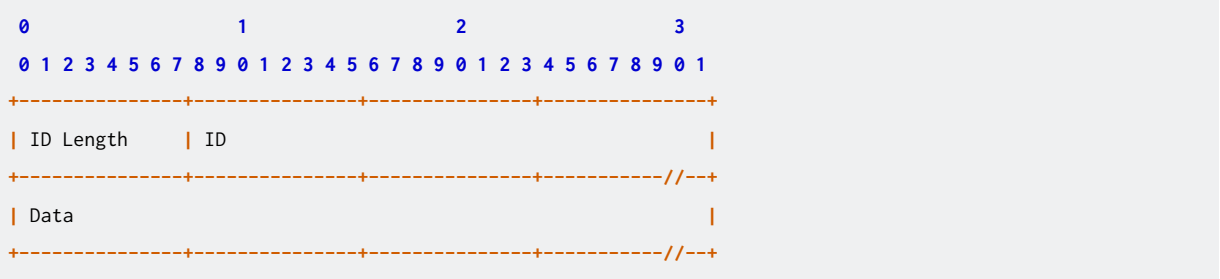
```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
| NALType       | NALUnits                                      |
+---------------+---------------+---------------+-----------//--+
```

フィールド名	バイト長	エンディアン	符号	値	説明
NALType	1	—	—	—	NALType（下記 NAL-Type を参照）
NALUnits	—	—	—	—	NALUnits

NALType

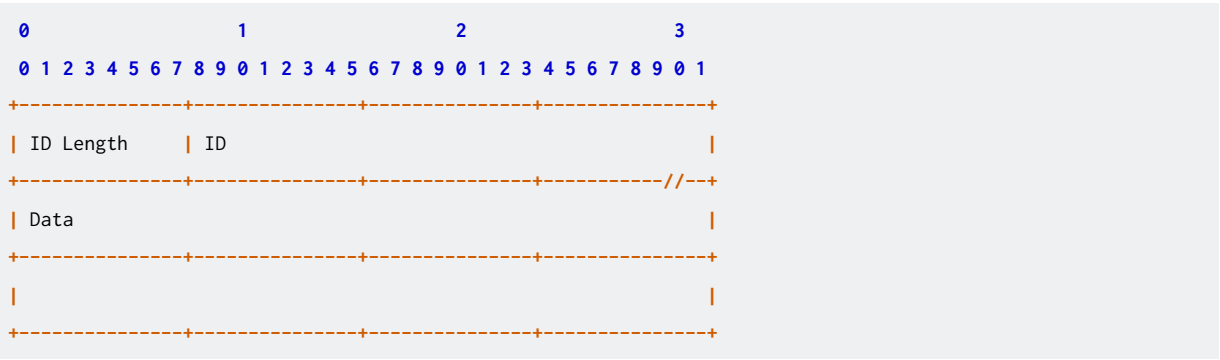
NALType	NALUnits に格納するデータ
0x00	IDR slice (nal_unit_type == 5) である NAL Units の 1 フレーム分[*2] の連結[*3]
0x01	non-IDR slice (nal_unit_type == 1) である NAL Units の 1 フレーム分の連結
0x08	H.264 のデコードに必要な NAL Units[*4] の連結。NALType 0x00 の都度、先行して 1 つ生成するものとする。

String（データタイプ: 0x1D）プリミティブ文字列型



フィールド名	バイト長	エンディアン	符号	値	説明
ID Length	1	—	—	0-255	ID の長さ
ID	0-255	—	—	0-16777215	UTF-8 エンコードした ID
Data	—	—	—	—	文字列

Float（データタイプ: 0x01E）プリミティブ Float64 型



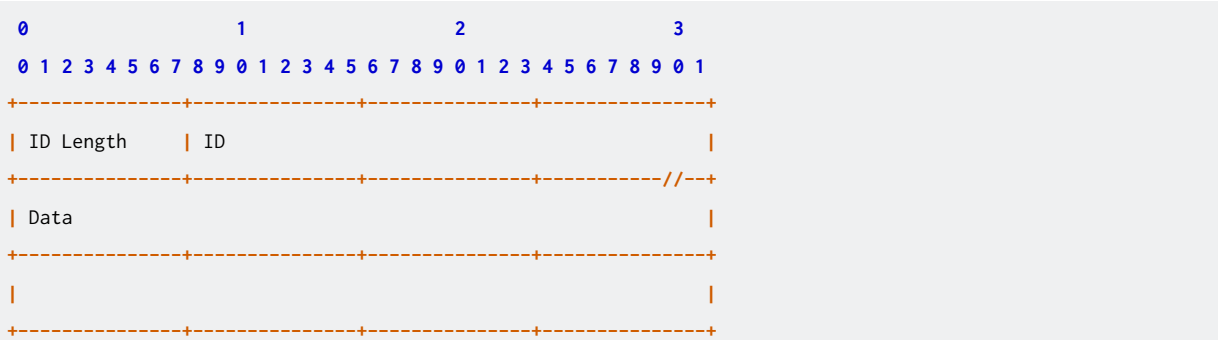
[*2] nal_unit_type が 1-5 (Coded slice) の時、同一の nal_unit_type の NAL unit について、slice_header の first_mb_in_slice が 0 であるものから、次に 0 であるものの前までの NAL units

[*3] “ITU-T Rec. H.264 ¦ ISO/IEC 14496-10 Advanced Video Coding” における “Byte stream format (Annex B)” に従い、次のように連結すること：start code prefix + NAL unit + start code prefix + NAL unit … start code prefix + NAL unit

[*4] H.264 のデコードに必要な NAL units は SPS（nal_unit_type == 7）、PPS（nal_unit_type == 8）の NAL units

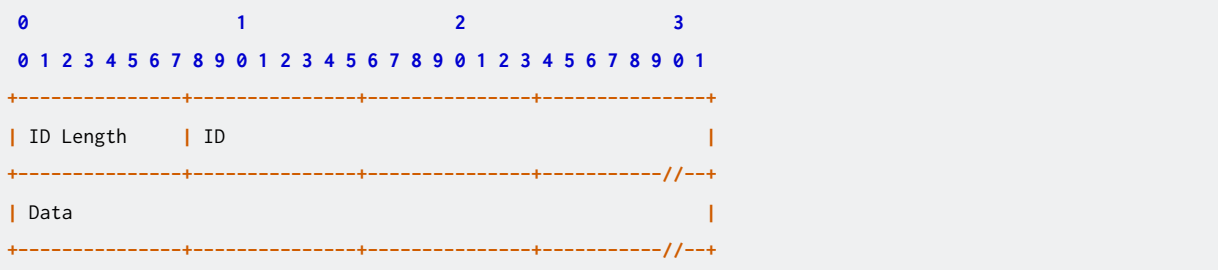
フィールド名	バイト長	エンディアン	符号	値	説明
ID Length	1	—	—	0-255	ID の長さ
ID	0-255	—	—	0-16777215	UTF-8 エンコードした ID
Data	8	LE	—	—	倍精度浮動小数点数を格納したバイト列 (IEEE 754 に準拠)

Int（データタイプ: 0x1F）プリミティブ Int64 型



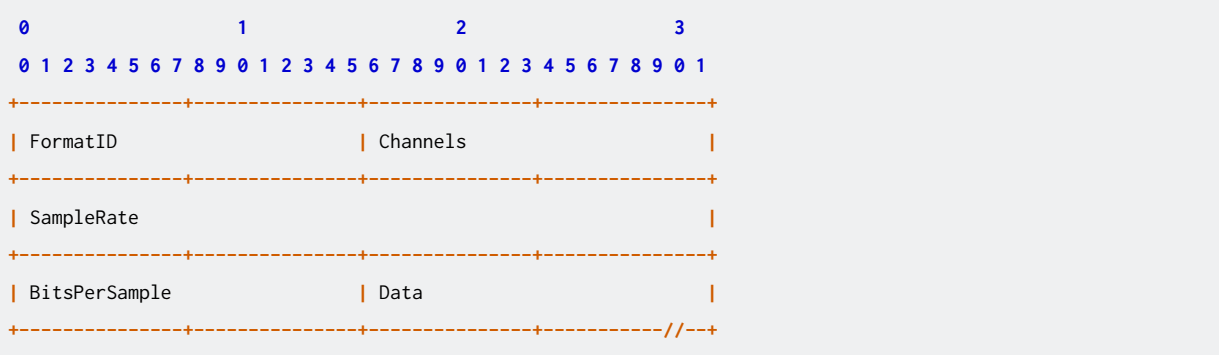
フィールド名	バイト長	エンディアン	符号	値	説明
ID Length	1	—	—	0-255	ID の長さ
ID	0-255	—	—	0-16777215	UTF-8 エンコードした ID
Data	8	LE	あり	—	Int64

Bytes（データタイプ: 0x20）プリミティブバイト配列型



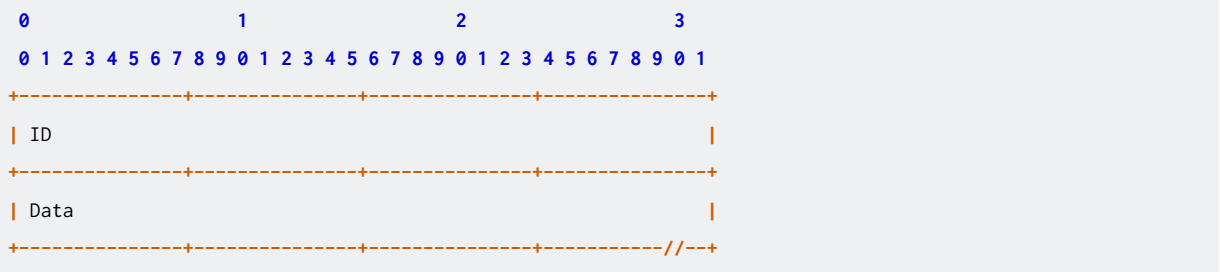
フィールド名	バイト長	エンディアン	符号	値	説明
ID Length	1	—	—	0-255	ID の長さ
ID	0-255	—	—	0-16777215	UTF-8 エンコードした ID
Data	—	—	—	—	バイナリデータ

PCM（データタイプ: 0x22）



フィールド名	バイト長	エンディアン	符号	値	説明
FormatID	2	LE	なし	0-65535	WAVE[5] で定義されたフォーマット ID
Channels	2	LE	なし	0-65535	格納される音声のチャンネル数
SampleRate	4	LE	なし	0-4294967295	サンプリング周波数 [Hz]
BitsPerSample	2	LE	なし	0-65535	ビットレート [bit/sample]
Data	—	—	—	—	WAVE[5] に準拠する波形情報

Generic（データタイプ: 0x7F）



[5] 音声データ用コンテナーフォーマットの WAVE (RIFF waveform Audio Format)

フィールド名	バイト長	エンディアン	符号	値	説明
ID	4	LE	なし	0-4294967295	数値 ID
Data	0-65531	—	—	—	任意のデータ

6.6 Agent の全設定

Agent の設定ファイルの全項目について解説します。

設定ファイルは以下のような構成になっています。各項目に設定すべき内容については以下の節を参照してください。

> **注釈:** 設定ファイル内の「logger」はデバイスコネクターを指します。

キー	型	説明
manager	object	マネージャーに関する設定 (p. 58)
clients	object[]	クライアントに関する設定 (p. 60)
loggers	object[]	デバイスコネクターに関する設定 (p. 62)

例

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

6.6.1 環境変数を使って設定を置き換える

設定ファイル内の文字列型の設定値は、環境変数により与えることができます。環境変数で設定を与えることにより、設定ファイルを書き換えなくても、起動時に柔軟に設定内容を変更することができます。

例えば、`clients.my_token` の値を環境変数で与えるためには、設定ファイル内に以下のように変数を書き込みます。以下の例では `$TOKEN` という変数を書き込んであります。

```
{
  ...
  "clients": [
    {
```

(次のページに続く)

(前のページからの続き)

```
      ...
      "my_token": "$TOKEN",
    }],
   ...
}
```

その上で、上記の変数名にプリフィックス INTDASH_EDGE_ を付けた環境変数を設定します。この例では、INTDASH_EDGE_TOKEN に値を設定します。

これらの準備をした上で、この設定ファイルを使って intdash-edge-manager を起動すると、設定ファイル内の $TOKEN が環境変数 INTDASH_EDGE_TOKEN の値に展開され、使用されます。

> **注釈:** 環境変数を使って設定を置き換えることができるのは、文字列型の値のみです。

6.6.2 マネージャーに関する設定

マネージャーに関する設定は、設定ファイルの manager フィールドにオブジェクトで設定します。設定できる項目は以下の通りです。

キー	型	デフォルト値	説明
workdirs	string[]	["/opt/vm2m/var/lib/intdash/meas", "/opt/vm2m/var/run/intdash"]	起動時に作成するディレクトリのパス
basetime	string	"/opt/vm2m/var/run/intdash/basetime"	基準時刻情報を保存するファイルのパス
meas_root	string	"/opt/vm2m/var/lib/intdash/meas"	再送データを保存するディレクトリのパス
rawdir	string	"/opt/vm2m/var/lib/intdash/raw"	RAW データを保存するディレクトリのパス
raw_autodelete	bool	true	RAW データの自動削除
raw_autodelete_th	number	85	RAW データを自動削除する閾値 [%]

次のページに続く

表 1 – 前のページからの続き

キー	型	デフォルト値	説明
required_space	number	90	Agent が計測停止の閾値 [%]。再送データを保存するパーティションの空き容量率がこの値より大きくなった場合に計測を自動停止します。設定を 100 にすると、自動停止しません。
required_space_raw	number	90	Agent が計測停止の閾値 [%]。RAW を保存するパーティションの空き容量率がこの値より大きくなった場合に計測を自動停止します。設定を 100 にすると、自動停止しません。
stat	string	"/opt/vm2m/var/run/intdash/manager.stat"	Manager のステータスを記録するファイルのパス
process_stat	string	"/opt/vm2m/var/run/intdash/process.stat"	Process のステータスを記録するファイルのパス
wwan_stat	string	"/opt/vm2m/var/run/intdash/wwan.stat"	WWAN のステータスを記録するファイルのパス
logger_stat	string	"/opt/vm2m/var/run/intdash/logger_%03hhu.stat"	デバイスコネクターのステータスを記録するファイルのパスのフォーマット。1 つ目のフォーマット指定子（デフォルト値の例では %03hhu ）がチャンネル番号に置き換えられます。
system_stat	string	"/opt/vm2m/var/run/intdash/system.stat"	System のステータスを記録するファイルのパス
filters	object[]	[]	フィルターの設定。詳細については 送信側でのフィルタリング (p. 39) を参照してください。

6.6.3 クライアントに関する設定

クライアントに関する設定は、設定ファイルの clients フィールドの配列に、オブジェクトで設定します。設定できる項目は以下の通りです。

キー	型	デフォルト値	説明
protocol	string	""	使用する通信ライブラリの名前（mod_websocket.v2|mod_http)）
type	string	""	クライアントの動作モード（realtime|bulk|resend|control）。protocol が mod_http の場合は resend のみが選択可能です。
my_id	string	""	intdash に接続する Edge の UUID
my_token	string	""	intdash に接続時に使用する EdgeToken
dst_id	string[]	[]	サーバーへ送信するデータの宛先 Edge の UUID。type が realtime|bulk|resend の場合に使用されます。(オプショナル)
ctlr_id	string	""	サーバーへ送信するデータの送信元 Edge の UUID。type が control の場合に使用されます。
ctlr_flts	object[]	[]	サーバーから受信するデータのフィルター。フィルターの内容は ctlr_flts.channel、ctlr_flts.dtype、ctlr_flts.ids を参照。
ctlr_flts.channel	number	0	サーバーから受信するデータのチャンネル。type が control の場合に使用されます。
ctlr_flts.dtype	string	0	サーバーから受信するデータのデータタイプ。type が control の場合に使用されます。

次のページに続く

表 2 – 前のページからの続き

キー	型	デフォルト値	説明
ctlr_flts.ids	number[]	[]	サーバーから受信するデータの ID。type が control の場合に使用されます。(オプショナル)
resend_cycle	number	1000	再送周期 [msec]。type が resend の場合に使用されます。
store_flushtime	number	3000	フラッシュ間隔 [msec]。type が bulk の場合に使用されます。
store_flushsize	number	10000	フラッシュ間隔 [unit 数]。type が bulk の場合に使用されます。
store_cycle	number	1000	送信間隔 [msec]。type が bulk の場合に使用されます。
http_client_count	number	1	同時再送数。protocol が mod_http の場合に使用されます。
connection.host	string	""	サーバーのホスト名 + ドメイン名（FQDN)
connection.path	string	"/"	サーバーのリソースへのパス。(protocol が mod_websocket* の場合は /api/v1/ws/measurements としてください。protocol が mod_http の場合は /api/v1/measurements としてください。)
connection.ssl	string	"secure"	SSL 接続のセキュリティー設定（none|lax|secure）。
connection.cert	string	""（OS にインストールされている証明書を使用する)	サーバー証明書のファイルパス（オプショナル)。
connection.client_cert	string	""	クライアント証明書を使用する場合の証明書（オプショナル)。
connection.client_key	string	""	クライアント証明書を使用する場合の秘密鍵（オプショナル)。

次のページに続く

表 2 – 前のページからの続き

キー	型	デフォルト値	説明
user_agent	string	"IntDash-Edge/unknown (Unknown; Unknown)"	ユーザーエージェント（オプショナル）。

6.6.4 デバイスコネクターに関する設定

デバイスコネクターに関する設定は、設定ファイルの loggers フィールドの配列にオブジェクトで設定します。設定できる項目は以下の通りです。

キー	型	デフォルト値	説明
path	string	""	自動起動するデバイスコネクターのフルパス。
conf	string	""	自動起動するデバイスコネクターに 2 番目の起動引数として渡される文字列。1 番目は -c が渡される。
details.plugin	string	""	使用するプラグイン名（fifo|status）。
details.plugin_dir	string	"/opt/vm2m/lib/plugins"	使用するプラグインの実行ファイルが保存されているディレクトリ。
details.plugin_arg	string	{}	プラグインに設定する JSON オブジェクト。設定内容はプラグインにより異なります。
details.plugin_with_process	bool	true	プラグインを使用する場合に、デバイスコネクターの自動起動を行うかの設定。
connections[].channel	number	-1	デバイスコネクターに設定するチャンネル（0-255）。
connections[].channel_rx	number	-1 （-1 の場合 channel と同じ値を使用する）	ダウンストリーム時にデバイスコネクターに送信するデータのチャンネル（0-255）。
connections[].receive_basetime	number	false	ダウンストリーム時にデバイスコネクターに基準時刻のデータが送信されるかの設定。

次のページに続く

表 3 – 前のページからの続き

キー	型	デフォルト値	説明
connections[].fifo_tx	string	"/opt/vm2m/var/run/intdash/logger_%03hhu.tx"	通信に使用する FIFO のファイルパス。1 つ目のフォーマット指定子（デフォルト値の例では %03hhu ）がチャンネル番号に置き換えられます。
connections[].fifo_rx	string	"/opt/vm2m/var/run/intdash/logger_%03hhu.rx"	通信に使用する FIFO のファイルパス。1 つ目のフォーマット指定子に（デフォルト値の例では %03hhu ）がチャンネル番号に置き換えられます、
connections[].disable_raw	number	0	このチャンネルのデータを RAW データとして保存するか（0:保存する、0 以外:保存しない)。

6.7 Agent のログ

Agent は標準出力にログメッセージを出力します。

標準出力をリダイレクトすることによって、ログメッセージの保存先をファイルにすることができます。例えば、以下のようにして Agent を起動すると、ログメッセージは、/var/run/intdash/intdash.log に出力されます。(${CONF_PATH} は、使用する設定ファイルのフルパスです。)

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

6.7.1 ログメッセージのフォーマット

ログメッセージは、以下の例のようなフォーマットで出力されます。

```
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)
```

番号	説明
(1)	発生日時
(2)	モジュール名 (プロセス番号)
(3)	プリフィックス（ INFO：正常動作での情報表示、WARN：復帰可能なエラーの情報、ERR：復帰の見込みのないエラーの情報)
(4)	関数名とログの内容

6.7.2 Agent アプリケーションに関するログメッセージ

Agent アプリケーションに関するログメッセージのうち、主なものは以下の通りです。

デバイスコネクターと接続するための FIFO の作成

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

ログメッセージ内のパスは、intdash-edge-manager により作成された FIFO のパスです。

FIFO のオープン

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

ログメッセージ内のパスは、intdash-edge-manager によりオープンされた FIFO のパスです。

FIFO のクローズ

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

ログメッセージ内のパスは、intdash-edge-manager によりクローズされた FIFO のパスです。

6.7.3 WebSocket でのリアルタイム送信に関するログメッセージ

WebSocket でのリアルタイム送信に関するログメッセージのうち、主なものは以下の通りです。

サーバーとのコネクション確立

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

1 つのセクションのデータを送信

1 つのセクションのデータを送信したことを表すメッセージです。この時点では、サーバー側での処理は完了していません。

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

番号	説明
(1)	セクションに含まれるユニットの数（基準時刻以外）
(2)	セクションに含まれる基準時刻ユニットの数
(3)	セクションに与えられた通し番号

1 つのセクションの送信が完了

サーバー側で 1 つのセクションの処理が完了したことを表すメッセージです。サーバーから ACK を受信すると、このメッセージが出力されます。

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

番号	説明
(1)	計測 ID
(2)	セクションに与えらえた通し番号
(3)	このセクションのユニット数
(4)	この計測で送信した総ユニット数

1 つのセクションのデータを再送用にファイルとして保存

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

番号	説明
(1)	セクションに含まれるユニット数
(2)	再送セクションファイルの保存先パス

サーバーとのコネクションのクローズ

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

6.7.4 HTTP による再送に関するログメッセージ

HTTP による再送に関するログメッセージのうち、主なものは以下の通りです。

1 つのセクションの再送が完了

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

番号	説明
(1)	セクションに含まれるユニット数
(2)	セクションに与えられた通し番号