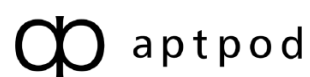


intdash Edge Agent 2 デベロッパーガイド

intdash Edge Agent 2 v1.1.0

第 2 版 (2024 年 1 月)



はじめに

01	intdash Edge Agent 2 とは	5
1.1	intdash Edge Agent 2 でできること	5
1.2	intdash Edge Agent 2 の構成	5
1.3	デバイスコネクタ	6
02	インストール	8
2.1	システム要件	8
2.2	インストールする	8
03	ライセンス	10
04	チュートリアル 1: アップストリームで送信する	11
4.1	準備するもの	11
4.2	接続先サーバーを設定する	11
4.3	リアルタイム送信を開始する	14
4.4	データを FIFO に書き込む	14
4.5	Edge Finder で確認する	15
05	チュートリアル 2: ダウンストリームで受信する	16
5.1	準備するもの	16
5.2	接続先サーバーを設定する	17
5.3	リアルタイム受信を開始する	19
5.4	データを確認する	19
06	チュートリアル 3: 付属デバイスコネクタを使用する	21
6.1	準備するもの	21
6.2	パイプライン設定を作成する	22
6.3	ストリーマーを起動する	23
6.4	Edge Finder で確認する	23
07	リアルタイム送受信の開始／終了	24
7.1	ストリーマーを起動／終了する	24
7.2	電源に連動させる	24
08	手動によるデーモン起動／終了	25
09	接続先サーバーと認証情報	26
10	トランスポートの設定	27
11	アップストリームによる送信	28
11.1	アップストリーム	28
11.2	デバイスコネクタ IPC	29
11.3	遅延アップロード用の行先 (deferred)	32
11.4	フィルター (必要な場合のみ)	32
11.5	送信の開始／終了	34
11.6	基準時刻について	35

12	ダウンストリームによる受信	36
12.1	デバイスコネクタ IPC.....	36
12.2	ダウンストリーム.....	38
12.3	フィルター（必要な場合のみ）.....	39
12.4	受信を開始／終了.....	41
13	付属デバイスコネクタの使用	42
13.1	パイプラインとエレメント.....	42
13.2	ストリームの方向とパイプライン.....	44
13.3	パイプライン設定ファイル.....	45
13.4	パイプライン設定ファイルの環境変数.....	46
13.5	パイプライン設定ファイルのサンプル.....	47
13.6	エレメント一覧.....	47
14	独自デバイスコネクタの使用	69
15	フィルタリング／サンプリング	70
15.1	フィルターとは.....	70
15.2	フィルターを設定する.....	72
15.3	フィルターのタイプ.....	73
16	遅延アップロード	79
16.1	優先度を設定する.....	79
16.2	データ蓄積の上限を設定する.....	79
17	ローカルストレージの管理	81
17.1	計測を削除する.....	81
18	ステータスの確認	82
19	ログの確認	83
20	設定の書き出し／読み込み	84
20.1	設定を出力する.....	84
20.2	ファイルから設定を読み込む.....	84
21	intdash Edge Agent 2 REST API	85
21.1	接続先サーバーと認証情報.....	85
21.2	トランスポート.....	85
21.3	ストリーム.....	85
21.4	フィルター.....	85
21.5	デバイスコネクタ IPC.....	85
21.6	遅延アップロード.....	86
21.7	ローカル計測データ.....	86
22	intdash-agentctl コマンド	87
22.1	intdash-agentctl run.....	87
22.2	intdash-agentctl config.....	88
22.3	intdash-agentctl config-file.....	94
22.4	intdash-agentctl measurement.....	95

22.5	intdash-agentctl status	95
22.6	intdash-agentctl ping	96
22.7	intdash-agentctl about	96
22.8	intdash-agentctl help	96
23	intdash-agentd コマンド	97
23.1	intdash-agentd serve	97
23.2	intdash-agentd help	98
24	設定一覧	99
24.1	設定の例	99
24.2	connection	101
24.3	transport	101
24.4	upstream	102
24.5	downstream	102
24.6	filters_upstream	103
24.7	filters_downstream	103
24.8	device_connectors_upstream	104
24.9	device_connectors_downstream	104
24.10	deferred_upload	105
25	付与されるデータ ID	106
25.1	データポイントへの ID 付与	106
25.2	iSCP v1 互換のデータ ID	107
25.3	iSCP v1 互換のデータ名称	108
26	FIFO 用データフォーマット	110
26.1	iscp-v2-compatible の全体像	110
26.2	Timestamp フィールドの意味	111
26.3	Data Name フィールドの意味	114
27	環境変数一覧	115
28	設定レシピ集	116
28.1	H.264 NAL Unit を使って H.264 データを送信する	116
28.2	カメラから V4L2 経由で取得した JPEG データを送信する	120
28.3	EDGEPLANT ANALOG-USB Interface のデータを送信する	122
28.4	EDGEPLANT CAN-USB Interface のデータを送信する	126
28.5	受信データを EDGEPLANT CAN-USB Interface に送信する	128
28.6	u-blox GNSS モジュールから取得した UBX プロトコルのメッセージを送信する	131
28.7	NMEA データを送信する	133
28.8	音声を送信する	135

01 intdash Edge Agent 2 とは

intdash Edge Agent 2 は、intdash サーバーとの間で時系列データの送受信を行うエージェントソフトウェアです。エッジデバイス（Linux PC）にインストールして使用します。

intdash Edge Agent 2 を使って時系列データを intdash サーバーに送ることで、intdash サーバーにそのデータを「計測 (measurement)」として保存することができます。

1.1 intdash Edge Agent 2 でできること

intdash Edge Agent 2 は以下を行います。

- iSCP (intdash Stream Control Protocol) を使って、時系列データをサーバーにストリーミング送信する、およびサーバーからストリーミング受信する（intdash サーバーのリアルタイム API を使った送受信）
- サーバーにデータをストリーミング送信する前に、データをフィルタリングまたはサンプリングする
- ネットワークの切断により送信できなかった時系列データを、自動的に後から送信する（intdash サーバーの REST API を使った遅延アップロード）

重要: intdash Edge Agent 2 は、従来提供していた旧 intdash Edge Agent を使いやすくリニューアルした新しいプロダクトです。旧 intdash Edge Agent とは操作や設定の方法が異なり、設定ファイルのフォーマットに互換性がない点にご注意ください。

1.2 intdash Edge Agent 2 の構成

intdash Edge Agent 2 の本体は、ストリーミング送受信を行うストリーマー（intdash-agent-streamer）と、遅延アップロードやローカルでの時系列データの管理を行うデーモン（intdash-agentd）です。

ユーザーは専用コマンド（intdash-agentctl）を使ってこれら进行操作します。

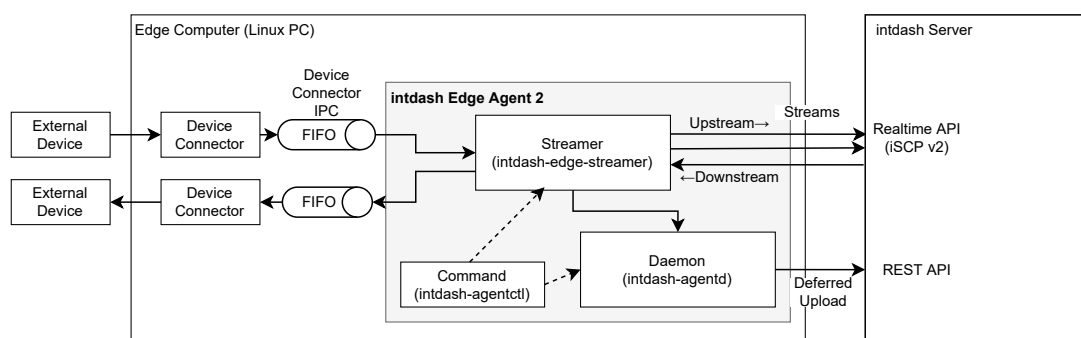


図 1 intdash Edge Agent 2 と外部デバイス、intdash サーバー

ストリーマー、デーモン、コマンドの機能はそれぞれ以下のとおりです。

1.2.1 ストリーマー (intdash-agent-streamer)

ストリーマー (intdash-agent-streamer) は、intdash サーバーとの間でリアルタイム API (iSCP) によるリアルタイムストリーミング送信または受信を行うプログラムです。ストリーマーの起動から終了までの時系列データが、intdash の 1 つの計測として扱われます。リアルタイムに送信できなかった時系列データは intdash-agentd に渡され、遅延アップロードされます。

また、ストリーマーでは、時系列データに対するフィルタリング/サンプリング処理も行うことができます。

ストリーマーの起動は intdash-agentctl コマンドを使って行います。

注釈: intdash において、リアルタイム API を使ってエッジからサーバーにデータを送信する経路を「アップストリーム」と呼び、逆に、サーバーからエッジにデータを送信する経路を「ダウンストリーム」と呼びます。単に「ストリーム」と呼んだ場合は、アップストリームまたはダウンストリーム双方のことを指します。

1.2.2 デーモン (intdash-agentd)

intdash-agentd は、遅延アップロード、設定の管理、ローカルでの時系列データの管理を行います。

- ネットワークの切断や帯域不足によりストリーマーがサーバーに送信できなかった時系列データは、後から intdash-agentd によりアップロードされます (遅延アップロード)。
- ユーザーが intdash-agentctl コマンドを使って intdash Edge Agent 2 の設定を変更すると、その設定は intdash-agentd によりファイルシステム上に保存されます。
- intdash Edge Agent 2 における時系列データの管理は、intdash-agentd によって行われます。

インストールされた直後の設定では、エッジデバイスが起動すると intdash-agentd は自動的に起動されます。

1.2.3 コマンド (intdash-agentctl)

intdash-agentctl は、前述のストリーマーやデーモンを操作するためのコマンドです。

以下の構文で使します。

```
intdash-agentctl <command> [<command_options>] [<arguments>...]
```

例えば、intdash-agentctl config connection --get を実行すると、intdash-agentd が管理している connection の設定値が返されます。

また、intdash-agentctl run を実行すると、現在の設定に従ってストリーマーが起動されます。

詳細については、[intdash-agentctl コマンド](#) (p. 87) を参照してください。

1.3 デバイスコネクター

外部デバイス (データを生成するセンサー、または、制御信号を受けて動作するアクチュエーターなど) との間でデータをやり取りするには、外部デバイスと intdash Edge Agent 2 の間を仲介するソフトウェアが必要です。このソフトウェアを「デバイスコネクター」と呼びます。

1.3.1 デバイスコネクターの機能

デバイスコネクターは、intdash Edge Agent 2 と同じエッジデバイスにインストールして使用します。以下のような機能を担います。

- データをサーバーに送信する場合 (アップストリーム): 外部デバイスからデータを受け取り intdash Edge Agent 2 に渡す。
- 逆に、データをサーバーから受信する場合 (ダウンストリーム): intdash Edge Agent 2 からデータを受け取り、外部デバイスに渡す。

デバイスコネクターと intdash Edge Agent 2 の間のデータの受け渡しには、Agent により作成される FIFO (名前付きパイプ) を使います。

intdash Edge Agent 2 とデバイスコネクター間の通信 (プロセス間通信、Inter-Process Communication) の設定は、デバイスコネクター IPC 設定で行います。

1.3.2 付属デバイスコネクターと独自デバイスコネクター

intdash Edge Agent 2 パッケージをインストールすると、依存パッケージとして、アプトポッド製デバイスコネクター `device-connector-intdash` がインストールされます。

注釈: 本ドキュメント内で、「付属デバイスコネクター」は、`device-connector-intdash` を指します。

付属デバイスコネクターを使用すると、アプトポッド製ペリフェラル製品 (CAN-USB Interface、ANALOG-USB Interface 等) からのデータ取得、カメラからの H.264 や JPEG 形式での動画取得、GPS デバイスからの NMEA メッセージの取得などを行うことができます。

付属デバイスコネクターを使用する場合は、取得するデータに応じて適切な設定を行う必要があります。付属デバイスコネクターの設定方法については [付属デバイスコネクターの使用](#) (p. 42) を参照してください。

付属デバイスコネクターが対応していないデータやデバイスを扱いたい場合は、独自のデバイスコネクターを開発することで対応できます。デバイスコネクターの開発には、デバイスコネクター開発用フレームワーク「Device Connector Framework」を使用することができます。詳細については、「intdash Edge Agent 用 Device Connector デベロッパーガイド」を参照してください。

注釈: `device-connector-intdash` パッケージには、パイプライン設定ファイルのサンプルも含まれます。パイプライン設定ファイルのサンプルは、`/etc/dc_conf` ディレクトリにインストールされます。

02 インストール

2.1 システム要件

intdash Edge Agent 2 を正しくインストールするには以下の要件を満たす必要があります。

最低ハードウェア要件

- Intel Atom プロセッサ E3815、1.46GHz 相当以上


推奨ハードウェア要件

- マルチコア CPU
- 2GB 以上のメモリー
- SSD

ディストリビューションとアーキテクチャー

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	22.04(LTS), 20.04(LTS), 18.04(LTS)	x86_64 (or amd64), armhf, arm64
Debian	11, 10, 9	x86_64 (or amd64), armhf, arm64

2.2 インストールする

 **警告:** intdash Edge Agent 2 は、旧 intdash Edge Agent と互換性がありません。旧 intdash Edge Agent を使用していたデバイスに intdash Edge Agent 2 をインストールする場合は、旧 intdash Edge Agent をアンインストールしてください。

旧 intdash Edge Agent のアンインストールは以下のコマンドで行います。

```
$ sudo apt-get purge intdash-edge
```

intdash Edge Agent 2 は、アプトポッドの公開リポジトリで「intdash-edge-agent」パッケージとして提供されています。intdash Edge Agent 2 をインストールするには、インストール先エッジデバイスのターミナルで以下のコマンドを実行します。

コマンド内の `${DISTRIBUTION}` には、ご使用の環境に応じて、`ubuntu` または `debian` を指定してください。

```
$ sudo apt-get update
$ sudo apt-get install -y \
  apt-transport-https \
  ca-certificates \
  curl \
  gnupg-agent \
  lsb-release
$ sudo mkdir -p /etc/apt/keyrings
$ curl -fsSL https://repository.aptpod.jp/intdash-edge/linux/${DISTRIBUTION}/gpg | \
  sudo gpg --dearmor -o /etc/apt/keyrings/intdash-edge.gpg
$ echo "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/intdash-edge.gpg] \
  https://repository.aptpod.jp/intdash-edge/linux/${DISTRIBUTION} \
  $(lsb_release -cs) \
  stable" \
  | sudo tee /etc/apt/sources.list.d/intdash-edge.list
```

(次のページに続く)

(前のページからの続き)

```
$ sudo apt-get update
$ sudo apt-get install -y intdash-edge-agent2
```

注釈: 上記のインストール手順では、推奨される依存パッケージも含めてインストールされます。
必要最低限の機能のみインストールしたい場合は、最後の `apt-get install` コマンドに `--no-install-recommends` オプションを付けて実行してください。

```
$ sudo apt-get install -y --no-install-recommends intdash-edge-agent2
```

注釈: 特定のユーザーに intdash Edge Agent 2 を実行するための権限を与えたい場合は、以下のコマンドを実行してください。<user_name> に、権限を付与するユーザー名を指定します。

```
$ sudo usermod -aG intdash <user_name>
```

インストール後、以下のコマンドを実行すると、intdash Edge Agent 2 のバージョンが表示されます。

```
$ intdash-agentctl --version
intdash Edge Agent 2 version 1.0.0
```

注釈: intdash Edge Agent 2 の詳細なバージョンを確認したい場合は `intdash-agentctl about` コマンドを実行してください。

```
$ intdash-agentctl about
intdash Edge Agent 2 version 1.0.0
```

Modules:

```
intdash-agentctl 1.0.0
intdash-agent-streamer 1.0.0
intdash-agentd version 1.0.0
```

Environment variables:

```
AGENT_LOG      Log level (d[debug]|i[nformation]|w[arning]|e[rrror]|q[uiet])
AGENT_VERSION  File name of agent version file
AGENT_DAEMON   Executable file name of Agent Daemon
AGENT_STREAMER Executable file name of Agent Streamer
AGENT_INFO_DIR Directory name of exporting information files
```

03 ライセンス

intdash Edge Agent 2

intdash Edge Agent 2 は、[Apache License, Version 2.0](#) によりライセンスされています。

ライセンステキストは、`/usr/share/doc/intdash-edge-agent2/copyright` としてインストールされます。

device-connector-intdash

付属デバイスコネクタ `device-connector-intdash` は、[Apache License, Version 2.0](#) によりライセンスされています。

ライセンステキストは、`/usr/share/doc/device-connector-intdash/copyright` としてインストールされます。

04 チュートリアル 1: アップストリームで送信する

このチュートリアルでは、intdash Edge Agent 2 から intdash サーバーに、「Hello」という文字列を送信します（アップストリーム）。

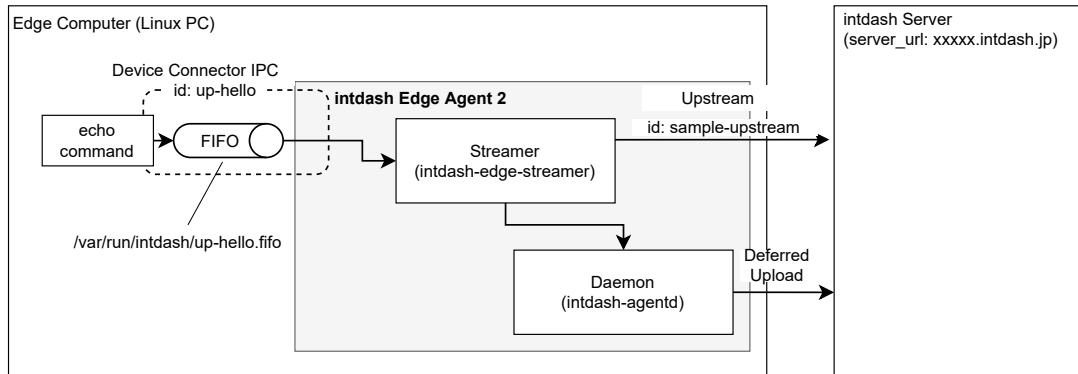


図 2 チュートリアル 1

操作手順は以下のようになります。

1. 接続先サーバーを設定する
2. アップストリームの設定を作成する
3. デバイスコネクター IPC の設定を作成する
4. リアルタイム送信を開始する
5. データを FIFO に書き込む（ここでは手順を簡単にするため、デバイスコネクターによる FIFO への書き込みを echo コマンドで代替します）

4.1 準備するもの

このチュートリアルを実行するには以下が必要です。

- intdash Edge Agent 2 をインストール済みのエッジデバイス
- 接続先の intdash サーバーのホスト名（例：xxxxx.intdash.jp）
- プロジェクト UUID
- このエッジデバイスに割り当てる、intdash のエッジ UUID とクライアントシークレット
 - エッジ UUID とクライアントシークレットは、ウェブブラウザで intdash にログイン後、My Page で発行することができます。エッジは使用するプロジェクトに所属させておいてください。

4.2 接続先サーバーを設定する

注釈: デーモン (intdash-agentd) の自動起動が行われない環境 (Docker Hub で公開されている Ubuntu など) をご使用の場合は、`sudo /etc/init.d/intdash-agentd start` でデーモンを起動してから以下の操作を行ってください。

intdash-agentctl config コマンドを使って、接続先サーバーと認証情報を設定します。

```
$ intdash-agentctl config connection --modify '  
  server_url: https://xxxxxx.intdash.jp  
  project_uuid: 00000000-0000-0000-0000-000000000000  
  edge_uuid: 03ace3b1-d208-4fc3-xxxx-xxxxxxxxxxxxx  
  client_secret: sEh9ZHPoKX8QYU-v0Noe0ZPzxGBF.....iBn5fn_eFM  
,
```

--modify に続く引数は YAML 形式の文字列です。あらかじめ準備した以下の情報を設定してください。

キー	入力する値
server_url	接続先の intdash サーバーのホスト名
project_uuid	使用するプロジェクトの UUID
edge_uuid	エッジの UUID
client_secret	エッジのクライアントシークレット

注釈: YAML 形式ではなく JSON 形式で設定を与えたい場合は、config コマンドの後に -j を指定してください。

```
$ intdash-agentctl config -j connection --modify '{  
  "server_url": "https://xxxxxx.intdash.jp",  
  "project_uuid": "00000000-0000-0000-0000-000000000000",  
  "edge_uuid": "03ace3b1-d208-4fc3-xxxx-xxxxxxxxxxxxx",  
  "client_secret": "sEh9ZHPoKX8QYU-v0Noe0ZPzxGBF.....iBn5fn_eFM"  
}'
```

注釈: intdash-agentctl コマンドの使用方法は -h オプションにより確認できます。コマンドごとに異なる情報が表示されます。

```
$ intdash-agentctl -h  
$ intdash-agentctl config -h  
$ intdash-agentctl config connection -h
```

4.2.1 アップストリームの設定を作成する

次に、iSCP によるデータ送信の経路（アップストリーム）の設定を作成します。ここでは ID のみを指定していますが、これにより、デフォルト設定のアップストリームが作成されます。アップストリームの ID は任意の文字列を設定可能ですが、ここでは ID を sample-upstream とします。

```
$ intdash-agentctl config upstream -c '  
  id: sample-upstream  
,
```

4.2.2 デバイスコネクター IPC の設定を作成する

デバイスコネクター IPC の設定（使用する FIFO のパスなど）を、以下のコマンドで作成します。

データはデバイスコネクターから intdash Edge Agent 2 へ流れ、その後 intdash サーバーへ流れるため、ここではアップストリーム方向 (upstream) のデバイスコネクター IPC 設定を作成します。ID には任意の文字列を設定可能です。

```
$ intdash-agentctl config device-connector upstream --create '
  id: up-hello
  data_name_prefix: v1/1
  dest_ids:
    - sample-upstream
  format: iscp-v2-compatible
  ipc:
    type: fifo
    path: /var/run/intdash/up-hello.fifo
'
```

このコマンドにより以下の設定が行われます。

キー	値
id	このデバイスコネクター IPC 設定に与える任意の ID
data_name_prefix	デバイスコネクターで取得したデータポイントに与えるデータ名称の接頭辞
dest_ids	デバイスコネクターで取得したデータポイントを intdash サーバーに送信する際に使用する、行先のアップストリームの ID（ここでは、さきほど作成した sample-upstream というアップストリームを指定しています。）
format	デバイスコネクターとの通信に使用するデータフォーマット（ここでは、iscp-v2-compatible を使用します。）
ipc.type	intdash Edge Agent 2 とデバイスコネクターの間のデータ通信方法（現在 fifo のみに対応しているため、fifo を指定してください。）
ipc.path	intdash Edge Agent 2 がデバイスコネクターからデータを受け取るための FIFO のパス。この FIFO に書き込まれたデータが intdash Edge Agent 2 に渡されます。

重要: 受信側で旧バージョンの iSCP を使用する場合は、送信側はルールに沿って data_name_prefix を与える必要があります。詳細は [iSCP v1 互換のデータ ID](#) (p. 107) を参照してください。

注釈: intdash-agentctl のコマンドでは短縮形を使用することもできます。以下のコマンドは上記の例と同等です。

```
$ intdash-agentctl conf dc up -c '
  id: up-hello
  data_name_prefix: v1/1
  dest_ids:
    - sample-upstream
  format: iscp-v2-compatible
  ipc:
    type: fifo
    path: /var/run/intdash/up-hello.fifo
'
```

コマンドの短縮形はヘルプで確認することができます。

```
$ intdash-agentctl conf -h
```

4.3 リアルタイム送信を開始する

以下のコマンドを実行します。

```
$ intdash-agentctl run
```

これによりストリーマーが起動され、また、デバイスコネクターからデータを受け取るための `/var/run/intdash/up-hello.fifo` が作成されます。

この時点で intdash サーバーとの接続が確立され、intdash 上は計測が開始されたこととなりますが、まだデバイスコネクターからデータポイントが流れてこないため、送信されません。

4.4 データを FIFO に書き込む

echo コマンドを使用して FIFO にバイナリデータを書き込みます。それを受け取った intdash Edge Agent 2 がデータをサーバーに送信することを確認します。新しいターミナルを開いて、以下のコマンドを実行してください。

```
$ echo -en \  
"\xd2\x04\x00\x00\x15\xcd\x5b\x07\x06\x00\x02\x00\x05\x00\x00\x00\x73\x74\x72\x69\x6e\x67\x61\x62\x48\x65\  
↪\x6c\x6c\x6f" \  
>/var/run/intdash/up-hello.fifo
```

上記で echo によって書き込まれるバイナリデータは、「Hello」という文字列に、以下の表のようにデータ型、タイムスタンプ、データ IDなどを付与し、専用のフォーマット（FIFO 用データフォーマット）に従ってエンコードしたものです。

注釈: echo ではなく Python を使ってバイナリデータを作成する場合は以下のようにします。

```
$ python3 -c "import struct, sys; sys.stdout.buffer.write(struct.pack('<LLHHL6s2s5s', 1234, 123456789, 6, \  
↪2, 5, b'string', b'ab', b'Hello'))" > /var/run/intdash/up-hello.fifo
```

データを FIFO に書き込む際のバイナリデータのフォーマットについては、[FIFO 用データフォーマット](#) (p. 110) を参照してください。デバイスコネクター IPC 設定で指定したとおり、ここでは、データフォーマット `iscp-v2-compatible` を使用しています。

フィールド名	値
タイムスタンプ	1234.123456789 秒（リトルエンディアン:整数秒部 <code>\xd2\x04\x00\x00</code> 、ナノ秒部 <code>\x15\xcd\x5b\x07</code> ）
Data Type Length	6（リトルエンディアン: <code>\x06\x00</code> ）
Data Name Length	2（リトルエンディアン: <code>\x02\x00</code> ）
Payload Length	5（リトルエンディアン: <code>\x05\x00\x00\x00</code> ）
Data Type	string（UTF-8 エンコード: <code>\x73\x74\x72\x69\x6e\x67</code> ）
Data Name	ab（UTF-8 エンコード: <code>\x61\x62</code> ）
Payload	Hello（UTF-8 エンコード: <code>\x48\x65\x6c\x6c\x6f</code> ）

このように FIFO にデータポイントを書き込むと、intdash Edge Agent 2 がサーバーに送信するデータポイントのデータ ID（<データ型>:<データ名称>）は、`string:v1/1/ab` となります。これは、デバイスコネクター IPC 設定において `data_name_prefix: v1/1` のように設定され、FIFO に書き込んだバイナリデータにおいて、Data Type として `string` が指定され、ID として `ab` が指定されているためです。

注釈: プレフィックスの v1/1 は、旧バージョンの iSCP でダウンストリームできるようにするための設定です。このあとの手順で可視化できるようにするために、設定しておく必要があります。

4.5 Edge Finder で確認する

ウェブブラウザで Edge Finder を開き、使用しているエッジのトラフィック画面を表示します。データが送信されていることが確認できます。

注釈: Edge Finder のトラフィック画面では、ページを開いたあとに受信したデータしか表示されません。トラフィック画面を開いてから、前の手順の echo コマンドを繰り返してください。



図 3 Edge Finder でトラフィックを確認する

確認が済んだら、`intdash-agentctl run` を実行したターミナルで `Ctrl+C` を押して、リアルタイム送信を終了します。

05 チュートリアル 2: ダウンストリームで受信する

このチュートリアルでは、[チュートリアル 1](#) (p. 11) でサーバーに送信されているデータを、別の intdash Edge Agent 2 で受信します。

intdash サーバーから受信したデータは、通常、FIFO を使ってデバイスコネクタに渡され、デバイスコネクタから外部デバイスに渡されますが、ここでは手順を簡単にするため、デバイスコネクタの代わりに cat コマンドを使ってデータを確認します。

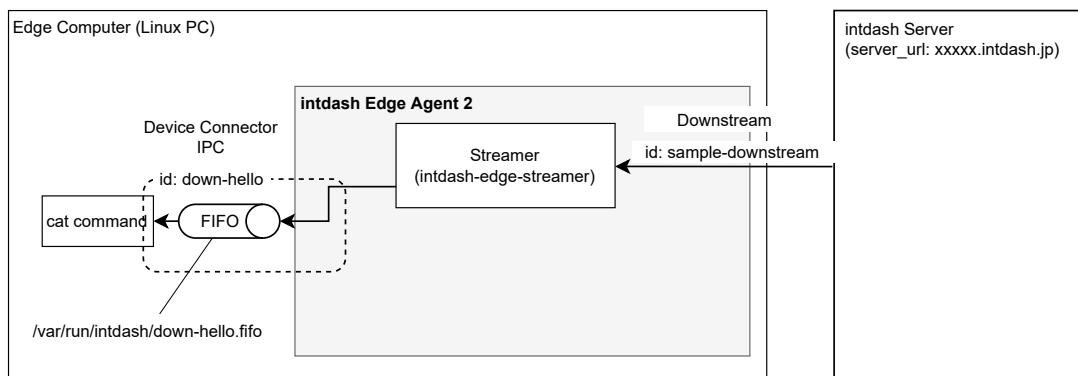


図 4 チュートリアル 2

操作手順は以下のようになります。

1. 接続先サーバーを設定する
2. デバイスコネクタ IPC の設定を作成する
3. ダウンストリームの設定を作成する
4. リアルタイム受信を開始する
5. データを確認する

5.1 準備するもの

注釈: このチュートリアルでは、アップストリーム用エッジとダウンストリーム用エッジの合計 2 つのエッジを使用します。このチュートリアルを実行する前に、先に [チュートリアル 1](#) (p. 11) を行って、アップストリーム用エッジをセットアップしてください。ダウンストリーム用エッジはこのあとの手順でセットアップします。

ダウンストリーム用エッジをセットアップするには、以下が必要です。

- intdash Edge Agent 2 をインストール済みのエッジデバイス
- 接続先の intdash サーバーのホスト名（アップストリーム用エッジに設定したものと同一）
- プロジェクト UUID（アップストリーム用エッジに設定したものと同一）
- エッジデバイスに割り当てるエッジ UUID とクライアントシークレット（ダウンストリーム用エッジ）
 - エッジ UUID とクライアントシークレットは、ウェブブラウザで intdash にログイン後、My Page で発行することができます。エッジは使用するプロジェクトに所属させておいてください。

注釈: 1つのエッジ（1つの intdash Edge Agent 2）でアップストリームとダウンストリームを行うことも可能です。

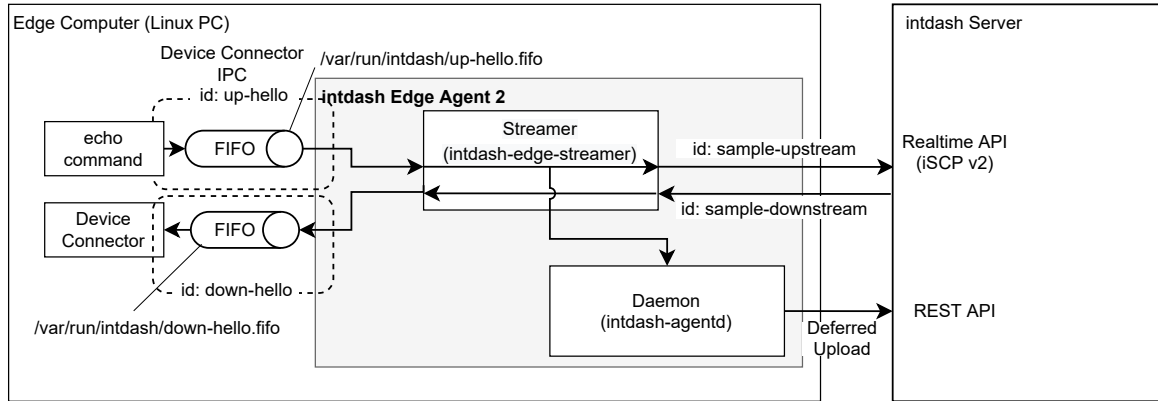


図 5 アップストリームとダウンストリームを 1つのエッジで行う

5.2 接続先サーバーを設定する

ダウンストリーム用エッジでの操作

注釈: デーモン (intdash-agentd) の自動起動が行われない環境 (Docker Hub で公開されている Ubuntu など) をご使用の場合は、`sudo /etc/init.d/intdash-agentd start` でデーモンを起動してから以下の操作を行ってください。

intdash-agentctl config コマンドを使って接続先サーバーと認証情報を設定します。

```
$ intdash-agentctl config connection --modify '
  server_url: https://xxxxxx.intdash.jp
  project_uuid: 00000000-0000-0000-0000-000000000000
  edge_uuid: xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx
  client_secret: f10MTtnSpE3LAh0arkUqgMywVVg.....JhvNgPJFRk
'
```

--modify に続く引数は YAML 形式の文字列です。あらかじめ準備した接続情報を使用してください。

キー	入力する値
server_url	intdash サーバーのホスト名
project_uuid	使用するプロジェクトの UUID
edge_uuid	ダウンストリーム用エッジの UUID
client_secret	ダウンストリーム用エッジのクライアントシークレット

5.2.1 デバイスコネクター IPC の設定を作成する

ダウンストリーム用エッジでの操作

デバイスコネクター IPC の設定（FIFO のパスなど）を、以下のコマンドで新規作成します。

データの流れは「intdash Edge Agent 2 からデバイスコネクターへ」という方向なので、ダウンストリーム方向（downstream）のデバイスコネクター IPC として作成します。

```
$ intdash-agentctl config device-connector downstream --create '
  id: down-hello
  enabled: true
  format: iscp-v2-compat
  ipc:
    type: fifo
    path: /var/run/intdash/down-hello.fifo
'
```

このコマンドにより以下の設定が行われます。

キー	値
id	このデバイスコネクター IPC 設定に与える任意の ID
enabled	デバイスコネクター IPC を有効化 (true)
format	デバイスコネクターとの通信に使用されるデータフォーマット（ここでは、iscp-v2-compat を使用します。）
ipc.type	intdash Edge Agent 2 とデバイスコネクターの間のデータ通信方法（現在 fifo のみに対応しているため、fifo を指定してください。）
ipc.path	デバイスコネクターが intdash Edge Agent 2 からデータを受け取るための FIFO のパス。

5.2.2 ダウンストリームの設定を作成する

ダウンストリーム用エッジでの操作

次に、iSCP によるデータ受信の経路（ダウンストリーム）の設定を作成します。ダウンストリームの ID は任意の文字列を設定可能ですが、ここでは down という ID とします。

```
$ intdash-agentctl config downstream --create '
  id: sample-downstream
  enabled: true
  dest_ids:
    - down-hello
  filters:
    - src_edge_uuid: 03ace3b1-d208-4fc3-xxxx-xxxxxxxxxxxx
      data_filters:
        - type: string
          name: v1/1/ab
'
```

キー	入力する値
id	このダウンストリームに与える任意の ID
enabled	ダウンストリームを有効化
dest_ids	このダウンストリームで受信したデータを渡す先の、デバイスコネクター IPC の ID。さきほど作成した down-hello というデバイスコネクター IPC を指定しています。
filters	受信対象とするデータポイントの指定
filters[].src_edge_uuid	受信対象とする送信元のエッジ UUID（チュートリアル 1 で使用したエッジ）
filters[].data_filters	受信対象とするデータ ID を定義した設定のリスト（チュートリアル 1 の設定に合わせる）
filters[].data_filters[].type	受信対象とするデータの型（チュートリアル 1 の設定に合わせる）
filters[].data_filters[].name	受信対象とするデータの名前（チュートリアル 1 の設定に合わせる）

注釈: ここで設定している filters は、iSCP において「ダウンストリームフィルター」と呼ばれるもので、サーバーから受信する対象のデータポイントを指定するものです。intdash Edge Agent 2 内でのデータの行先を変えるフィルター（[フィルタリング／サンプリング](#) (p. 70)）とは別の概念です。

5.3 リアルタイム受信を開始する

ダウンストリーム用エッジでの操作

以下のコマンドを実行します。

```
$ intdash-agentctl run
```

これによりストリーマーが起動され、/var/run/intdash/down-hello.fifo が作成されます。

また、intdash Edge Agent 2 と intdash サーバーとの接続が確立され、サーバーからのデータを待ち受ける状態になります。

5.4 データを確認する

アップストリーム用エッジでの操作

[チュートリアル 1](#) (p. 11) でセットアップしたアップストリーム用エッジで、intdash-agentctl run を実行し、チュートリアル 1 の手順のとおり echo コマンドでデータを書き込みます。データはリアルタイムにサーバーに送信されます。

ダウンストリーム用エッジでの操作

サーバーからデータが受信され、デバイスコネクター IPC の設定に従って /var/run/intdash/down-hello.fifo に書き込まれます。

確認のため、以下のように cat コマンドを使って、FIFO に書き込まれたデータを表示させます。

```
$ cat /var/run/intdash/down-hello.fifo
```

バイナリデータであるためヘッダー部分は認識できる文字にはなりませんが、文字列 Hello が表示されており、受信できていることが分かります。

ダウンストリーム用エッジでの操作

確認が済んだら、intdash-agentctl run を実行したターミナルで Ctrl+C を押して、intdash-agentctl run を

終了します。

アップストリーム用エッジでの操作

intdash-agentctl run を実行したターミナルで Ctrl+C を押して、intdash-agentctl run を終了します。

06 チュートリアル 3: 付属デバイスコネクタを使用する

このチュートリアルでは、付属デバイスコネクタ `device-connector-intdash` からデータを取得し、サーバーに送信します。

注釈: 付属デバイスコネクタは、intdash Edge Agent 2 をインストールした際に依存パッケージとしてインストールされています。インストールされていることを確認するには、以下のコマンドを実行してください。

```
device-connector-intdash --version
```

もし `device-connector-intdash` が見つからない場合は、`intdash-edge-agent` パッケージを再インストールしてください。

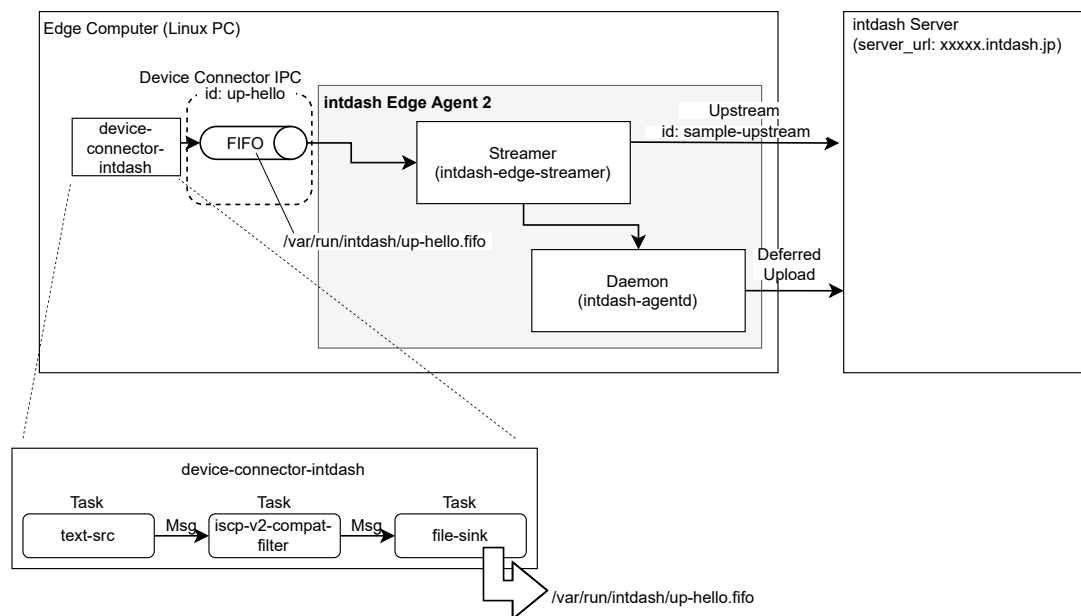


図 6 チュートリアル 3 (デバイスコネクタ以外はチュートリアル 1 と同じ)

6.1 準備するもの

このチュートリアルを実行するには以下が必要です。

- [チュートリアル 1](#) (p. 11) でセットアップ済みのアップストリーム用エッジ

注釈: このチュートリアルでは、エッジは 1 つしか使用しません。

6.2 パイプライン設定を作成する

device-connector-intdash では、データをデバイスから受け取り intdash Edge Agent 2 に渡すまでの流れを「パイプライン設定ファイル」で設定します。

ここでは、以下の内容のパイプライン設定ファイルを作成し、/tmp/dc-hello.yaml として保存してください。

```
tasks:
- id: 1
  element: text-src
  conf:
    text: "Hello from device connector!"
    interval_ms: 100

- id: 2
  element: iscp-v2-compat-filter
  from: [[ 1 ]]
  conf:
    timestamp:
      stamp:
        clock_id: CLOCK_MONOTONIC
    convert_rule:
      string:
        name: ab

- id: 3
  element: file-sink
  from: [[ 2 ]]
  conf:
    path: "/var/run/intdash/up-hello.fifo"
```

このパイプライン設定ファイルでは、以下が定義されています。

- 「Hello from device connector!」というテキストを 100 ミリ秒おきに生成する（タスク ID: 1）
- FIFO 用データフォーマットに変換し、タイムスタンプ、型、ID を与える（タスク ID: 2）
- iscp-v2-compat フォーマットで FIFO に書き出す（タスク ID: 3）

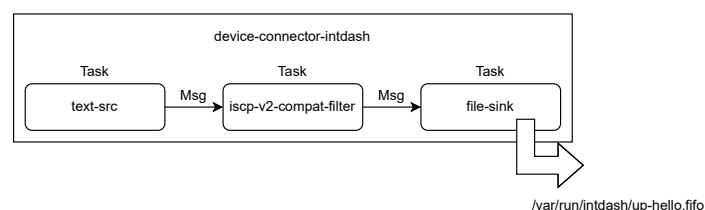


図 7 intdash-device-connector のパイプライン設定

注釈: パイプライン設定ファイルの詳細については [付属デバイスコネクタの使用](#) (p. 42) を参照してください。

注釈: intdash Edge Agent 2 がデバイスコネクタとの間でデータを受け渡す際に使用する FIFO のフォーマットとして、iscp-v2-compat フォーマットと、logger-msg フォーマットの 2 種類があります。device-connector-intdash は、iscp-v2-compat フォーマットで出力します。

6.3 ストリーマーを起動する

以下のコマンドを実行して、ストリーマーを起動します。

```
$ intdash-agentctl run
```

これにより、`/var/run/intdash/up-hello.fifo` への入力を待ち受けている状態になります。

次に、新しいターミナルを開いて、デバイスコネクターを起動します。

```
$ device-connector-intdash --config /tmp/dc-hello.yaml
```

これにより、`device-connector-intdash` は 100 ミリ秒ごとに `/var/run/intdash/up-hello.fifo` に文字列を書き込みます。intdash Edge Agent 2 はそれを受け取り、intdash サーバーに送信します。

注釈: ストリーマーが起動されたときに連動してデバイスコネクターを起動させることも可能です。[デバイスコネクター IPC](#) (p. 29) を参照してください。

6.4 Edge Finder で確認する

ウェブブラウザで Edge Finder を開き、使用しているエッジのトラフィック画面を表示します。データが送信されていることを確認します。



図 8 Edge Finder でトラフィックを確認する

確認が済んだら、ストリーマーとデバイスコネクターを起動したターミナルで、それぞれ `Ctrl+C` を押して終了します。

07 リアルタイム送受信の開始／終了

リアルタイムデータの送受信はストリーマーが行います。

ストリーマーを起動すると、以下が行われます。

- アップストリームが設定済みの場合は、ストリーマーはデバイスコネクターからデータを受け取り、intdash のリアルタイム API を使ってデータを intdash サーバーに送信します。
- ダウンストリームが設定済みの場合は、ストリーマーは intdash のリアルタイム API を使って intdash サーバーからデータを受信し、受け取ったデータをデバイスコネクターに渡します。

アップストリームの設定については、[アップストリームによる送信](#) (p. 28) を参照してください。ダウンストリームの設定については、[ダウンストリームによる受信](#) (p. 36) を参照してください。

7.1 ストリーマーを起動／終了する

ストリーマーを起動するには、以下のコマンドを実行します。

```
$ intdash-agentctl run
```

リアルタイムデータの送信開始から終了までのデータが、intdash における 1 つの「計測」になります。アップストリームを設定してストリーマーを開始すると、1 つの計測が開始されることになります。

注釈:

- intdash-agentctl コマンドを使って intdash Edge Agent 2 の操作を行うためには、intdash-agentd が起動している必要があります。インストールされた直後の設定では intdash-agentd は自動的に起動されます。手動で起動する方法については、[手動によるデモン起動／終了](#) (p. 25) を参照してください。
- 遅延アップロードは、ストリーマーの起動や終了とは関係なく intdash-agentd により行われます。

ストリーマーを終了するには、ストリーマーを起動したターミナルで Ctrl+C を押し、intdash-agentctl に SIGINT を送信します。

7.2 電源に連動させる

エッジデバイスの電源がオンになったら自動的に intdash-agentctl run が実行されてストリーマーが起動するように設定するには、以下のコマンドを実行します。

```
$ sudo update-rc.d intdash-agentctl-run defaults
```

電源への連動の設定を解除する場合は、以下のコマンドを実行します。

```
$ sudo update-rc.d intdash-agentctl-run remove
```


08 手動によるデーモン起動／終了

重要: 通常の方法でインストールした場合、エッジデバイスの起動と同時にデーモンが起動される設定になっているため、デーモンの起動や終了を手動で行う必要はありません。デーモンの自動起動が行われない環境（Docker Hub で公開されている Ubuntu など）でデーモンを手動で起動／停止したい場合にこちらの手順を使用します。

デーモン（intdash-agentd）を起動するには、以下のように init スクリプトを実行します。

```
$ /etc/init.d/intdash-agentd start
```

デーモンを終了するには、以下のように init スクリプトを実行します。

```
$ /etc/init.d/intdash-agentd stop
```

09 接続先サーバーと認証情報

intdash Edge Agent 2 が intdash サーバーに接続するには、以下の情報が必要です。

- 接続先の intdash サーバーのホスト名（例：xxxxx.intdash.jp）
- プロジェクトの UUID（省略した場合は、Global Project が使用されます）
- このエッジデバイスに割り当てる、intdash のエッジ UUID
- 割り当てられたエッジのクライアントシークレット

注釈: プロジェクトの UUID は、Project Console で使用したいプロジェクトを開いた状態で、左上の Copy UUID をクリックすることにより得ることができます。

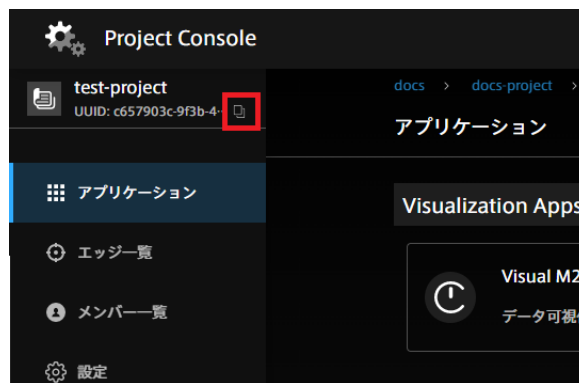


図 9 プロジェクト UUID のコピー

intdash Edge Agent 2 で接続に関する設定を確認するには、以下のコマンドを実行します。

```
$ intdash-agentctl config connection --get
```

設定を変更するには、以下のコマンドを実行します。

```
$ intdash-agentctl config connection --modify <config_in_yaml_format>
```

例: 接続先サーバーと認証情報を設定する。

```
$ intdash-agentctl config connection --modify '
  server_url: https://xxxxxx.intdash.jp
  edge_uuid: 03ace3b1-d208-4fc3-xxxx-xxxxxxxxxxxx
  client_secret: f10MTtnsNpE3LAh0arkUqgMyw.....NgPJFRk
'
```

config_in_yaml_format で設定できるパラメーターは以下のとおりです。

キー	型	説明
server_url	string	接続先 intdash サーバーの URL
project_uuid	string	プロジェクト UUID
edge_uuid	string	intdash サーバーに接続する際に使用する、このエッジの UUID
client_secret	string	intdash サーバーに接続する際に使用するクライアントシークレット

intdash-agentctl config connection コマンドの使用方法については、[connection サブコマンド](#) (p. 88) を参照してください。

10 トランスポートの設定

iSCP によるリアルタイム送受信で使用するトランスポートプロトコルを設定します。QUIC または WebSocket のいずれかを選択することができます。（ここで設定するのは、iSCP によるリアルタイム送受信用に使われるトランスポートプロトコルです。遅延アップロードは常に HTTP による REST API で行われます。）

QUIC と WebSocket はいずれも到達保証と順序保証のあるプロトコルです。ただし、QUIC を選択し、ストリームの設定で `qos: unreliable` を指定すると、到達保証と順序保証のないトランスポート（QUIC DATAGRAM）が使用されます。そのため、QUIC と WebSocket はそれぞれ以下のような場合に向いています。

- QUIC: 帯域が十分でない環境で、欠損があってもよいので回線詰まりによる遅延が発生しない方法でデータを確認したい
- WebSocket: 帯域が十分でない環境で、回線詰まりによる遅延があってもよいので欠損なくデータを確認したい

例えば、リアルタイム性を重視して動画を送信したいときには、QUIC を選択した上で、動画内のサブフレームを送信するのに `qos: unreliable` のストリームを使用すると効果的です。サブフレームが欠損した場合、画質は悪化しますが、再生は継続できます。

重要: QUIC を使用するためにはサーバー側で設定が必要です。

現在の設定を確認するには、以下のコマンドを実行します。

```
$ intdash-agentctl config transport --get
```

設定を変更するには、以下のコマンドを実行します。

```
$ intdash-agentctl config transport --modify <config_in_yaml_format>
```

例: QUIC を使用するように設定する

```
$ intdash-agentctl config transport --modify 'protocol: quic'
```

`config_in_yaml_format` で設定できるパラメーターは以下のとおりです。

キー	型	説明
protocol	string	<ul style="list-style-type: none">• quic: QUIC による接続• websocket: WebSocket による接続

11 アップストリームによる送信

intdash サーバーにリアルタイムデータを送信する経路（ストリーム）をアップストリームと呼びます。アップストリームによる送信を行うには以下の準備が必要です。本章ではこれらについて説明します。[接続先サーバーと認証情報](#) (p. 26) は設定済みとします。

- アップストリームの設定を作成する
- デバイスコネクター IPC の設定を作成する
- (必要な場合のみ) フィルターを設定する（フィルターを使用すると、条件に一致したデータポイントの行先のアップストリームを変更することができます）

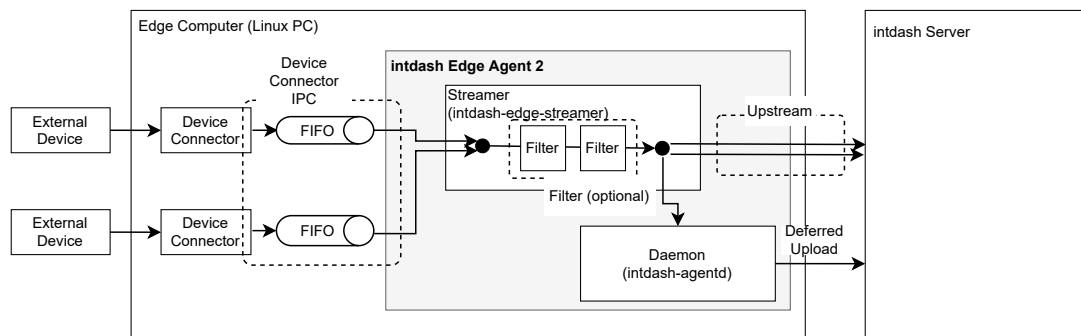


図 10 リアルタイムデータをサーバーに送信する

以上の設定を行ったうえで `intdash-agentctl run` コマンドを実行すると、ストリーマーが起動され、サーバーへのリアルタイム送信が開始されます。ストリーマーの起動から終了までの一連の時系列データが 1 つの計測となります。

送信中に通信が切断された場合や帯域が不十分だった場合、送信できなかったデータはローカルストレージに蓄積され、通信が回復したときに REST API で送信されます。これを遅延アップロードと呼びます。

11.1 アップストリーム

intdash Edge Agent 2 では、設定の異なる複数のアップストリームを設定し、同時に使用することができます。センサーデバイスからデバイスコネクターを使って取得されたデータは、指定されたアップストリームを使ってサーバーに送信されます。

各アップストリームについては、通信が切断された場合に遅延アップロードをするか、信頼性のある接続を使用するか、サーバーでのデータ永続化するか、などの設定が可能です。

アップストリームの設定は ID により管理されます。デバイスコネクター IPC 設定やフィルター設定において、行先のストリームを指定する際には、この ID で指定します。

アップストリームを作成するには以下のコマンドを実行します。

```
$ intdash-agentctl config upstream --create <config_in_yaml_format>
```

例:

```
$ intdash-agentctl config upstream --create '
  id: recoverable
  enabled: true
  recover: true
  persist: true'
```

(次のページに続く)

(前のページからの続き)

```
qos: unreliable
flush_policy: interval
flush_interval: 5
,
```

作成済みのアップストリームの設定を変更するには、ID と、変更したい設定を与えます。

```
$ intdash-agentctl config upstream --modify <id> <config_in_yaml_format>
```

設定できるパラメーターについては、以下を参照してください。

キー	型	説明
id	string	アップストリームを識別するための文字列。deferred は予約されているため使用できません。
enabled	bool	アップストリームの有効 (true) / 無効 (false)
recover	bool	送信ができなかった場合の遅延アップロードの有効 (true) / 無効 (false)。true にすると、リアルタイム送信ができなかった場合に、遅延アップロードが行われます。ただし、persist が false の場合、遅延アップロードは行われません。
persist	bool	サーバーでのデータ永続化の有効 (true) / 無効 (false)
qos	string	partial: iSCP の QoS として PARTIAL を指定します。 unreliable: iSCP の QoS として UNRELIABLE を指定します。
flush_policy	string	interval - リアルタイム送信を一定周期でフラッシュします。 immediately - リアルタイム送信をデータポイントごとにフラッシュします。
flush_interval	string	リアルタイム送信をフラッシュする間隔 (ミリ秒)。flush_policy が interval の場合のみ使用されます。

注釈: アップストリーム設定の一覧表示、削除などのコマンドについては、[intdash-agentctl config upstream/downstream](#) (p. 90) を参照してください。

11.2 デバイスコネクター IPC

intdash Edge Agent 2 では、複数のデバイスコネクターを設定し、同時に使用することができます。センサーデバイスからデバイスコネクターを使って取得されたデータは、指定されたアップストリームを使ってサーバーに送信されます。

デバイスコネクターから intdash Edge Agent 2 にデータを渡すための設定は、デバイスコネクター IPC 設定で行います。

デバイスコネクター IPC 設定では、intdash Edge Agent 2 との通信に使用する FIFO のパス、データの行先のアップストリーム、データ名称などを設定します。デバイスコネクター IPC 設定は ID により管理されます。

アップストリーム用のデバイスコネクター IPC 設定を作成するには以下のコマンドを実行します。

```
$ intdash-agentctl config device-connector upstream --create <config_in_yaml_format>
```

例:

```
$ intdash-agentctl config device-connector upstream --create '
  id: up
  data_name_prefix: v1/1
  dest_ids:
    - recoverable
  format: iscp-v2-compatible
  ipc:
    type: fifo
    path: /var/run/intdash/uplink-hello1.fifo
  launch:
    cmd: device-connector-intdash
    args:
      - --config
      - /home/agent-volume-a/dc-hello-world-1.yaml
  ,
```

作成済みのデバイスコネクター IPC 設定を変更するには、ID と、変更したい設定を与えます。

```
$ intdash-agentctl config device-connector upstream --modify <id> <config_in_yaml_format>
```

config_in_yaml_format で設定できるパラメーターは以下のとおりです。

キー	型	説明
id	string	デバイスコネクタ IPC 設定の識別子
enabled	bool	デバイスコネクタ IPC 設定の有効 (true) / 無効 (false)
data_name_prefix	string	iSCP のデータ名称のプリフィックスとして使用する文字列。受信側で iSCP v1 を使用する場合は、決められたルールに沿って data_name_prefix を与える必要があります。詳細は 付与されるデータ ID (p. 106) を参照してください。
dest_ids	[string]	データは、ここで指定した ID のアップストリームを使ってサーバーに送られます。複数の ID を指定した場合、データポイントは複製され、それぞれのアップストリームからサーバーに送られます。ただし、フィルターを設定すると、行先のアップストリームを変更することができます。 フィルター (必要な場合のみ) (p. 32) を参照してください。
format	string	デバイスコネクタとの通信に使用されるデータフォーマット iscp-v2-compatible : iSCPv2 と同等の FIFO データフォーマット logger-msg : 旧 intdash Edge Agent の FIFO データフォーマット
ipc.type	string	intdash Edge Agent 2 とデバイスコネクタのデータ通信方法 (現在 fifo のみに対応しているため、fifo を指定してください。)
ipc.path	string	intdash Edge Agent 2 がデバイスコネクタからデータを受信する FIFO のパス
launch.cmd	string	intdash Edge Agent 2 が起動したらそれに連動してデバイスコネクタも起動させたい場合は、デバイスコネクタの実行ファイルのパスを指定します。付属デバイスコネクタを使用する場合は、device-connector-intdash とします。 この設定を使ってデバイスコネクタを起動した場合、intdash Edge Agent 2 を終了すると、デバイスコネクタも終了します。
launch.args	[string]	デバイスコネクタ起動時の引数。付属デバイスコネクタを使用するために launch.cmd を device-connector-intdash とした場合は、第 1 引数を --config、第 2 引数をパイプライン設定ファイルのパスとします。 - --config - <path-to-pipeline-configuration.yaml>
launch.environment	[string]	デバイスコネクタ起動時に追加する環境変数 (VARIABLE=VALUE の形式で記載します)

重要: 付属デバイスコネクタを使用する場合は、センサーデバイスからデータを取得し、所定のフォーマットに従って FIFO にデータを書き込むまでの流れを定義したパイプライン設定ファイルが必要です。パイプライン設定ファイルについては、[付属デバイスコネクタの使用](#) (p. 42) を参照してください。

注釈: デバイスコネクター IPC 設定の一覧表示、削除などのコマンドについては、[intdash-agentctl config device-connector](#) (p. 92) を参照してください。

11.3 遅延アップロード用の行先 (deferred)

「persist: true」かつ「recover: true」のアップストリームでは、リアルタイム送信ができなかった場合は遅延アップロードが行われます。しかし、リアルタイム送信を試みることなく最初から遅延アップロードした場合は、疑似的な行先ストリームとして deferred を指定します。

deferred は設定のための名称で、実際には deferred というストリームが intdash Edge Agent 2 とサーバーとの間に作成されるわけではありません。

注釈: deferred というストリーム ID は予約されており、ユーザーが作成するストリームにこの名前を付けることはできません。また、intdash-agentctl config device-connector upstream で deferred の設定を変更したり削除したりすることはできません。

遅延アップロードの優先度や、遅延アップロード用のデータに使用するストレージに関しては、[遅延アップロード](#) (p. 79) を参照してください。

11.4 フィルター (必要な場合のみ)

どのアップストリームを使ってサーバーにデータを送信するか (行先のアップストリーム) はデバイスコネクター IPC 設定で指定しましたが、デバイスコネクター IPC とアップストリームの間にフィルターを挟むことにより、データの行先を他のストリームに変更することができます。

フィルターには条件を設定します。条件に一致するデータだけを別のアップストリームに送ることができます。これにより、一部のデータだけをリアルタイム送信し、その他のデータはリアルタイム送信せずに遅延アップロードするといった設定が可能です。

フィルターの設定方法については、[フィルタリング/サンプリング](#) (p. 70) を参照してください。

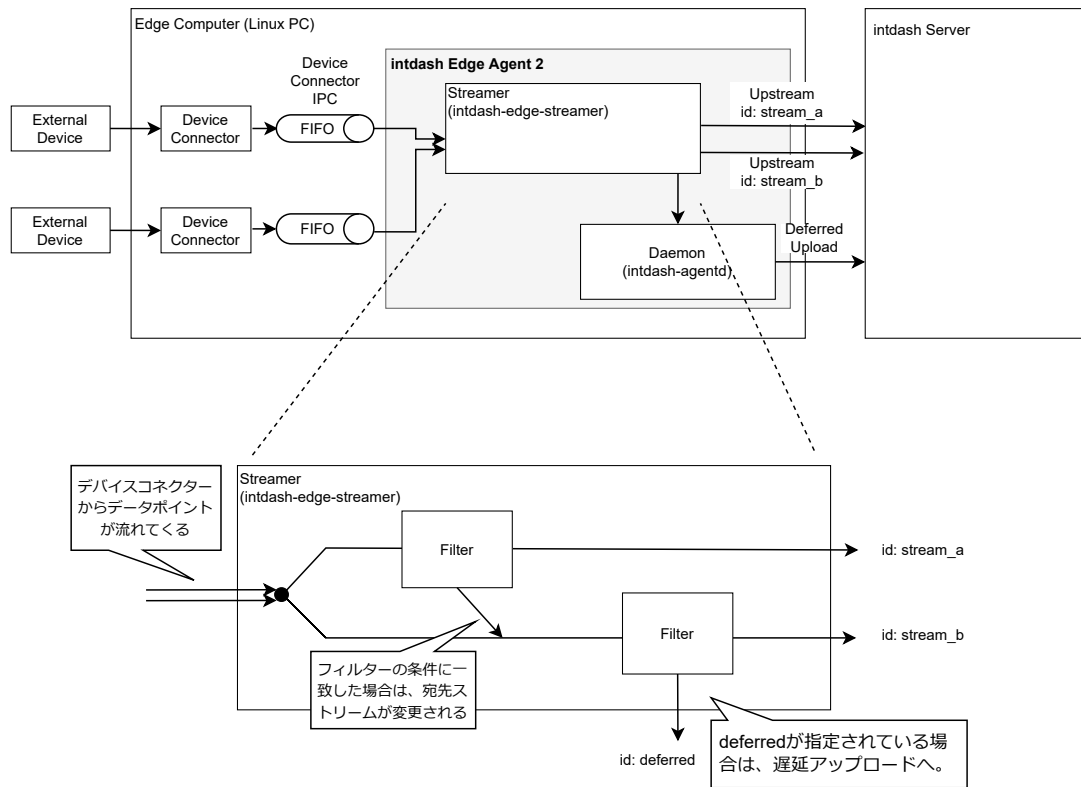


図 11 フィルター（アップストリームの場合）

注釈: デバイスコネクター IPC 設定やフィルター設定では、存在しないストリーム ID を指定することができます。

最終的に、存在しない ID を行先としているデータポイントは破棄されるため、存在しないストリーム ID を一時的な行先として使用すると便利な場合があります。以下の例では、デバイスコネクター IPC 設定の `dest_ids` において、存在しないアップストリーム `stream_x` を行先として指定しています。そのうえで、フィルタリングの段階で、フィルター条件に一致したデータポイントの行先を変更しています。フィルタリングの段階を完了してもなお `stream_x` を行先としているデータポイントは、破棄されます。

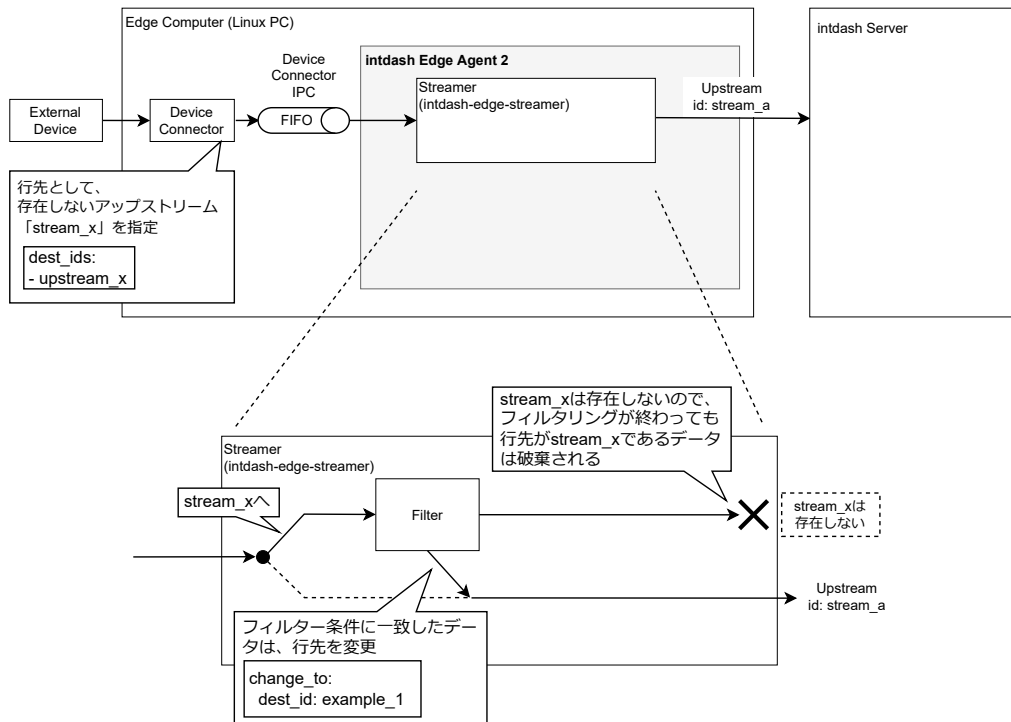


図 12 存在しないアップストリーム ID の使用

11.5 送信の開始／終了

アップストリームの設定とデバイスコネクター IPC の設定を行ったうえで、`intdash-agentctl run` を実行すると、intdash サーバーへの接続が行われます。

この状態で、デバイスコネクターから FIFO にデータポイントを書き込むことで、そのデータポイントは、intdash Edge Agent 2 から intdash サーバーに送信されます。

リアルタイムデータの送受信を終了する場合は、`intdash-agentctl run` を実行したターミナルで `Ctrl+C` を押してください。

「`recover: true`」のアップストリームにおいてデータの欠損が発生した場合や、行先ストリームとして `deferred` が指定されていた場合は、遅延アップロードが行われます。リアルタイムデータの送受信を終了しても、intdash-agentd が起動している間は、遅延アップロードが継続されます。

11.6 基準時刻について

intdash において、基準時刻とは、計測を開始した時刻を指します。intdash Edge Agent 2 が送信する基準時刻には以下の 2 つがあります。

- EdgeRTC による基準時刻
 - intdash Edge Agent 2 を実行しているシステムの時計による基準時刻です。計測開始時は必ず EdgeRTC で基準時刻が作成されます。その後、基準時刻は一定間隔でサーバーに送信されます。
- NTP による基準時刻
 - NTP サーバーから取得した時計による基準時刻です。NTP サーバーとの通信ができれば NTP 基準時刻が作成されます。その後、基準時刻は一定間隔でサーバーに送信されます。

12 ダウンストリームによる受信

アップストリームと反対に、intdash サーバーからのリアルタイムデータを受信する経路をダウンストリームと呼びます。ダウンストリームによる受信を行うには以下の準備が必要です。([接続先サーバーと認証情報](#) (p. 26) は設定済みとします。)

- デバイスコネクター IPC の設定を作成する
- ダウンストリームの設定を作成する
- (必要な場合のみ) フィルターを設定する (フィルターを使用すると、条件に一致したデータポイントの行先のデバイスコネクターを変更することができます)

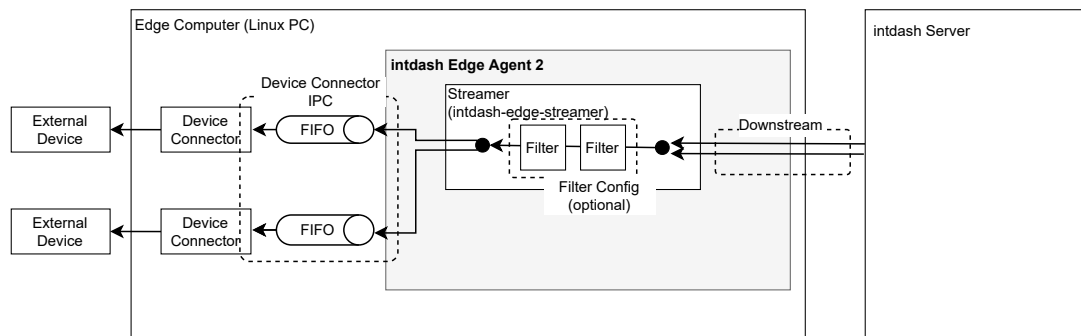


図 13 リアルタイムデータをサーバーから受信する

以上の設定を行ったうえで、`intdash-agentctl run` コマンドを実行することによりサーバーからのリアルタイム受信が開始されます。

注釈: ダウンストリームの場合、データは、ストリームからデバイスコネクター IPC (さらにデバイスコネクターへ) という方向で流れますが、設定しやすさの観点からデバイスコネクター IPC を先に説明します。

12.1 デバイスコネクター IPC

intdash Edge Agent 2 では、複数のデバイスコネクターを設定し、同時に使用することができます。ダウンストリームにより intdash Edge Agent 2 が受信したデータは、デバイスコネクターを使って外部デバイス (アクチュエーターなど) に送られます。

デバイスコネクターにデータを渡すための設定は、デバイスコネクター IPC 設定で行います。デバイスコネクター IPC 設定では、intdash Edge Agent 2 との通信に使用する FIFO のパスなどを設定します。

デバイスコネクター IPC 設定の作成時には ID を付与します。後ほど、どのデバイスコネクターにデータを流すかを指定する際にこの ID を使用します。

ダウンストリーム用のデバイスコネクター IPC 設定を作成するには以下のコマンドを実行します。

```
$ intdash-agentctl config device-connector downstream --create <config_in_yaml_format>
```

例:

```
$ intdash-agentctl config device-connector downstream --create '  
  id: cat  
  enabled: true  
  format: iscp-v2-compatible  
  ipc:
```

(次のページに続く)

(前のページからの続き)

```
type: fifo
path: /var/run/intdash/downlink-hello.fifo
,
```

作成済みのデバイスコネクタ IPC 設定を変更するには、ID と、変更したい設定を与えます。

```
$ intdash-agentctl config device-connector downstream --modify <id> <config_in_yaml_format>
```

config_in_yaml_format で設定できるパラメーターは以下のとおりです。

キー	型	説明
id	string	デバイスコネクタ IPC 設定の識別子
enabled	bool	デバイスコネクタ IPC 設定の有効 (true) / 無効 (false)
format	string	デバイスコネクタが受信するデータのフォーマット iscpv2-compatible : iSCPV2 と同等の FIFO データフォーマット logger-msg : 旧 intdash Edge Agent の FIFO データフォーマット
ipc.type	string	intdash Edge Agent 2 とデバイスコネクタのデータ通信方法 (現在 fifo のみに対応しているため、fifo を指定してください。)
ipc.path	string	デバイスコネクタが intdash Edge Agent 2 からデータを受信する FIFO のパス
launch.cmd	string	intdash Edge Agent 2 が起動したらそれに連動してデバイスコネクタも起動させたい場合は、デバイスコネクタの実行ファイルのパスを指定します。付属デバイスコネクタを使用する場合は、device-connector-intdash とします。
launch.args	[string]	デバイスコネクタ起動時の引数。付属デバイスコネクタを使用するために launch.cmd を device-connector-intdash とした場合は、第 1 引数を --config、第 2 引数をパイプライン設定ファイルのパスとします。 - --config - <path-to-pipeline-configuration.yaml>
launch.environment	[string]	デバイスコネクタ起動時に追加する環境変数 (VARIABLE=VALUE の形式で記載します)

重要: 付属デバイスコネクタを使用する場合は、intdash Edge Agent 2 からデータを取得し、外部デバイスに渡すまでの流れを定義したパイプライン設定ファイルが必要です。
パイプライン設定ファイルについては、[付属デバイスコネクタの使用](#) (p. 42) を参照してください。

注釈: デバイスコネクタ IPC 設定の一覧表示、削除などのコマンドについては、[device-connector コマンド](#) (p. 92) を参照してください。

12.2 ダウンストリーム

intdash Edge Agent 2 では、複数のダウンストリームを設定し、同時に使用することができます。

ダウンストリームの設定では、どのエッジからのどのデータ ID を持つデータを受信するかを設定します。ストリームの設定は ID により管理されます。

ダウンストリームを作成するには以下のコマンドを実行します。

```
$ intdash-agentctl config downstream --create <config_in_yaml_format>
```

例:

```
intdash-agentctl config downstream --create '  
  id: down  
  enabled: true  
  dest_ids:  
    - cat  
  qos: unreliable  
  filters:  
    - src_edge_uuid: f5e7a9d2-099b-432f-86e4-9b496af27d3b  
      data_filters:  
        - type: string  
          name: v1/1/a  
,
```

作成済みのダウンストリームの設定を変更するには、ID と、変更したい設定を与えます。

```
$ intdash-agentctl config downstream --modify <id> <config_in_yaml_format>
```

config_in_yaml_format で設定できるパラメーターは以下のとおりです。

キー	型	説明
id	string	ストリーム設定を識別するための文字列。deferred は予約されているため使用できません。
enabled	bool	iSCP のダウンストリームの有効 (true) / 無効 (false)
dest_ids	[string]	データは、ここで指定された ID のデバイスコネクター IPC を使って外部デバイスに送られます。複数の ID を指定した場合、データポイントは複製され、それぞれのデバイスコネクター IPC に送られます。ただし、フィルターを設定すると、行先のデバイスコネクター IPC を変更することができます。 フィルター (必要な場合のみ) (p. 39) を参照してください。
qos	string	partial: iSCP の QoS として PARTIAL を指定します。 unreliable: iSCP の QoS として UNRELIABLE を指定します。
filters	[object]	ダウンストリーム対象とするデータを定義した設定のリスト
filters[].src_edge_uuid	string	ダウンストリーム対象とする送信元のエッジ UUID。この UUID のエッジから送信されたデータを受信します。
filters[].data_filters	[object]	ダウンストリーム対象とするデータ ID を定義した設定のリスト。ここで指定したデータ ID (データ型とデータ名称) に一致したデータを受信します。
filters[].data_filters[].type	[object]	ダウンストリーム対象とするデータ型
filters[].data_filters[].name	[object]	ダウンストリーム対象とするデータ名称 (#, + を利用したワイルドカード指定が可能。ワイルドカードについては iSCP の仕様を参照してください。)

注釈: ダウンストリーム設定の一覧表示、削除などのコマンドについては、[upstream](#)、[downstream サブコマンド](#) (p. 90) を参照してください。

注釈: 上記の `filters[].data_filters` は、どのデータをサーバーからダウンストリームするかを設定するもの (ダウンストリームフィルター) です。intdash Edge Agent 2 が受信したデータについて、その行先を変更する [フィルター](#) (p. 39) とは別の機能です。

12.3 フィルター (必要な場合のみ)

どのデバイスコネクター IPC にデータを送信するかはストリームの設定で指定しましたが、ストリームとデバイスコネクター IPC の間にフィルターを挟むことにより、データの行先を他のデバイスコネクターに変更することができます。

フィルターには条件を設定し、その条件に一致するデータだけを別のデバイスコネクター IPC に送ることができます。これにより、一部のデータだけをデバイスコネクター A に送信し、その他のデータはデバイスコネクター B に送信するといった設定が可能です。

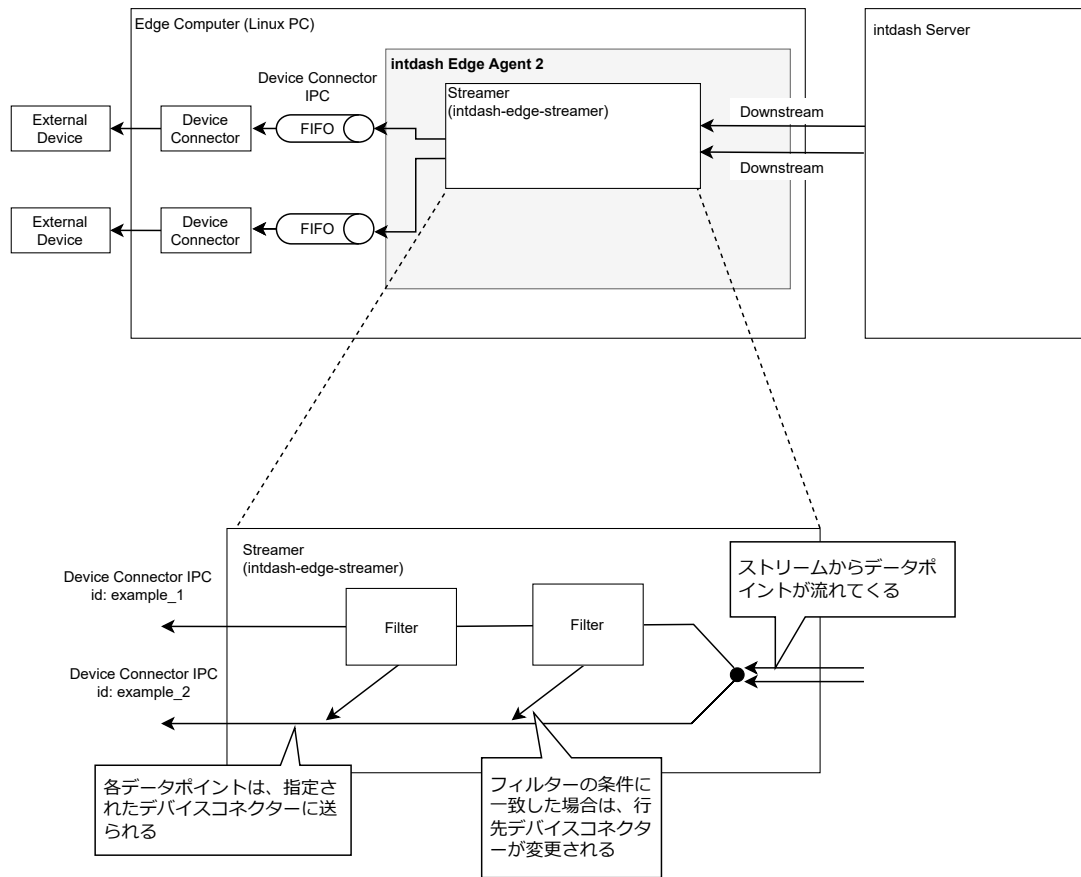


図 14 フィルター（ダウンストリームの場合）

フィルターの設定方法については、[フィルタリング／サンプリング](#) (p. 70) を参照してください。

注釈: ダウンストリームの設定やフィルターの設定では、存在しないデバイスコネクタ IPC の ID を指定することができます。最終的に、存在しない ID を行先としているデータポイントは破棄されるため、存在しないデバイスコネクタ IPC の ID を一時的な行先として使用すると便利な場合があります。

以下の例では、ダウンストリーム設定の `dest_ids` において、存在しないデバイスコネクタ IPC `dc_x` を行先として指定しています。そのうえで、フィルタリングの段階で、フィルター条件に一致したデータポイントの行先を変更しています。フィルタリングの段階を完了してもなお `dc_x` を行先としているデータポイントは、破棄されます。

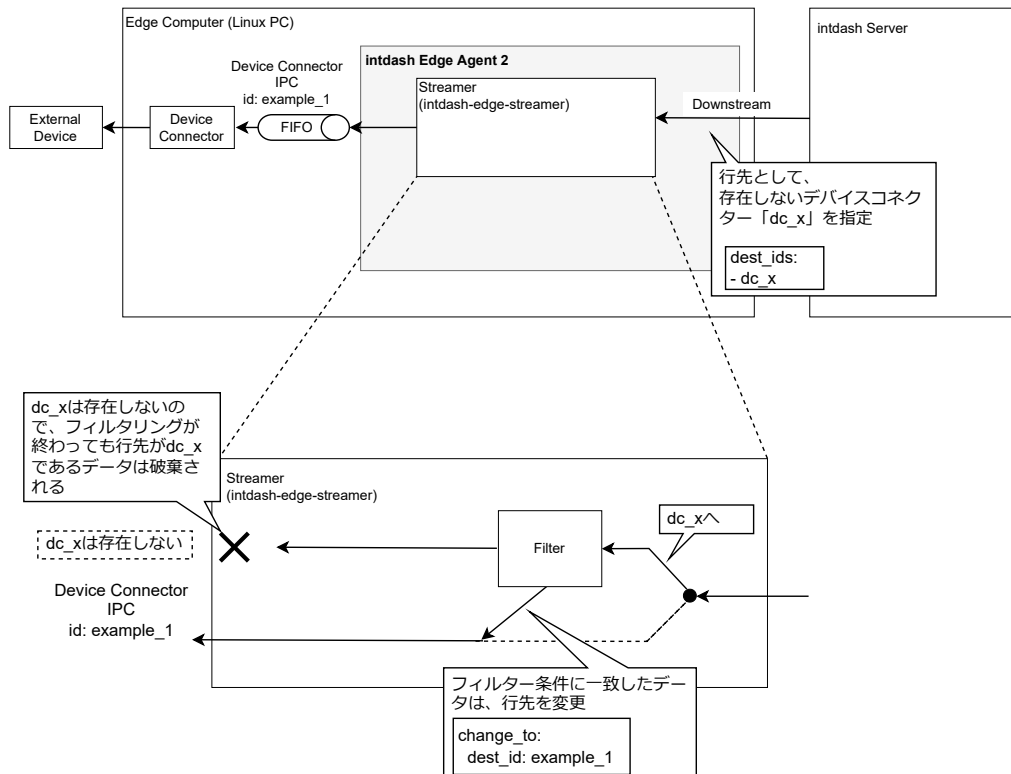


図 15 存在しないデバイスコネクタ IPC の ID を使用

12.4 受信を開始／終了

ダウンストリームの設定とデバイスコネクタ IPC の設定を行ったうえで、`intdash-agentctl run` を実行すると、intdash サーバーへの接続が行われます。

別のエッジからサーバーにデータがリアルタイム送信されているとき、そのデータがこのエッジにとってのダウンストリーム対象であればこのエッジで受信されます。

リアルタイムデータの送受信を終了する場合は、`intdash-agentctl run` を実行したターミナルで `Ctrl+C` を押してください。

13 付属デバイスコネクタの使用

intdash Edge Agent 2 をインストールすると、アプトポッド製のデバイスコネクタである device-connector-intdash が、依存パッケージとしてインストールされます。

device-connector-intdash は、intdash 用のデータを扱うのに便利な機能を集めたデバイスコネクタです。device-connector-intdash を使用すると、外部デバイスからのデータを、intdash Edge Agent 2 に渡すための FIFO 用データフォーマット ([iscp-v2-compat](#) (p. 110)) に簡単に変換することができます。また、EDGEPLANT CAN-USB Interface や、EDGEPLANT ANALOG-USB Interface からのデータを簡単に扱うことができます。

注釈: device-connector-intdash は、アプトポッド製のデバイスコネクタ作成用フレームワークである、Device Connector Framework を使用して開発されています。

Device Connector Framework の詳細や、Device Connector Framework を使ったデバイスコネクタの開発方法については、「intdash Edge Agent 用デバイスコネクタデベロッパーガイド」を参照してください。

13.1 パイプラインとエレメント

device-connector-intdash はパイプラインアーキテクチャを採用しています。これにより、外部デバイスからデータを取得する処理、データを加工する処理、書き出す処理など、さまざまな処理を組み合わせで実行することができます。この一連のデータの処理の流れをパイプラインと呼びます。パイプライン内で行われる処理は、エレメントと呼ばれる部品を使って表現します。また、パイプラインを流れるデータはメッセージと呼ばれます。

タスク間で送受信されるメッセージは型を持ちます。型は MIME タイプを表す文字列またはカスタムタイプを表す文字列で指定されます。各ポートがどの型のメッセージを受け取れるかは、エレメントにおいて定義されています。

エレメントに関連する主要な概念を以下に挙げます。

エレメント

エレメントは、メッセージの生成やメッセージの処理を定義したものです。デバイスコネクタ内にメッセージを生成する「src エレメント」、受け取るだけの「sink エレメント」、送受信を共に行う「filter エレメント」があります。

1 つのパイプラインの定義は、src エレメントから始まり、sink エレメントで終わります。必要に応じて途中で 1 個以上の filter エレメントを挟むことができます。

タスク

パイプラインの定義はエレメントの組み合わせで行いますが、各エレメントによる処理は実行時には「タスク」と呼ばれます。

メッセージは、各タスクの「ポート」を使って送受信されます。タスクによっては複数のポートで送受信ができます。使用可能なポートの個数は各タスクの元になったエレメントにおいて決まっています。

メッセージ

タスク間で送受信されるデータです。

タスク間で送受信されるメッセージは、何らかの型を持ちます。型情報には、MIME タイプ (例: MIME タイプ "image/jpeg") またはカスタムタイプ (例: カスタムタイプ "iscp-v2-compat-msg") を使用します。

各ポートがどの型のメッセージを受け取れるかは、エレメントにおいて定義されます。

送信元のタスクは、メッセージを最初に送信するとき、そのメッセージの型をデバイスコネクタに通知

します。デバイスコネクタは、メッセージを受信するポートがその型を受け入れ可能かどうかを判定し、受け入れができない場合はエラーとします。

データを "image/jpeg" や "text/plain" のような MIME タイプ、または、"iscp-v2-compat-msg" のようなカスタムタイプで送出するエレメントの場合は、1 枚の画像や 1 まとまりのテキスト、1 つのデータポイントのように、意味を持つ単位を 1 つのメッセージとして送出します。

MIME タイプに "application/octet-stream" を使用すると、既存の MIME タイプで定義されていない任意のバイナリ形式を表すことができます。また、この MIME タイプを使用することで、例えば延々と続くバイナリストリームをセグメントに区切って受け渡すような使い方も可能です。

src エレメント

src エレメントは、デバイスコネクタ内のパイプラインの始点です。src エレメントによって定義されたタスクを src タスクと呼びます。src タスクは、デバイスコネクタの外部（外部デバイスまたは intdash Edge Agent 2）からデータを受け取ってデバイスコネクタ内にデータを生成し、そのデータをメッセージとして次のタスクに渡します。次のタスクにメッセージを渡すための送信用ポートを 1 つ持っています。

sink エレメント

sink エレメントは、デバイスコネクタ内のパイプラインの終点です。sink エレメントによって定義されたタスクを sink タスクと呼びます。sink タスクは、前のタスクからメッセージを受け取り、デバイスコネクタの外部（外部デバイスまたは intdash Edge Agent 2）にデータを渡します。前のタスクからメッセージを受け取るための受信用ポートを 1 つ以上持っています。

filter エレメント

filter エレメントは 2 つのエレメントの間に挟むことができるエレメントです。filter エレメントによって定義されたタスクを filter タスクと呼びます。filter タスクは、前のタスクからメッセージを受け取り、何らかの処理を行って、次のタスクにメッセージを渡します。前のタスクからメッセージを受け取るための受信用ポートを 1 つ以上持ち、次のタスクにメッセージを渡すための送信用ポートを 1 つ持っています。

デバイスコネクタの filter エレメント（filter タスク）は、[フィルタリング／サンプリング](#) (p. 70) のフィルタリング機能とは別の概念です。

注釈: データは、sink エレメントを使って intdash Edge Agent 2 にデータを渡す前までに、intdash Edge Agent 2 の FIFO 用データフォーマットにしておく必要があります。

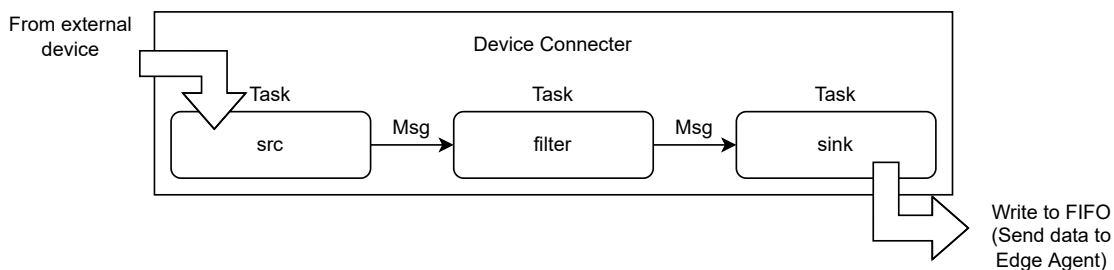


図 16 デバイスから intdash Edge Agent 2 へ入力するパイプラインの例

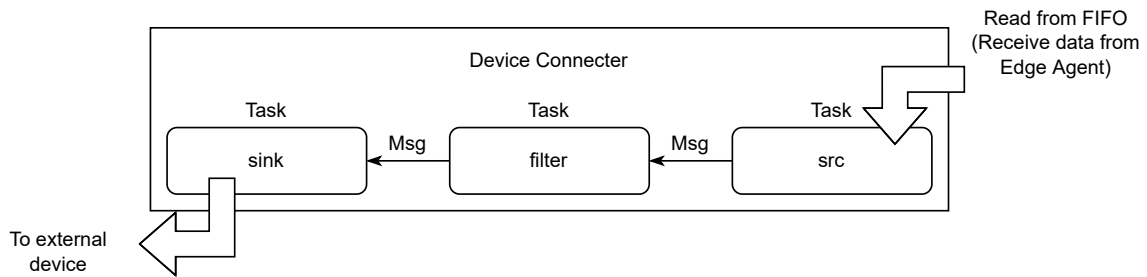


図 17 intdash Edge Agent 2 からデバイスへ出力するパイプラインの例

注釈: ここで説明しているパイプラインアーキテクチャは、Device Connector Framework が持っている仕組みです。従って、エレメントやパイプラインについての説明は、device-connector-intdash に限らず、Device Connector Framework を使って作られたデバイスコネクタに共通です。

13.2 ストリームの方向とパイプライン

アップストリームを行う場合のデバイスコネクタも、ダウンストリームを行う場合のデバイスコネクタも、基本的な設定方法は同じです。いずれも、src エレメントから、必要に応じて filter エレメントを挟み、sink エレメントまでのパイプラインとして作成します。

アップストリームのためのデバイスコネクタの場合は、「外部デバイスからデータを取得する src」→「必要に応じた filter」→「intdash Edge Agent 2 にデータを渡すための FIFO に書き込む sink (file-sink)」というパイプラインになります。

反対に、ダウンストリームのためのデバイスコネクタの場合は、「intdash Edge Agent 2 からデータを受け取るために FIFO から読み出す src (file-src)」→「必要に応じた filter」→「外部デバイスにデータを渡す sink」というパイプラインになります。

重要:

- アップストリーム方向のデバイスコネクタの場合、1 つのパイプライン設定ファイルでは、1 つの sink エレメント (intdash Edge Agent 2 にデータを渡すための FIFO) を終点とするパイプラインを設定します。
- ダウンストリーム方向のデバイスコネクタの場合、1 つのパイプライン設定ファイルでは、1 つの src エレメント (intdash Edge Agent 2 からデータを受け取る FIFO) を起点とするパイプラインを設定します。

13.3 パイプライン設定ファイル

パイプラインの定義は、YAML 形式のパイプライン設定ファイルで行います。例えば、以下のようなパイプラインを考えることができます。

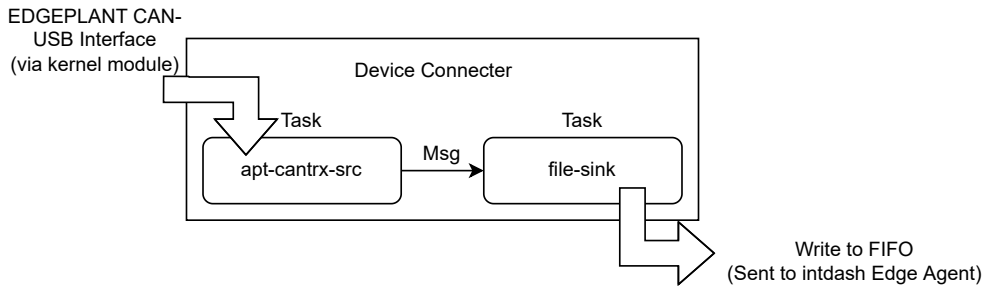


図 18 パイプラインの例

この例は、apt-cantrx-src タスクが EDGEPLANT CAN-USB Interface からデータを取得し、そのデータをメッセージとして file-sink タスクに送信しています。そして、file-sink タスクはメッセージを FIFO に書き出しています。

これをパイプライン設定ファイルで表現すると以下ようになります。タスクの設定は、tasks: 内に並列に記述します。

```
tasks:
- id: 1
  element: apt-cantrx-src
  conf:
    path: /dev/apt-usb/by-id/usb-xxx
    baudrate: 500
    clock_id: CLOCK_MONOTONIC

- id: 2
  element: file-sink
  from: [[ 1 ]]
  conf:
    path: "/var/run/intdash/can.fifo"
```

注釈: ここでは、パイプライン設定ファイルのうち、最も重要な tasks についてのみ説明します。tasks 以外の設定については、「intdash Edge Agent 用デバイスコネクタデベロッパーガイド」を参照してください。

項目	設定値	説明
id	整数	タスクに割り当てる ID（タスク ID）です。これはこのパイプライン上で一意でなくてはなりません。
element	文字列	このタスクに割り当てるエレメントの名称です。
from	文字列の二次元配列	<p>このタスクが受け取るメッセージの送信元を指定します。</p> <p>[["<送信元タスク ID>:<送信元ポート番号>"], ...] メッセージは非同期に送られます。</p> <p>例えば、[["1:0"], ["2:1"]] は、以下を意味します。</p> <ul style="list-style-type: none"> 0 番ポートが「タスク ID1 の 0 番ポート」からメッセージを受け取る 1 番ポートが「タスク ID2 の 1 番ポート」からメッセージを受け取る <p>ポート番号が 0 の場合、ポート番号を省略して "1:0" を "1" のように記述することができます。したがって、[["1:0"], ["2:1"]] は [["1"], ["2:1"]] と同義です。</p>
conf	マップ	エレメント独自の設定項目について設定します。

重要: from に設定する送信元のタスク ID は、自身のタスク ID より小さい数字である必要があります。

13.4 パイプライン設定ファイルの環境変数

device-connector-intdash 用パイプライン設定ファイルの設定値においては、環境変数を展開して使用することができます。環境変数を展開するには、パイプライン設定ファイル内で \$(VARIABLE_NAME) の形式で記述します。

例として、以下のような設定ファイル (conf.yaml) を考えます。path に \$(MY_FIFO_NAME) が含まれています。

```
...
- id: 3
  element: file-sink
  from: [[2]]
  conf:
    flush_size: 10
    path: /var/run/intdash/$(MY_FIFO_NAME).fifo
```

その上で、この設定ファイルを使ってデバイスコネクタを実行する際に環境変数を与えます。

```
$ MY_FIFO_NAME=device123 device-connector-run --config conf.yaml
```

そうすると、この場合、path の値が /var/run/intdash/device123.fifo となります。

13.5 パイプライン設定ファイルのサンプル

付属デバイスコネクタにはパイプライン設定ファイルのサンプルが付属しています。サンプルは `/etc/dc_conf` ディレクトリにインストールされます。

13.6 エレメント一覧

device-connector-intdash には、以下のエレメントがあらかじめ含まれています。

src エレメント

- [apt-analogtrx-src](#) (p. 47)
- [apt-cantrx-src](#) (p. 49)
- [file-src](#) (p. 50)
- [nmea-packet-src](#) (p. 50)
- [process-src](#) (p. 51)
- [text-src](#) (p. 52)
- [ubx-src](#) (p. 52)
- [v4l2-src](#) (p. 54)

filter エレメント

- [h264-split-filter](#) (p. 54)
- [iscp-v2-compat-filter](#) (p. 55)
- [jpeg-split-filter](#) (p. 62)
- [pcm-split-filter](#) (p. 62)
- [print-log-filter](#) (p. 65)
- [stat-filter](#) (p. 65)
- [ubx-iscpv2-filter](#) (p. 66)

sink エレメント

- [apt-cantrx-sink](#) (p. 66)
- [file-sink](#) (p. 67)
- [null-sink](#) (p. 68)
- [stdout-sink](#) (p. 68)

注釈: device-connector-intdash には以下のエレメントも付属していますが、intdash Terminal System 専用または実験的なものであるため、直接は使用しないでください。

- [h265-split-filter](#)

独自エレメントを開発する場合は、名前が付属エレメントと重複しないよう注意してください。

13.6.1 apt-analogtrx-src

エッジデバイスに接続された EDGEPLANT ANALOG-USB Interface からデータを取得するときに使用する src エレメントです。

このエレメントは、デバイスパスにより指定された EDGEPLANT ANALOG-USB Interface から専用カーネルモジュール経由でデータを取得します。そして [iscp-v2-compat フォーマット](#) (p. 110) (bytes 型) で送出します。

注釈: EDGEPLANT ANALOG-USB Interface 用のカーネルモジュールについては、[アプトボットのウェブサイト](#) の周辺機器についてのページを参照してください。

エレメントの種類

src

次のエレメントに送信するポートの数

1

送信ポートでの送信形式

カスタムタイプ "iscp-v2-compatible-msgs" (iscp-v2-compatible フォーマット (bytes 型) を複数個連結したメッセージ)

エレメント独自の設定項目は以下のとおりです。パイプライン設定ファイルの conf: 以下に記述します。

項目	設定値	説明
path	文字列	ANALOG-USB Interface のデバイスパス (例: /dev/apb-usb/by-id/usb-xxx)
input_send_rate	整数 (10 / 100 / 1000 / 10000 / 156250 / 312500 / 625000 / 1250000 / 2500000 / 5000000 / 10000000)	ANALOG-USB Interface のサンプリング周波数 [mHz](全ポート共通)
input_enabled	boolean(true / false) 8 個の配列	ANALOG-USB Interface のアナログ入力ポートごと (全 8 ポート) の有効/無効
input_voltage_min	整数 8 個の配列 (0 / -5000 / -10000)	ANALOG-USB Interface のアナログ入力ポートごと (全 8 ポート) の入力電圧 [mV] の最小値
input_voltage_max	整数 8 個の配列 (5000 / 10000)	ANALOG-USB Interface のアナログ入力ポートごと (全 8 ポート) の入力電圧 [mV] の最大値 ただし、同一ポートの最小値と最大値の組み合わせは以下のいずれかである必要があります。 <ul style="list-style-type: none"> • min:-10000, max:10000 • min:-5000, max:5000 • min:0, max:5000 上記に一致しない場合は実行時にエラーになります。
output_enabled	true / false	ANALOG-USB Interface のアナログ出力ポートの有効/無効
output_voltage	整数 (20~5000 (分解能 20))	ANALOG-USB Interface のアナログ出力ポートから出力する信号の電圧 [mV]
output_waveform_type	整数 (0 / 1 / 2 / 3 / 16)	ANALOG-USB Interface のアナログ出力ポートから出力する信号の波形 (0:擬似ランダム信号、1:正弦波、2:三角波、3:矩形波、16:固定)
output_frequency	整数 (1000~100000 (分解能 1000))	ANALOG-USB Interface のアナログ出力ポートから出力する信号波形の周波数 [mHz]
clock_id	文字列 (CLOCK_MONOTONIC / CLOCK_MONOTONIC_RAW)	タイムスタンプを算出するときにシステムからどのように時刻を取得するか <ul style="list-style-type: none"> • CLOCK_MONOTONIC: NTP が行う段階的な調整の影響を受ける • CLOCK_MONOTONIC_RAW: NTP が行う段階的な調整の影響を受けない intdash Edge Agent 2 の設定と同じにする必要があります。intdash Edge Agent 2 のデフォルトは CLOCK_MONOTONIC です。

次のページに続く

表 1 – 前のページからの続き

項目	設定値	説明
timestamp_mode	文字列 (device / host) デフォルト: device	受信したデータに対して、タイムスタンプをどのように付与するか <ul style="list-style-type: none"> device: ANALOG-USB Interface 上で、ANALOG-USB Interface のクロックを元にタイムスタンプを付与する host: エッジデバイスが ANALOG-USB Interface からのデータを USB 経由で受信したタイミングで、エッジデバイスのクロックを元にタイムスタンプを付与する

13.6.2 apt-cantrx-src

エッジデバイスに接続された EDGEPLANT CAN-USB Interface からデータを取得するときに使用する src エレメントです。

このエレメントは、デバイスパスにより指定された EDGEPLANT CAN-Interface から、専用カーネルモジュール経由でデータを取得します。そして、[iscp-v2-compatible フォーマット](#) (p. 110) (can_frame 型) で送ります。

注釈: EDGEPLANT CAN-USB Interface 用のカーネルモジュールについては、[アプトボットのウェブサイト](#)の周辺機器についてのページを参照してください。

エレメントの種類

src

次のエレメントに送信するポートの数

1

送信ポートでの送信形式

カスタムタイプ "iscp-v2-compatible-msgs" (iscp-v2-compatible フォーマット (can_frame 型) を複数個連結したメッセージ)

エレメント独自の設定項目は以下のとおりです。パイプライン設定ファイルの conf: 以下に記述します。

重要: EDGEPLANT CAN-USB Interface の 1 つのデバイスパスに対して apt-cantrx-src と apt-cantrx-sink を同時に使用する場合は、apt-cantrx-src と apt-cantrx-sink の設定をすべて同じにしてください。

項目	設定値	説明
path	文字列	CAN-USB Interface のデバイスパス (例: /dev/apb-usb/by-id/usb-xxx)
baudrate	整数 (125 / 250 / 500 / 10000)	CAN 通信のボーレート [kbps]
silent	true / false	CAN-USB Interface から CAN バスへの ACK の送信 <ul style="list-style-type: none"> true: ACK を送信しない false: ACK を送信する

次のページに続く

表 2 – 前のページからの続き

項目	設定値	説明
clock_id	文字列 (CLOCK_MONOTONIC / CLOCK_MONOTONIC_RAW)	<p>タイムスタンプを算出するときにシステムからどのように時刻を取得するか</p> <ul style="list-style-type: none"> CLOCK_MONOTONIC: NTP が行う段階的な調整の影響を受ける CLOCK_MONOTONIC_RAW: NTP が行う段階的な調整の影響を受けない <p>intdash Edge Agent 2 の設定と同じにする必要があります。intdash Edge Agent 2 のデフォルトは CLOCK_MONOTONIC です。</p>
timestamp_mode	文字列 (device / host) デフォルト: device	<p>受信したデータに対して、タイムスタンプをどのように付与するか</p> <ul style="list-style-type: none"> device: CAN-USB Interface 上で、ANALOG-USB Interface のクロックを元にタイムスタンプを付与する host: エッジデバイスが CAN-USB Interface からのデータを USB 経由で受信したタイミングで、エッジデバイスのクロックを元にタイムスタンプを付与する

13.6.3 file-src

指定されたパスのファイルを読み込み、その内容を送出します。(このエレメントは、Device Connector Framework に含まれる基本のエレメントです。)

エレメントの種類

src

次のエレメントに送信するポートの数

1

送信ポートでの送信形式

MIME タイプ "application/octet-stream"

エレメント独自の設定項目は以下のとおりです。conf: 以下に記述します。

項目	設定値	説明
path	文字列	読み取るファイルのパス

13.6.4 nmea-packet-src

このエレメントでは、デバイスパスにより指定された TTY デバイスから TTY ドライバー経由で NMEA データを取得します。

エレメントの種類

src

次のエレメントに送信するポートの数

1

送信ポートでの送信形式

MIME タイプ "text/plain" (1 つの NMEA センテンスが 1 つのメッセージ)

エレメント独自の設定項目は以下のとおりです。conf: 以下に記述します。

項目	設定値	説明
path	文字列	GPS デバイスのデバイスパス (例: /dev/ttyUSB9)
baudrate	整数 (0, 50, 75, 110, 134, 150, 200, 300, 600, 1200, 1800, 2400, 4800, 9600, 19200, 38400, 57600, 115200, 230400)	ボーレート (例:57600) termios で定義された左記いずれかの値である必要があります。

注釈:

- nmea-packet-src エレメントは、GPS デバイスの初期化は行いません。あらかじめ GPS デバイス (tty デバイス) が所定のボーレートで NMEA を出力する状態にしてください。
- EDGEPLANT T1 で L4T を使用する場合は、以下のようにすることで GPS デバイスを初期化してデータを取得することができます。
 1. edgeplant-l4t-tools サービスを有効にします。方法については、[EDGEPLANT T1 デベロッパーガイド](#) を参照してください。
 2. パイプライン設定ファイルに以下を追加します。

before_task:

```
- while true; do if test $(stty -F /dev/ttyTHS1 speed) = 57600; then break; fi; sleep 1; done
- sleep 3
```

13.6.5 process-src

プロセスを起動し、起動したプロセスの標準出力をバッファサイズに収まるように区切って送ります。(このエレメントは、Device Connector Framework に含まれる基本のエレメントです。)

エレメント独自の設定項目は以下のとおりです。conf: 以下に記述します。

エレメントの種類

src

次のエレメントに送信するポートの数

1

送信ポートでの送信形式

MIME タイプ "application/octet-stream"

項目	設定値	説明
program	文字列	実行ファイル
args	文字列の配列	実行ファイルに渡す引数
command	文字列	実行ファイルと引数を 1 文字列で指定したもの。program と args が指定されなかった場合に使用。
buffer_size	整数	バッファサイズ (バイト)。デフォルトは 255。

13.6.6 text-src

指定されたテキストを送出します。(このエレメントは、Device Connector Framework に含まれる基本のエレメントです。)

エレメントの種類

src

次のエレメントに送信するポートの数

1

送信ポートでの送信形式

MIME タイプ "application/octet-stream"

エレメント独自の設定項目は以下のとおりです。conf: 以下に記述します。

項目	設定値	説明
text	文字列	送出するテキスト
interval_ms	整数	メッセージ生成の間隔をミリ秒にて指定
repeat	整数	「メッセージ送出のタイミングごとに、メッセージを何回送出するか」を指定します。デフォルトは 1 です。例えば interval_ms: 100 かつ repeat: 5 の場合、100 ミリ秒経過ごとに 5 回メッセージを送出します。

13.6.7 ubx-src

u-blox GNSS モジュールから UBX プロトコルのメッセージを取得し、そのままバイナリデータとして送出します。

取得対象の UBX プロトコルのメッセージは以下のとおりです。

- UBX-ESF-STATUS
- UBX-HNR-STATUS
- UBX-HNR-ATT
- UBX-HNR-INS
- UBX-HNR-PVT

重要: u-blox GNSS モジュールとして NEO-M8U のみをサポートしています。NEO-M8U を搭載していないエッジデバイス (VTC1910-S など) はサポートしていません。

エレメントの種類

src

次のエレメントに送信するポートの数

1

送信ポートでの送信形式

MIME タイプ "application/octet-stream"

エレメント独自の設定項目は以下のとおりです。conf: 以下に記述します。

項目	設定値	説明
path	文字列	u-blox GNSS モジュールのデバイスパス (例: /dev/ttyTHS1)

次のページに続く

表 3 – 前のページからの続き

項目	設定値	説明
baud_rate	整数 (9600, 19200, 38400, 57600, 115200, 230400, 460800)	ボーレート (例:57600)
meas_rate_ms	小数	GNSS 測定周期 (ミリ秒) (例: 200) 50 ミリ秒以上の値である必要があります。 以下のメッセージの送信頻度に影響します。 <ul style="list-style-type: none"> • UBX-ESF-STATUS • UBX-HNR-STATUS
nav_rate	整数	ナビゲーションソリューションに対する測定頻度 (例: 1) 127 以下の値である必要があります。n を指定した場合、各ナビゲーションソリューションに対して n 回の測定を行います (n が大きくなると、送信頻度が下がります)。 以下のメッセージの送信頻度に影響します。 <ul style="list-style-type: none"> • UBX-ESF-STATUS • UBX-HNR-STATUS
esf_status_rate	整数	UBX-ESF-STATUS メッセージ送信頻度 (例: 1) n を指定した場合は、 $n * \text{nav_rate} * \text{meas_rate_ms}$ の周期で UBX-ESF-STATUS メッセージが送信されます (n が大きくなると、送信頻度が下がります)。 0 を指定した場合は、送信されません。
nav_status_rate	整数	UBX-NAV-STATUS メッセージ送信頻度 (例: 1) n を指定した場合は、 $n * \text{nav_rate} * \text{meas_rate_ms}$ の周期で UBX-NAV-STATUS メッセージが送信されます (n が大きくなると、送信頻度が下がります)。 0 を指定した場合は、送信されません。
high_nav_rate_hz	整数	UBX-HNR メッセージの送信間隔 (周波数) (例: 5) 20 以下の値である必要があります。 以下のメッセージの送信頻度に影響します。 <ul style="list-style-type: none"> • UBX-HNR-ATT • UBX-HNR-INS • UBX-HNR-PVT
hnr_att_rate	整数	UBX-HNR-ATT メッセージの送信頻度 (例: 1) n を指定した場合は、 $\text{high_nav_rate_hz} / n$ の周波数で UBX-HNR-ATT メッセージが送信されます (n が大きくなると、送信頻度が下がります)。 0 を指定した場合は、送信されません。
hnr_ins_rate	整数	UBX-HNR-INS メッセージの送信頻度 (例: 1) n を指定した場合は、 $\text{high_nav_rate_hz} / n$ の周波数で UBX-HNR-INS メッセージが送信されます (n が大きくなると、送信頻度が下がります)。 0 を指定した場合は、送信されません。

次のページに続く

表 3 – 前のページからの続き

項目	設定値	説明
hnr_pvt_rate	整数	UBX-HNR-PVT メッセージの送信頻度 (例: 1) n を指定した場合は、high_nav_rate_hz / n の周波数で UBX-HNR-PVT メッセージが送信されます (n が大きくなると、送信頻度が下がります)。 0 を指定した場合は、送信されません。

注釈: 各設定項目の詳細については、u-blox GNSS モジュールのドキュメント "u-blox 8 / u-blox M8 Receiver description" を参照してください。

13.6.8 v4l2-src

指定されたカメラから V4L2 (Video for Linux 2) 経由で H.264 形式の動画データを受け取り、h264_annex_b 型の [iscp-v2-compatible フォーマット](#) (p. 110) で送出します。

エレメントの種類

src

次のエレメントに送信するポートの数

1

送信ポートでの送信形式

カスタムタイプ "iscp-v2-compatible-msg" (h264_annex_b 型)

エレメント独自の設定項目は以下のとおりです。パイプライン設定ファイルの conf: 以下に記述します。

項目	設定値	説明
path	文字列	デバイスパス (例: /dev/video0)
type	文字列 (h264 / jpeg)	ドライバーから取得するデータの種類の。 • h264: H.264 のデータを取得します。 • jpeg: JPEG データを取得します。
width	整数	画像幅。カメラが対応している画像幅を指定してください。
height	整数	画像高さ。カメラが対応している画像高さを指定してください。
fps	整数	FPS。カメラが対応している FPS を指定してください。

13.6.9 h264-split-filter

メッセージを受け取り、H.264 形式の動画データとしてパースし、[iscp-v2-compatible フォーマット](#) (p. 110) (h264_annex_b 型) で送出します。

エレメントの種類

filter

前のエレメントから受信するポートの数

1

次のエレメントに送信するポートの数

1

受信ポートでの受信形式

H.264 Annex B 形式のバイナリストリームを任意の箇所で区切ったバイト列

送信ポートでの送信形式

カスタムタイプ "iscp-v2-compatible-msg" (h264_annex_b 型)

エレメント独自の設定項目は以下のとおりです。conf: 以下に記述します。

項目	設定値	説明
clock_id	文字列 (CLOCK_MONOTONIC / CLOCK_MONOTONIC_RAW)	<p>タイムスタンプを算出するときにシステムからどのように時刻を取得するか</p> <ul style="list-style-type: none"> CLOCK_MONOTONIC: NTP が行う段階的な調整の影響を受ける CLOCK_MONOTONIC_RAW: NTP が行う段階的な調整の影響を受けない <p>intdash Edge Agent 2 の設定と同じにする必要があります。intdash Edge Agent 2 のデフォルトは CLOCK_MONOTONIC です。</p>
delay_ms	小数	<p>タイムスタンプを付与する際に差し引くオフセット (カメラ処理時間)</p> <p>例えば、カメラ内での処理に 100 ミリ秒かかる場合は、「100」を指定します。これにより、タイムスタンプ付与時に 100 ミリ秒前の時刻が使用されます。</p> <p>設定例:</p> <pre>conf: clock_id: CLOCK_MONOTONIC delay_ms: 100</pre>

13.6.10 iscp-v2-compatible-filter

データを指定されたフォーマットでパースし、指定に応じてタイムスタンプを付与し、can_frame / jpeg / string/nmea / bytes / float64 / int64 / string いずれかの型の **iscp-v2-compatible フォーマット** (p. 110) で送出します。

エレメントの種類

filter

前のエレメントから受信するポートの数

1

次のエレメントに送信するポートの数

1

受信ポートでの受信形式

任意 (timestamp および convert_rule の指定に適合していること)

送信ポートでの送信形式

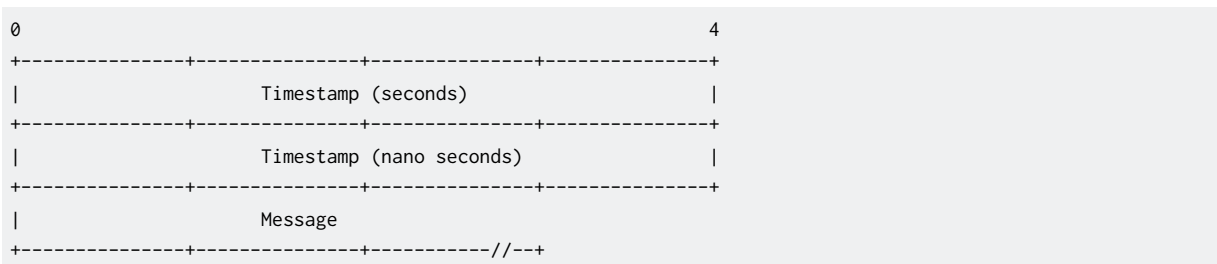
カスタムタイプ "iscp-v2-compatible-msg" (can_frame 型 / jpeg 型 / string/nmea 型 / bytes 型 / float64 型 / int64 型 / string 型)

エレメント独自の設定項目は以下のとおりです。conf: 以下に記述します。

項目	設定値	説明
timestamp	文字列 (stamped)、または、タイムスタンプ付与方法を定義したオブジェクト	<p>受信するメッセージにタイムスタンプ（受信データのタイムスタンプ (p. 56)）が含まれているか</p> <ul style="list-style-type: none"> stamped: タイムスタンプが含まれている（このタイムスタンプを出力時にも付与します。） <pre>conf: timestamp: stamped</pre> <ul style="list-style-type: none"> stamp: タイムスタンプが含まれていない（出力時には clock_id の指定に従ってタイムスタンプを付与します。） <pre>conf: timestamp: stamp: clock_id: CLOCK_MONOTONIC</pre> <p>clock_id の指定:</p> <ul style="list-style-type: none"> CLOCK_MONOTONIC: NTP が行う段階的な調整の影響を受ける時計を使用する CLOCK_MONOTONIC_RAW: NTP が行う段階的な調整の影響を受けない時計を使用する <p>clock_id は、intdash Edge Agent 2 の設定と同じにする必要があります。intdash Edge Agent 2 のデフォルトは CLOCK_MONOTONIC です。</p>
convert_rule	変換ルールを定義したオブジェクト	<p>iscp-v2-compatible フォーマットに変換するためのルールを指定します。詳細は以下を参照してください。</p> <ul style="list-style-type: none"> can (p. 57) jpeg (p. 57) nmea (p. 58) bytes (p. 58) float64 (p. 59) int64 (p. 60) string (p. 61)

受信データのタイムスタンプ

受信するメッセージの先頭にタイムスタンプが追加されている場合は、設定項目 timestamp の値を stamped とします。この場合、受信したメッセージの先頭 8 バイトがタイムスタンプとしてパースされます。期待するフォーマットは以下のとおりです。



この Timestamp (seconds) フィールド、Timestamp (nano seconds) フィールドの値は、それぞれ iscp-v2-compatible での出力時に同名のフィールドに出力されます。

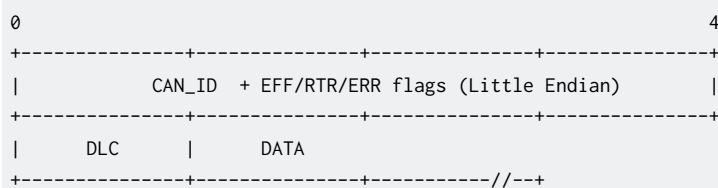
Message は、設定項目 `convert_rule` で指定された任意のフォーマットのデータです。

CAN データ用変換ルール (can)

設定例:

```
conf:
  timestamp:
    stamp:
      clock_id: CLOCK_MONOTONIC
  convert_rule:
    can:
      lower_hex: true
```

CAN データ用の変換で期待する入力データのフォーマットは以下のとおりです。



- CAN_ID + EFF/RTR/ERR flags: [Linux SocketCan の can_frame 構造体](#) の can_id。
- DLC: [Linux SocketCan の can_frame 構造体](#) の len。
- DATA: [Linux SocketCan の can_frame 構造体](#) の data。

この変換ルールでは以下の項目を設定可能です。

項目	設定値	説明
lower_hex	boolean(true / false)	出力する iscp-v2-compatible フォーマットの Data Name フィールドに入れる 16 進数数値表現を小文字で行うか

出力される [iscp-v2-compatible](#) (p. 110) の各フィールドは以下のようになります。

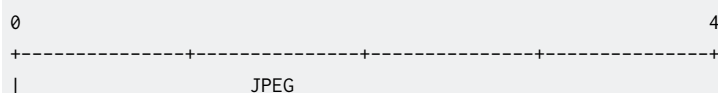
- Data Type: can_frame
- Data Name: 上記入力データ内の CAN_ID (16 進 8 桁の文字列表現) (ただし EFF フラグはオフになります)
- Payload: 上記入力データ内の DATA フィールドの内容

JPEG 用変換ルール (jpeg)

設定例:

```
conf:
  timestamp:
    stamp:
      clock_id: CLOCK_MONOTONIC
  convert_rule: jpeg
```

JPEG 用の変換では、入力されたメッセージを、JPEG (ITU-T Rec. T.81 ; ISO/IEC 10918-1, 10918-2) としてパースします。期待する入力データのフォーマットは以下のとおりです。



(次のページに続く)

(前のページからの続き)

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

出力される `iscp-v2-compat` (p. 110) の各フィールドは以下のようになります。

- Data Type: jpeg
- Data Name: 空文字
- Payload: 上記入力データ内の JPEG フィールドの内容

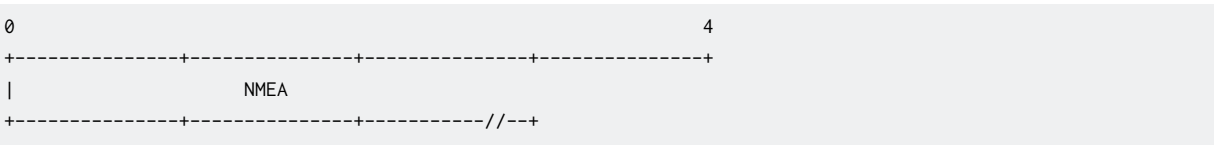
この変換ルールには設定項目はありません。

NMEA データ用変換ルール (nmea)

設定例:

```
conf:
  timestamp:
    stamp:
      clock_id: CLOCK_MONOTONIC
  convert_rule: nmea
```

NMEA データ用の変換では、入力されたメッセージを NMEA 0183 としてパースします。期待する入力データのフォーマットは以下のとおりです。



出力される `iscp-v2-compat` (p. 110) の各フィールドは以下のようになります。

- Data Type: string/nmea
- Data Name: <NMEA のトーカ>/<NMEA のメッセージ> (上記入力データ内の NMEA フィールドから抽出されたもの)
- Payload: 上記入力データ内の NMEA フィールドの内容

この変換ルールには設定項目はありません。

Bytes データ用変換ルール (bytes)

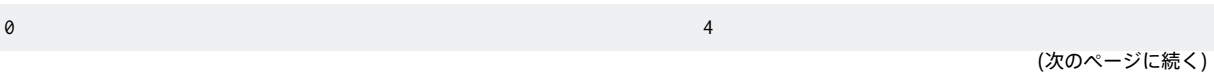
設定例:

```
conf:
  clock_id: CLOCK_MONOTONIC
  convert_rule:
    bytes:
      name: null
```

この変換ルールでは以下の項目が設定可能です。

項目	設定値	説明
name	string または null	出力する iscp-v2-compat フォーマットの Data Name フィールドに入れる文字列

設定項目の name に何らかの文字列を設定した場合は、期待する入力データのフォーマットは以下のとおりです。



(次のページに続く)

(前のページからの続き)

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

設定項目の `name` が `null` の場合、データ名称は入力データ内で指定します。期待する入力データのフォーマットは以下のとおりです。

```
0                                     4
+-----+-----+-----+-----+
| Name Length |               Name               |
+-----+-----+-----+-----+
//--+
|               Bytes               |
+-----+-----+-----+-----+
//--+
```

- Name Length: Name フィールドの長さ。
- Name: データ名称として使用する文字列。

出力される [iscp-v2-compatible](#) (p. 110) の各フィールドは以下のようになります。

- Data Type: bytes
- Data Name: 設定項目 `name` に指定された文字列。または、`name` が `null` の場合、上記入力データ内の Name フィールドの内容。
- Payload: 上記入力データ内の Bytes フィールドの内容

Float64 データ用変換ルール (float64)

設定例:

```
conf:
  clock_id: CLOCK_MONOTONIC
  convert_rule:
    float64:
      name: null
```

この変換ルールでは以下の項目が設定可能です。

項目	設定値	説明
name	string または null	出力する iscp-v2-compatible フォーマットの Data Name フィールドに入れる文字列

設定項目の `name` に何らかの文字列を設定した場合は、期待する入力データのフォーマットは以下のとおりです。

```
0                                     4
+-----+-----+-----+-----+
|               Float (Little Endian)               |
+-----+-----+-----+-----+
//--+
```

設定項目の `name` が `null` の場合、データ名称は入力データ内で指定します。期待する入力データのフォーマットは以下のとおりです。

```
0                                     4
+-----+-----+-----+-----+
| Name Length |               Name               |
+-----+-----+-----+-----+
//--+
```

(次のページに続く)

(前のページからの続き)

+ Float (Little Endian) +	
+-----+	+-----+

- Name Length: Name フィールドの長さ。
- Name: データ名称として使用する文字列。

出力される [iscp-v2-compat](#) (p. 110) の各フィールドは以下のようになります。

- Data Type: float64
- Data Name: 設定項目 `name` に指定された文字列。または、`name` が `null` の場合、上記入力データ内の Name フィールドの内容。
- Payload: 上記入力データ内の Float フィールドの内容

int64 データ用変換ルール (int64)

設定例:

```
conf:
  clock_id: CLOCK_MONOTONIC
  convert_rule:
    int64:
      name: null
```

この変換ルールでは以下の項目が設定可能です。

項目	設定値	説明
name	string または null	出力する iscp-v2-compat フォーマットの Data Name フィールドに入れる文字列

設定項目の `name` に何らかの文字列を設定した場合は、期待する入力データのフォーマットは以下のとおりです。

0	4
+-----+	+-----+
+ Int (Little Endian) +	
+-----+	+-----+

設定項目の `name` が `null` の場合、データ名称は入力データ内で指定します。期待する入力データのフォーマットは以下のとおりです。

0	4
+-----+	+-----+
Name Length	Name
+-----+	+-----+//
+ Int (Little Endian) +	
+-----+	+-----+

- Name Length: Name フィールドの長さ。
- Name: データ名称として使用する文字列。

出力される [iscp-v2-compat](#) (p. 110) の各フィールドは以下のようになります。

- Data Type: int64
- Data Name: 設定項目 `name` に指定された文字列。または、`name` が `null` の場合、上記入力データ内の `Name` フィールドの内容。
- Payload: 上記入力データ内の `Int` フィールドの内容

String データ用変換ルール (string)

設定例:

```
conf:
  clock_id: CLOCK_MONOTONIC
  convert_rule:
    string:
      name: abc
```

この変換ルールでは以下の項目が設定可能です。

項目	設定値	説明
<code>name</code>	string または <code>null</code>	出力する <code>iscp-v2-compatible</code> フォーマットの <code>Data Name</code> フィールドに入れる文字列

設定項目の `name` に何らかの文字列を設定した場合は、期待する入力データのフォーマットは以下のとおりです。

```
0                                     4
+-----+-----+-----+-----+
|                                     |
|           String                   |
+-----+-----+-----+-----+//--+
```

設定項目の `name` が `null` の場合、データ名称は入力データ内で指定します。期待する入力データのフォーマットは以下のとおりです。

```
0                                     4
+-----+-----+-----+-----+
| Name Length |           Name           |
+-----+-----+-----+-----+//--+
|           String                   |
+-----+-----+-----+-----+//--+
```

- Name Length: `Name` フィールドの長さ。
- Name: データ名称として使用する文字列。

出力される [iscp-v2-compatible](#) (p. 110) の各フィールドは以下のようになります。

- Data Type: string
- Data Name: 設定項目 `name` に指定された文字列。または、`name` が `null` の場合、上記入力データ内の `Name` フィールドの内容。
- Payload: 上記入力データ内の `String` フィールドの内容

13.6.11 jpeg-split-filter

jpeg-split-filter は、受信したバイナリストリームを JPEG が複数並んでいるバイナリとしてパースして、1 枚ごとの JPEG に分けて送信します。

エレメントの種類

filter

前のエレメントから受信するポートの数

1

次のエレメントに送信するポートの数

1

受信ポートでの受信形式

JPEG フォーマットのバイナリを並べたバイナリストリームを任意の箇所で区切ったバイト列

送信ポートでの送信形式

MIME タイプ "image/jpeg" (1 枚の画像が 1 つのメッセージ)

エレメント独自の設定項目はありません。

13.6.12 pcm-split-filter

GStreamer から出力された音声データを受け取り、bytes 型の [iscp-v2-compatible フォーマット](#) (p. 110) に変換して送出します。

重要: エッジデバイスとして T1 / VTC1910-S 以外を使用する場合は正しく設定を行えない可能性があります。

エレメントの種類

filter

前のエレメントから受信するポートの数

1

次のエレメントに送信するポートの数

1

受信ポートでの受信形式

任意 (前のタスクで process-src エレメントを使って GStreamer から音声出力させ、その出力を本タスクで受け取るようにします。)

GStreamer からの出力は [Raw Audio Media Types](#) における以下のタイプとしてください。

- format: S16LE / S32LE / F32LE
- rate: 48000
- channels: 1
- channel-mask: 使わない (GStreamer コマンドに設定しないこと)
- layout: 使わない (GStreamer コマンドに設定しないこと)

送信ポートでの送信形式

カスタムタイプ "iscp-v2-compatible-msg" (bytes 型)

エレメント独自の設定項目は以下のとおりです。パイプライン設定ファイルの conf: 以下に記述します。

項目	設定値	説明
clock_id	文字列 (CLOCK_MONOTONIC / CLOCK_MONOTONIC_RAW)	<p>タイムスタンプを算出するときにシステムからどのように時刻を取得するか</p> <ul style="list-style-type: none"> CLOCK_MONOTONIC: NTP が行う段階的な調整の影響を受ける CLOCK_MONOTONIC_RAW: NTP が行う段階的な調整の影響を受けない <p>intdash Edge Agent 2 の設定と同じにする必要があります。intdash Edge Agent 2 のデフォルトは CLOCK_MONOTONIC です。</p>
delay_ms	小数	<p>データ打刻のオフセットを設定入力します。音声が発生してからデバイスコネクタにデータが届くまでに 100 ミリ秒かかる場合は、100 を指定します。ほとんどの場合 0 で問題ありません。</p>
audio_element	文字列 (Mic Jack/ y Jack-state)	<p>コンピューターのマイクジャックを使用する場合、マイクジャックの状態（抜き差し）を示すデバイスエレメント名を入力します。</p> <ul style="list-style-type: none"> Terminal System のコンピューターとして VTC 1910-S を使用する場合: Mic Jack Terminal System のコンピューターとして EDGEPLANT T1 を使用する場合: y Jack-state
audio_iface	文字列 (card /mixer)	<p>音声を制御するデバイスの種別を入力します。</p> <ul style="list-style-type: none"> Terminal System のコンピューターとして VTC 1910-S を使用する場合: card Terminal System のコンピューターとして EDGEPLANT T1 を使用する場合: mixer
audio_format	文字列 (S16LE / S32LE / F32LE)	<p>音声データの形式を選択します</p> <ul style="list-style-type: none"> S16LE: signed 16 bit little-endian S32LE: signed 32 bit little-endian F32LE: floating-point little-endian
audio_rate	整数 (48000) (固定)	サンプリング周波数 [Hz]
audio_channels	整数 (1) (固定)	チャンネルの数。
audio_volume_iface	文字列 (mixer)	<p>音量を設定する場合、音量を制御するデバイスの種別を入力します。</p> <ul style="list-style-type: none"> Terminal System のコンピューターとして VTC 1910-S を使用する場合: mixer Terminal System のコンピューターとして EDGEPLANT T1 を使用する場合: mixer
audio_volume_element	文字列 (Capture Volume/ y Mic Capture Volume)	<p>音量を設定する場合、音量を示すデバイスエレメント名を入力します。</p> <ul style="list-style-type: none"> Terminal System のコンピューターとして VTC 1910-S を使用する場合: Capture Volume Terminal System のコンピューターとして EDGEPLANT T1 を使用する場合: y Mic Capture Volume

次のページに続く

表 10 – 前のページからの続き

項目	設定値	説明
audio_volume_value	整数	<p>音量を設定する場合、音量として設定する値を入力します。</p> <ul style="list-style-type: none"> Terminal System のコンピューターとして VTC 1910-S を使用する場合: <ul style="list-style-type: none"> 範囲: 0 - 46 デフォルト: 28 0 のときに 16.00 dB 設定 値を 1 上げるごとに 1.00 dB 上昇 Terminal System のコンピューターとして EDGEPLANT T1 を使用する場合: <ul style="list-style-type: none"> 範囲: 0 - 20 デフォルト: 20 0 のときに 0.00 dB (音量上昇なし、等倍) 設定 値を 1 上げるごとに 1.00 dB 上昇
audio_boost_element	文字列 (Mic Boost Volume/ y Mic Boost Capture Volume)	<p>音量を設定する場合、ブーストを示すデバイスエレメント名を入力します。</p> <ul style="list-style-type: none"> Terminal System のコンピューターとして VTC 1910-S を使用する場合: Mic Boost Volume Terminal System のコンピューターとして EDGEPLANT T1 を使用する場合: y Mic Boost Capture Volume
audio_boost_value	整数	<p>音量を設定する場合、ブーストとして設定する値を入力します。</p> <ul style="list-style-type: none"> Terminal System のコンピューターとして VTC 1910-S を使用する場合: <ul style="list-style-type: none"> 範囲: 0 - 3 デフォルト: 0 0 のときに 0.00 dB 設定 値を 1 上げるごとに 10.00 dB 上昇 Terminal System のコンピューターとして EDGEPLANT T1 を使用する場合: <ul style="list-style-type: none"> 範囲: 0 - 3 デフォルト: 1 0: -20.00 dB (Mute) 1: 0.00 dB (gain なし) 2: 20.00 dB 3: 30.00 dB

13.6.13 print-log-filter

受け取ったメッセージの数とサイズの情報をログとして出力します。ログの出力先としてデバイスコネクタのログ（[ログの確認](#) (p. 83)）または標準エラーを選択できます。

エレメントの種類

filter

前のエレメントから受信するポートの数

1

次のエレメントに送信するポートの数

1

受信ポートでの受信形式

任意

送信ポートでの送信形式

任意（受け取ったメッセージをそのまま送信ポートに書き出します）

エレメント独自の設定項目は以下のとおりです。conf: 以下に記述します。

項目	設定値	説明
interval_ms	小数	ログを出力するミリ秒単位の周期（例：1000） 例えば「1000」を指定すると、受け取ったメッセージ数と合計サイズを、1000 ミリ秒ごとにログとして出力します。
tag	string	ログを識別できるように各行の先頭に出力されるタグ。 connector1 と指定した場合は、以下のように出力されます。 [connector1] 100 msgs, 1000 bytes
output	string (log-trace / stderr)	ログの出力先を指定します。 log-trace: デバイスコネクタのログの仕組みを使用して出力します。ログレベルは trace となります。 stderr: 標準エラーに出力します。

13.6.14 stat-filter

パイプラインを流れるメッセージの回数やサイズを確認するためのエレメントです。受け取ったメッセージはそのまま加工せずに送信ポートから送ります。受け取ったメッセージの回数とサイズを記録し、その統計情報を標準エラー出力に書き出します。（このエレメントは、Device Connector Framework に含まれる基本のエレメントです。）

エレメントの種類

filter

前のエレメントから受信するポートの数

1

次のエレメントに送信するポートの数

1

受信ポートでの受信形式

任意

送信ポートでの送信形式

受信ポートに入力されたデータをそのまま出力

エレメント独自の設定項目は以下のとおりです。conf: 以下に記述します。

項目	設定値	説明
interval_ms	整数	出力頻度をミリ秒にて指定

13.6.15 ubx-iscpv2-filter

ubx-src から UBX メッセージを受け取り、メッセージをパースして Data Name をバイナリデータの先頭に付与し、iscpv2-compat-filter に送信します。

エレメントの種類

filter

前のエレメントから受信するポートの数

1

次のエレメントに送信するポートの数

1

受信ポートでの受信形式

MIME タイプ "application/octet-stream" (ubx-src が送信したメッセージ)

送信ポートでの送信形式

MIME タイプ "application/octet-stream" (iscpv2-compat-filter が受信するメッセージ)

エレメント独自の設定項目はありません。

13.6.16 apt-cantrx-sink

メッセージを受け取り、iscpv2-compat フォーマット (p. 110) (can_frame 型) としてパースし、専用カーネルモジュール経由で EDGEPLANT CAN-USB Interface に送出します。

注釈: EDGEPLANT CAN-USB Interface 用のカーネルモジュールについては、[アプトボットのウェブサイト](#)の周辺機器についてのページを参照してください。

エレメントの種類

sink

前のエレメントから受信するポートの数

1

受信ポートでの受信形式

任意 (iscpv2-compat フォーマット (can_frame 型) を複数個連結したバイナリストリーム)

エレメント独自の設定項目は以下のとおりです。conf: 以下に記述します。

重要: EDGEPLANT CAN-USB Interface の 1 つのデバイスパスに対して apt-cantrx-src と apt-cantrx-sink を同時に使用する場合は、apt-cantrx-src と apt-cantrx-sink の設定をすべて同じにしてください。

項目	設定値	説明
path	文字列	CAN-USB Interface のデバイスパス (例: /dev/apb-usb/by-id/usb-xxx)
baudrate	整数 (125 / 250 / 500 / 10000)	CAN 通信のボーレート [kbps]
silent	true / false	CAN-USB Interface から CAN バスに ACK を送信しない設定の有効/無効 (true:ACK なし、false:ACK あり)
clock_id	文字列 (CLOCK_MONOTONIC / CLOCK_MONOTONIC_RAW)	タイムスタンプを算出するときにシステムからどのように時刻を取得するか 本要素はタイムスタンプを付与しないため、任意の値で構いません。 ただし、EDGEPLANT CAN-USB Interface の 1 つのデバイスパスに対して、apb-cantrx-src と本要素を同時に使用する場合は、apb-cantrx-src と同じ設定にしてください。
timestamp_mode	文字列 (device / host) デフォルト: device	受信したデータに対して、タイムスタンプをどのように付与するか 本要素はタイムスタンプを付与しないため、任意の値で構いません。 ただし、EDGEPLANT CAN-USB Interface の 1 つのデバイスパスに対して、apb-cantrx-src と本要素を同時に使用する場合は、apb-cantrx-src と同じ設定にしてください。

13.6.17 file-sink

指定されたパスに、受け取ったメッセージをそのまま書き出します。(この要素は、Device Connector Framework に含まれる基本の要素です。)

要素独自の設定項目は以下のとおりです。conf: 以下に記述します。

要素の種類

sink

前の要素から受信するポートの数

1

受信ポートでの受信形式

任意

項目	設定値	説明
path	文字列	出力先のファイルパス
create	true / false	true の場合、出力先のファイルがない場合は作成する。デフォルトは false
separator	文字列	メッセージ出力ごとに挿入するテキスト。デフォルトは空
flush_size	整数	入力をフラッシュするまでにバッファにためるデータ量 (バイト)。デフォルトは 0

13.6.18 null-sink

メッセージを受け取りますが、それを何にも使わずに破棄します。(このエレメントは、Device Connector Framework に含まれる基本のエレメントです。)

このエレメントに設定項目はありません

エレメントの種類

sink

前のエレメントから受信するポートの数

1

受信ポートでの受信形式

任意

13.6.19 stdout-sink

受け取ったメッセージを標準出力に書き出します。(このエレメントは、Device Connector Framework に含まれる基本のエレメントです。)

エレメントの種類

sink

前のエレメントから受信するポートの数

1

受信ポートでの受信形式

任意

エレメント独自の設定項目は以下のとおりです。conf: 以下に記述します。

項目	設定値	説明
separator	文字列	メッセージ出力ごとに挿入するテキスト。デフォルトは空

14 独自デバイスコネクタの使用

付属デバイスコネクタ [device-connector-intdash](#) (p. 42) の代わりに、独自のプログラムをデバイスコネクタとして使うことも可能です。

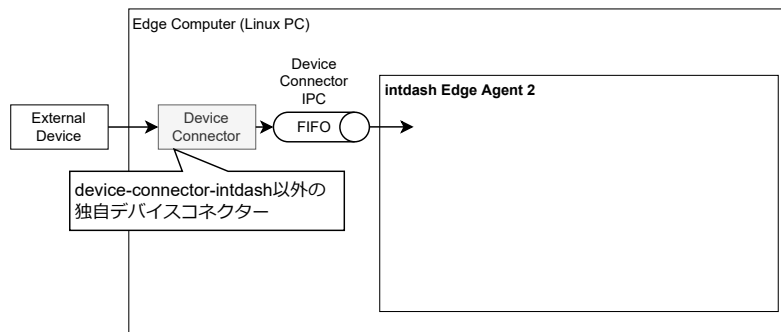


図 19 アップストリームの場合の独自デバイスコネクタ（ダウンストリームの場合も同様）

独自デバイスコネクタを使用する場合も、ストリーマーとのデータのやり取りは FIFO を使って行います。

アップストリーム用のデバイスコネクタは、FIFO に [iscp-v2-compatible フォーマット](#) (p. 110) でデータを書き込むようにしてください。ダウンストリーム用のデバイスコネクタの場合は、FIFO から [iscp-v2-compatible フォーマット](#) (p. 110) のデータを読み出せるようにしてください。

注釈: デバイスコネクタ IPC に関する設定（[アップストリーム用](#) (p. 104)、[ダウンストリーム用](#) (p. 104)）の format を変更することで、旧 intdash Edge Agent 用の logger-msg フォーマットを使用することも可能です（非推奨）。

デバイスコネクタ IPC に関する設定（[アップストリーム用](#) (p. 104)、[ダウンストリーム用](#) (p. 104)）では、デバイスコネクタを intdash Edge Agent 2 と同時に起動する設定や、デバイスコネクタに与える引数や環境変数、使用する FIFO のパスの指定を行うことができます。

15 フィルタリング／サンプリング

15.1 フィルターとは

intdash Edge Agent 2 では、intdash Edge Agent 2 内を流れるデータポイントに対して、フィルターを適用することができます。フィルターを使用すると、以下のことができます。

- アップストリーム方向の場合: デバイスコネクターから取得したデータポイントのうち、指定された条件に一致するデータポイントを選び出し、指定されたアップストリーム、または、遅延アップロードの処理に渡す。
- ダウンストリーム方向の場合: サーバーから取得したデータポイントのうち、指定された条件に一致するデータポイントを選び出し、指定されたデバイスコネクターに渡す。

さまざまな条件を指定するために、条件の指定方法が異なるいくつかのフィルタータイプが用意されています。例えば、name タイプのフィルターでは、条件としてデータ名称を指定します。この場合、流れてきたデータポイントのデータ名称が指定されたものと一致すると、フィルターの定義に従って処理されることになります。

使用できるフィルターのタイプは以下のとおりです。条件の設定方法については各フィルタータイプの説明を参照してください。:

- [always](#) (p. 73)
- [type](#) (p. 73)
- [name](#) (p. 74)
- [src-id](#) (p. 75)
- [rename](#) (p. 75)
- [sampling](#) (p. 76)
- [h264-essential-nal-units](#) (p. 78)

注釈: フィルターは、上記のタイプのみを使用できます。ユーザーが独自のフィルタータイプを開発することはできません。既存のフィルタータイプでは不足がある場合は、同様の仕組みをデバイスコネクターにおいて開発することで対応できます。

複数のフィルターを設定して多段階の条件を設定することもできます。複数のフィルターを設定した場合、設定ファイルに書かれた順にフィルターが適用されます。

また、一定時間ごとに最初の 1 つのデータポイントを選択するフィルター (sampling タイプのフィルター) を設定することで、サンプリングを実行することができます。

注釈: ここで説明するフィルターは、intdash Edge Agent 2 内を流れるデータに対して適用されるものです。サーバーからダウンストリームする対象を指定するための「ダウンストリームフィルター」とは別の機能です。ダウンストリームフィルターについては、[ダウンストリーム](#) (p. 38) を参照してください。

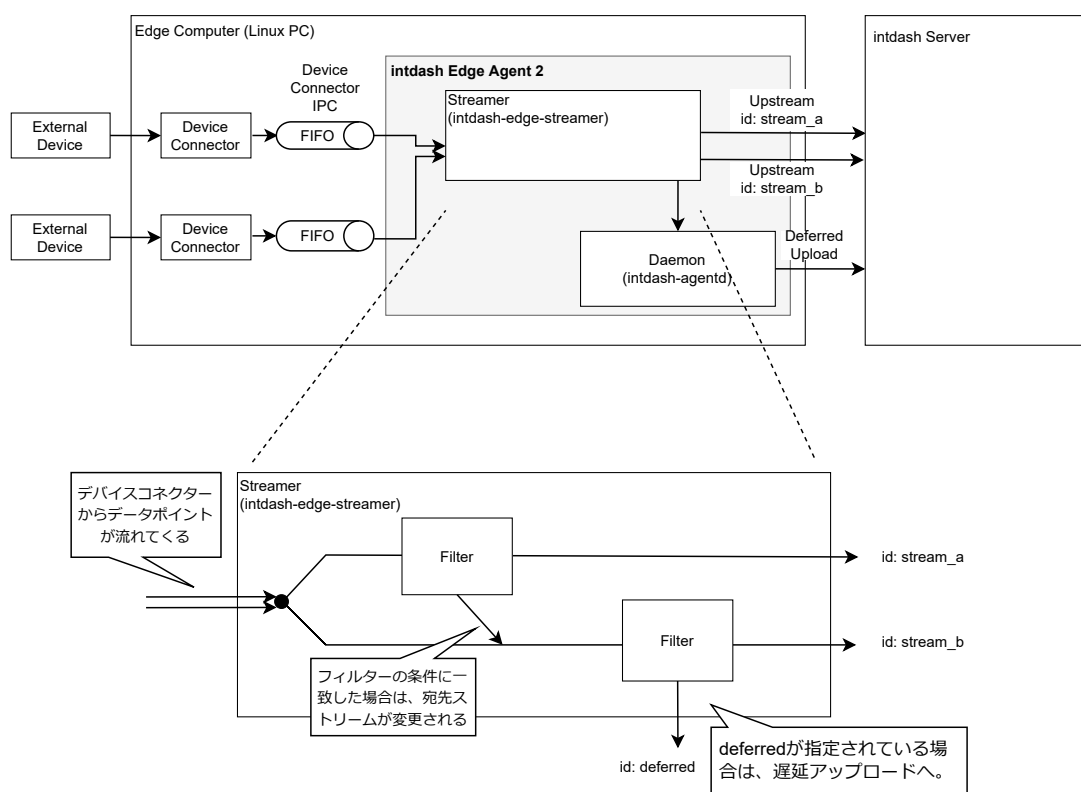


図 20 フィルター（アップストリームの場合）

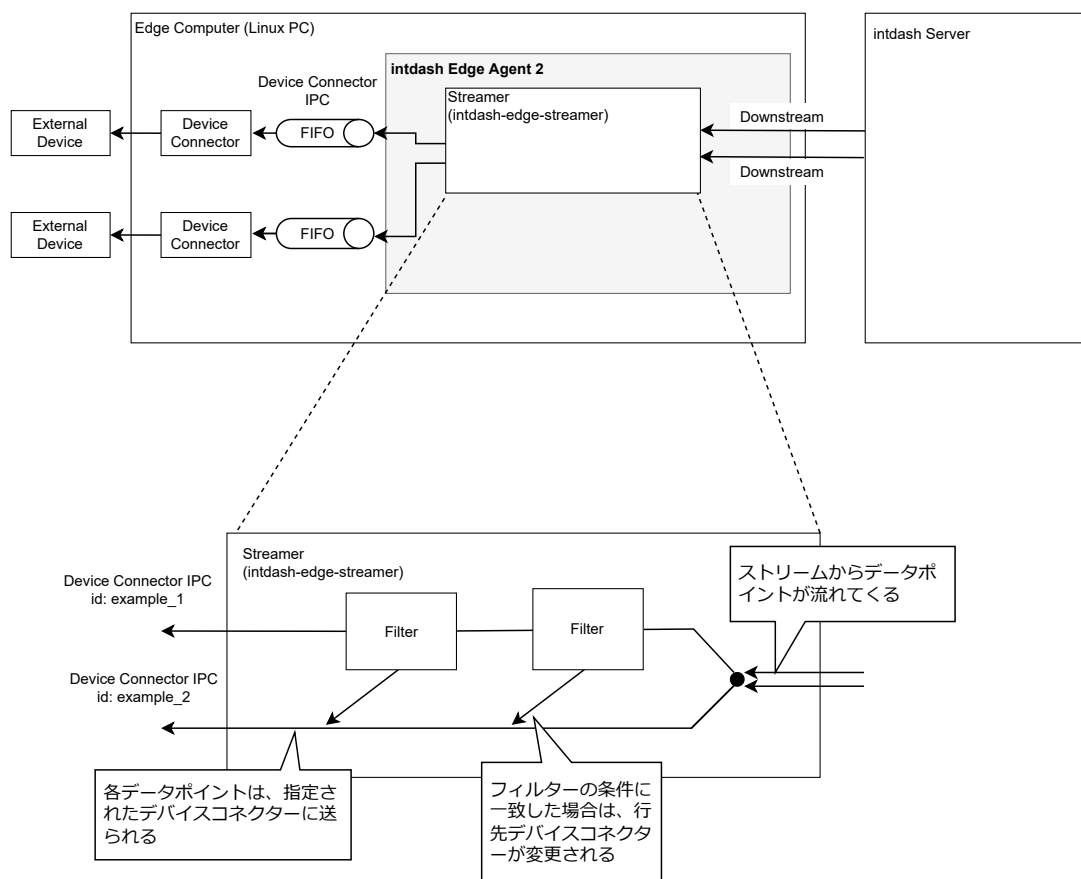


図 21 フィルター（ダウンストリームの場合）

15.2 フィルターを設定する

フィルターを作成するには以下のコマンドを実行します。

アップストリーム方向:

```
$ intdash-agentctl config filter upstream --create <config_in_yaml_format>
```

ダウンストリーム方向:

```
$ intdash-agentctl config filter downstream --create <config_in_yaml_format>
```

例:

```
$ intdash-agentctl config filter upstream --create '  
  id: string_to_deferred  
  enabled: true  
  type: type  
  target:  
    dest_ids:  
      - recoverable  
    type: string  
  change_to:  
    dest_id: deferred  
,
```

作成済みのフィルターの設定を変更するには、ID と、変更したい設定を与えます。

```
$ intdash-agentctl config filter upstream --modify <id> <config_in_yaml_format>  
$ intdash-agentctl config filter downstream --modify <id> <config_in_yaml_format>
```

キー	型	説明
id	string	フィルター設定を識別する任意の ID
enabled	bool	フィルターの有効 (true) / 無効 (false)
type	string	フィルターのタイプ (p. 73)
target	object	フィルターのタイプに応じた対象データポイントの条件指定。条件の指定方法については、フィルターの種類別の説明を参照してください。データポイントがここで記載された条件に一致した場合、データポイントに change_to で指定された変更が行われます。
change_to	object	フィルターの条件に合致したデータポイントは、ここに指定された変更が行われます。

15.3 フィルターのタイプ

フィルターのタイプと、それぞれで指定できる条件は以下のとおりです。

15.3.1 always

always タイプのフィルターでは、データポイントの行先を条件にしてフィルタリングが行われます。データポイントの行先が `target.dest_ids` で列挙された行先のいずれかである場合、その行先は `change_to` で指定されたものに変更されます。

条件指定（`target`）では以下を指定します。

キー	型	説明
<code>dest_ids</code>	[string]	アップストリームの場合、行先ストリームの ID のリスト。 ダウンストリームの場合、行先デバイスコネクター IPC の ID のリスト。

`change_to` では、以下のように新しい行先を指定します。

キー	型	説明
<code>dest_id</code>	string	アップストリームの場合、行先ストリームの ID。 ダウンストリームの場合、行先デバイスコネクター IPC の ID。

例：

```
id: example
enabled: true
type: always
target:
  dest_ids:
    - recoverable
change_to:
  dest_id: deferred
```

この例では、行先が `recoverable` であるデータポイントすべてについて、行先が `deferred` に変更されます。

15.3.2 type

type タイプのフィルターでは、データポイントの行先とデータ型を条件にしてフィルタリングが行われます。データポイントの行先が `target.dest_ids` で列挙された行先のいずれかであり、データ型が `target.type` で指定されたものである場合、その行先は `change_to` で指定されたものに変更されます。

条件指定（`target`）では以下を指定します。

キー	型	説明
<code>dest_ids</code>	[string]	アップストリームの場合、行先ストリームの ID のリスト。 ダウンストリームの場合、行先デバイスコネクター IPC の ID のリスト。
<code>type</code>	string	iSCP v2 のデータ型名によるフィルター条件（正規表現）

`change_to` では、以下のように新しい行先を指定します。

キー	型	説明
dest_id	string	アップストリームの場合、行先ストリームの ID。 ダウンストリームの場合、行先デバイスコネクタ IPC の ID。

例：

```
id: example
enabled: true
type: type
target:
  dest_ids:
    - recoverable
  type: h26.+
change_to:
  dest_id: deferred
```

この例では、行先が recoverable であるデータポイントのうち、データ型が h26.+ の正規表現にマッチするものについて、行先が deferred に変更されます。

15.3.3 name

name タイプのフィルターでは、データポイントの行先とデータ名称を条件にしてフィルタリングが行われます。データポイントの行先が target.dest_ids で列挙された行先のいずれかであり、データ名称が target.name で指定されたものである場合、その行先は change_to で指定されたものに変更されます。

条件指定（target）では以下を指定します。

キー	型	説明
dest_ids	[string]	アップストリームの場合、行先ストリームの ID のリスト。 ダウンストリームの場合、行先デバイスコネクタ IPC の ID のリスト。
name	string	データ名称によるフィルター条件（正規表現）

change_to で行先を変更できます。

キー	型	説明
dest_id	string	アップストリームの場合、行先ストリームの ID。 ダウンストリームの場合、行先デバイスコネクタ IPC の ID。

例：

```
id: example
enabled: true
type: name
target:
  dest_ids:
    - recoverable
  name: v1/.*
change_to:
  dest_id: deferred
```

この例では、行先が recoverable であるデータポイントのうち、データ名称が v1/.* の正規表現にマッチするものについて、行先が deferred に変更されます。

15.3.4 src-id

src-id タイプのフィルターでは、データポイントの行先と生成元（アップストリームではデバイスコネクタ IPC、ダウンストリームではストリーム）の ID を条件にしてフィルタリングが行われます。データポイントの行先が `target.dest_ids` で列挙された行先のいずれかであり、生成元が `target.src_id` で指定されたものである場合、その行先は `change_to` で指定されたものに変更されます。

条件指定（`target`）では以下を指定します。

キー	型	説明
<code>dest_ids</code>	<code>[string]</code>	アップストリームの場合、行先ストリームの ID のリスト。 ダウンストリームの場合、行先デバイスコネクタ IPC の ID のリスト。
<code>src_id</code>	<code>string</code>	アップストリームの場合、生成元デバイスコネクタ IPC の ID（正規表現）。 ダウンストリームの場合、生成元ストリームの ID（正規表現）。

`change_to` では、以下のように新しい行先を指定します。

キー	型	説明
<code>dest_id</code>	<code>string</code>	アップストリームの場合、行先ストリームの ID。 ダウンストリームの場合、行先デバイスコネクタ IPC の ID。

例：

```
id: example
enabled: true
type: src-id
target:
  dest_ids:
    - recoverable
  src_id: dc?
change_to:
  dest_id: deferred
```

この例では、行先が `recoverable` であるデータポイントのうち、生成元の ID が `dc?` の正規表現にマッチするものについて、行先が `deferred` に変更されます。

15.3.5 rename

データポイントの行先が `target.dest_ids` で列挙された行先のいずれかであり、データ型、データ名称、生成元の ID（アップストリームではデバイスコネクタ IPC の ID、ダウンストリームではストリームの ID）が条件に一致した場合、そのデータポイントのデータ名称がリネームされます。

正規表現を空文字にした項目は、条件に一致するかどうかの評価対象にはなりません。例えば、`target.type` と `target.name` を指定し、`target.src_id`（生成元 ID）を空文字にした場合、`target.type` と `target.name` がマッチすれば、生成元 ID に関係なく行先変更が行われます。

条件指定（`target`）では以下を指定します。

キー	型	説明
dest_ids	[string]	アップストリームの場合、行先ストリームの ID のリスト。 ダウンストリームの場合、行先デバイスコネクター IPC の ID のリスト。
type	string	iSCP v2 のデータ型名によるフィルター条件（正規表現）
name	string	データ名称によるフィルター条件（正規表現）
src_id	string	生成元 ID によるフィルター条件（正規表現）

change_to では、以下のように新しい行先とデータ名称を指定します。

キー	型	説明
dest_id	string	アップストリームの場合、行先ストリームの ID。 ダウンストリームの場合、行先デバイスコネクター IPC の ID。
name	string	新しいデータ名称

例：

```
id: example
enabled: true
type: rename
target:
  dest_ids:
    - recoverable
  name: "v1/2/.+"
  type: ""
  src_id: ""
change_to:
  dest_id: deferred
  name: "v1/10/aaa"
```

この例では、行先が recoverable であるデータポイントのうち、データ名称が v1/2/.+ の正規表現にマッチするものについて、行先が deferred に変更されます。また、データ名称が v1/10/aaa に変更されます。

15.3.6 sampling

sampling タイプのフィルターでは、データポイントのサンプリングが行われます。

sampling タイプのフィルターでは、行先、データ型、データ名称、生成元（アップストリームではデバイスコネクター IPC、ダウンストリームではストリーム）の ID を条件として設定します。

指定された行先（target.dest_ids のうちいずれか）を持つデータポイントのうち、指定された正規表現すべてにマッチしたデータポイントがサンプリングの対象となります。サンプリング対象のうち、サンプリング間隔 interval_ms 内で最初の 1 つのデータポイントのみ、行先が change_to.dest_id の値に変更されます。サンプリング間隔内で 2 つ目以降のデータポイントの行先は変更されません。

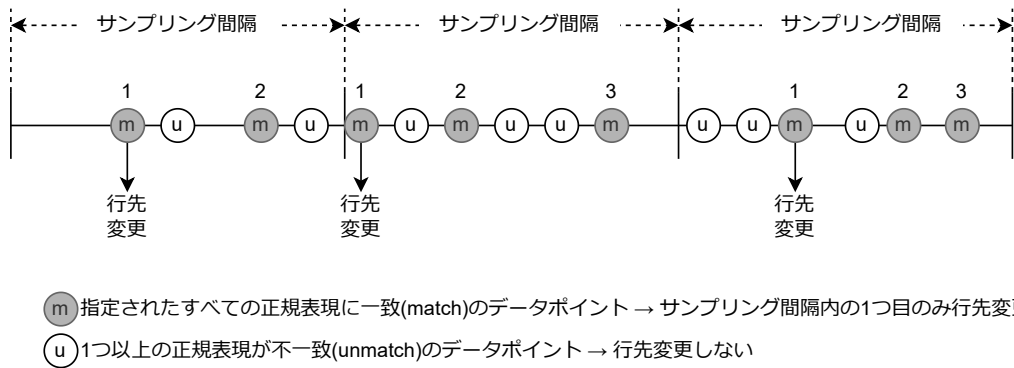


図 22 サンプリング

キー	型	説明
dest_ids	[string]	アップストリームの場合、行先ストリームの ID のリスト。 ダウンストリームの場合、行先デバイスコネクター IPC の ID のリスト。
interval_ms	integer	サンプリング間隔 (ミリ秒)
type	string	iSCIP v2 のデータ型名によるフィルター条件 (正規表現)
name	string	データ名称によるフィルター条件 (正規表現)
src_id	string	生成元 ID によるフィルター条件 (正規表現)

change_to では、以下のように新しい行先を指定します。

キー	型	説明
dest_id	string	アップストリームの場合、行先ストリームの ID。 ダウンストリームの場合、行先デバイスコネクター IPC の ID。

例：

```
id: example
enabled: true
type: sampling
target:
  dest_ids:
    - recoverable
  name: "v1/1/abc"
  type: ""
  src_id: ""
  interval_ms: 300
change_to:
  dest_id: deferred
```

この例では、行先が recoverable であり、かつデータ名称が正規表現 v1/1/abc にマッチするデータポイントのうち、300 ミリ秒に 1 つだけ、行先が deferred に変更されます。それ以外のデータポイントの行先は変更されません。

15.3.7 h264-essential-nal-units

h264-essential-nal-units タイプのフィルターは、h264_nal_unit 型専用です。

h264-essential-nal-units タイプのフィルターでは、データポイントの行先と、ペイロードの内容によってフィルタリングが行われます。データポイントの行先が `target.dest_ids` で列挙された行先のいずれかであり、ペイロードが H.264 動画の再生において重要な NAL ユニット（SPS、PPS または IDR ピクチャー）である場合、行先が変更されます。

条件指定（`target`）では以下を指定します。

キー	型	説明
<code>dest_ids</code>	<code>[string]</code>	アップストリームの場合、行先ストリームの ID のリスト。 ダウンストリームの場合、行先デバイスコネクター IPC の ID のリスト。

`change_to` では、以下のように新しい行先を指定します。

キー	型	説明
<code>dest_id</code>	<code>string</code>	アップストリームの場合、行先ストリームの ID。 ダウンストリームの場合、行先デバイスコネクター IPC の ID。

注釈: H.264 動画を送信する際に、適切に設定されたストリームとこのフィルターを組み合わせることで、Visual M2M Data Visualizer においてより少ない遅延で H.264 動画を再生できます。[H.264 NAL Unit を使って H.264 データを送信する](#) (p. 116) の使用例を参照してください。

例：

```
id: example
enabled: true
type: h264-essential-nal-units
target:
  dest_ids:
    - h264_nal_unit_extra_units
change_to:
  dest_id: h264_nal_unit_key_units
```

この例では、行先が `h264_nal_unit_extra_units` であるデータポイントのうち、再生表示に必須の NAL ユニットの行先が `h264_nal_unit_key_units` に変更されます。

16 遅延アップロード

recover: true の設定になっているアップストリームにおいてデータが送信できなかった場合、また、アップストリームとして [deferred](#) (p. 32) が指定された場合は、そのデータは intdash Edge Agent 2 内に保存され、順次遅延アップロードされます。

遅延アップロードは intdash-agentd により行われます。そのため、ネットワークに接続され、intdash-agentd が起動していれば常に遅延アップロードが行われます。

遅延アップロードによりサーバーに保存されたデータは、intdash Edge Agent 2 からは自動的に削除されます。

注釈: 遅延アップロード用のデータは、データベースファイル `/var/lib/intdash/agent.db` に保存されます。ただし環境変数 `AGENT_LIB_DIR` が設定されている場合は `${AGENT_LIB_DIR}/agent.db` が使用されます。

遅延アップロードの優先度や、遅延アップロード用のデータに使用するストレージの容量は変更することができます。

16.1 優先度を設定する

以下のようなコマンドで設定します。

```
$ intdash-agentctl config deferred -m 'priority: same_as_realtime'
```

設定できるパラメーターは以下のとおりです。

キー	型	説明
priority	string	遅延アップロードのネットワーク通信優先度 <ul style="list-style-type: none">higher_than_realtime -リアルタイム送受信よりも高い優先度で遅延アップロードを行います。遅延アップロードが帯域を占有した場合はリアルタイム送信は行われません。same_as_realtime -リアルタイム送受信と同じ優先度で遅延アップロードを行います。遅延アップロードが帯域を占有することはありませんが、遅延アップロードによりリアルタイム送受信が影響を受ける場合があります。lower_than_realtime -リアルタイム送受信よりも低い優先度で遅延アップロードを行います。リアルタイム送受信が帯域を占有した場合は遅延アップロードは行われません。

16.2 データ蓄積の上限を設定する

遅延アップロード用のデータの蓄積に上限を設定することができます。

以下のようなコマンドで設定します。

```
$ intdash-agentctl config deferred -m '  
  limit_data_storage: true  
  data_storage_capacity: 102400  
,
```

キー	型	説明
limit_data_storage	bool	古い計測の自動削除の有効（true）／無効（false）
data_storage_capacity	integer	遅延アップロード用のデータの保存のために使用するディスク容量（MiB）。データの量がこの容量を超える場合は、古いデータから順に削除されます。



警告:

- limit_data_storage を true にすると、遅延アップロード用のデータが data_storage_capacity を超えた場合に、古いデータから順に削除されます。削除されたデータを復元することはできません。
- limit_data_storage を false にすると、遅延アップロード用のデータが増加し、ディスクの使用量が 90% 以上になった場合、ストリーマーが自動的に終了し、計測が終了します。

17 ローカルストレージの管理

intdash Edge Agent 2 で計測を行うと、ローカルストレージ内のデータベースに、計測に関する情報と未送信データが保存されます。

注釈: ストリーマーを起動し終了するまでの一連の時系列データが 1 つの計測です。

intdash-agentctl measurement コマンドにより、この intdash Edge Agent 2 で行った計測についての情報を確認することができます。

```
$ intdash-agentctl measurement
  UUID                                BASE_TIME                                PENDING_DATA_SIZE
2f968138-c8d1-4277-998c-6c1d0cb15a11 2022-09-12T04:43:39.662313221Z 123
4529ef77-911b-4256-af8f-6e592ead6d86 2022-09-12T04:53:59.074454299Z 0
3f6e2c18-8275-486a-abbd-53ddadc3af6e 2022-09-12T05:21:21.819922212Z 10
```

計測は古い順に表示されます。一番下が最新の計測です。表示される情報は以下のとおりです。

キー	説明
UUID	計測の UUID。intdash サーバーで管理している計測 UUID と同じです。
BASE_TIME	計測の基準時刻（取得できた基準時刻のうち最も優先度の高い基準時刻）。
PENDING_DATA_SIZE	遅延アップロード用の送信待ちデータのサイズ (Byte)。0 の場合、遅延アップロード用のデータはありません。

17.1 計測を削除する

intdash Edge Agent 2 内に保存されている、計測に関する情報を削除するには、`--delete` オプションの引数として計測の UUID を 指定します。

重要:

- 計測を削除すると、その計測に関するすべてのデータが intdash Edge Agent 2 から削除されます。その計測にサーバーに未送信のデータがあった場合、そのデータも削除されます。
- intdash Edge Agent 2 内で計測を削除しても、サーバー側のデータは影響を受けません。

```
$ intdash-agentctl measurement --delete 4529ef77-911b-4256-af8f-6e592ead6d86
```

注釈: 実行中の計測を削除することはできません。

18 ステータスの確認

アップストリーム、ダウンストリーム、デバイスコネクター IPC、遅延アップロードの現在の状態を確認するには、`intdash-agentctl status` コマンドを実行します。

1 つのストリーム、1 つのデバイスコネクター IPC についての情報が 1 行で表示されます。また、遅延アップロードについての情報（TYPE が `deferred` となっている行）も表示されます。

```
$ intdash-agentctl status
ID            TYPE            DIRECTION  ENABLED STATUS.CODE  STATUS.ERROR
up            stream          upstream   true    connected
down         stream          downstream true    connected
up-hello     device-connector upstream   true    disconnected  uninitialized
deferred     deferred        upload     true    connected
```

列	説明
ID	ストリームまたはデバイスコネクター IPC の ID。遅延アップロードの場合は <code>deferred</code> と表示されます。
TYPE	<ul style="list-style-type: none"><code>stream</code>: ストリーム<code>device-connector</code>: デバイスコネクター IPC<code>deferred</code>: 遅延アップロード
DIRECTION	送信方向。ストリームまたはデバイスコネクター IPC については方向が表示されます（ <code>upstream</code> / <code>downstream</code> ）。遅延アップロードの場合は <code>upload</code> と表示されます。
ENABLED	設定の有効 (<code>true</code>) / 無効 (<code>false</code>)
STATUS.CODE	接続の状態が表示されます。 <ul style="list-style-type: none"><code>connected</code>: 接続<code>disconnected</code>: 接続していない<code>quiet</code>: 接続しているがデータが流れていない<code><空白></code>: 状態の情報が未取得
STATUS.ERROR	エラー発生時にエラーの概要が表示されます。

19 ログの確認

intdash Edge Agent 2 はログを出力します。intdash-agentd のログは、`/var/log/intdash-agentd.log` に出力されます。コマンドのログ、およびコマンドにより起動されるストリーマー、デバイスコネクターのログは、intdash-agentctl の標準出力に出力されます。

環境変数 `AGENT_LOG` にログレベルを設定すると、そのレベル以上のログが出力されます。設定できるログレベルは、`trace`、`debug`、`information`、`warning`、`error`、`quiet` です。`quiet` を選択すると、ログは出力されません。

注釈:

- ストリーマー、デーモン、コマンドとも、環境変数 `AGENT_LOG` を参照します。
- 環境変数はプロセス起動時に読み込まれるので、`AGENT_LOG` の設定を intdash-agentd に反映するには、デーモンを再起動する必要があります。

20 設定の書き出し／読み込み

intdash Edge Agent 2 の設定は、コマンドを使って書き出すことができます。設定をファイルに書き出すことで、設定を俯瞰したり、履歴管理したり、直接修正したりすることが可能になります。

また、書き出された設定を読み込んで適用することも可能です。

注釈: intdash Edge Agent 2 の設定は、設定ファイル `/var/lib/intdash/agent.yaml` (ただし環境変数 `AGENT_LIB_DIR` が設定されている場合は `${AGENT_LIB_DIR}/agent.yaml`) として保存されていますが、このファイルを直接編集することは避けてください。間違って正しくない形式にしまうと、最悪の場合 intdash Edge Agent 2 が正常に動作しなくなる場合があります。

設定変更は、`intdash-agentctl config` コマンドを使って行うか、以下で説明する `intdash-agentctl config-file apply` コマンドを使ってファイルを読み込むことで行ってください。これらのコマンドでは設定内容のバリデーションが行われるため、安全に設定を変更することができます。

20.1 設定を出力する

以下のコマンドを実行すると、現在の設定を標準出力に書き出すことができます。

```
$ intdash-agentctl config-file show
```

ファイルに保存したい場合は、出力をリダイレクトしてください。

設定情報の見方については、[設定一覧](#) (p. 99) を参照してください。

20.2 ファイルから設定を読み込む

[上記の手順](#) (p. 84) で出力した設定は、以下の手順で読み込むことができます。

1. `intdash-agentctl run` を実行している場合 (計測を実行中の場合) は `Ctrl+C` を押して停止します。
2. 以下のコマンドを実行します。

```
$ intdash-agentctl config-file apply -c <path_to_conf_file>
```

21 intdash Edge Agent 2 REST API

intdash Edge Agent 2 は REST API を持っており、API へのリクエストにより設定の確認や変更を行うことができます。

REST API の仕様については以下を参照してください。

- [Agent Command API](#)

REST API を使ってアクセスが可能な設定は以下のとおりです。

21.1 接続先サーバーと認証情報

接続先サーバーと認証情報に関する設定には /connection エンドポイントを使用します。

API リファレンス : [Connection to Server](#)

21.2 トランスポート

トランスポートの設定には /transport エンドポイントを使用します。

API リファレンス : [Connection to Server](#)

21.3 ストリーム

ストリームに関する設定には /upstreams と /downstreams エンドポイントを使用します。

API リファレンス : [Stream](#)

21.4 フィルター

ストリーマーを流れるデータに対するフィルターの設定には /filters_upstream と /filters_downstream エンドポイントを使用します。

API リファレンス : [Filter](#)

21.5 デバイスコネクター IPC

デバイスコネクター IPC に関する設定には /device_connectors_upstream と /device_connectors_downstream エンドポイントを使用します。

API リファレンス : [Device Connector IPC](#)

21.6 遅延アップロード

未送信データの遅延アップロードに関する設定には `/deferred_upload` エンドポイントを使用します。

API リファレンス : [Deferred Upload](#)

21.7 ローカル計測データ

ローカルストレージ内に保存されている計測データの管理には `/measurements` エンドポイントを使用します。

API リファレンス : [Measurement](#)

22 intdash-agentctl コマンド

intdash-agentctl コマンドは以下の書式で使⽤します。

```
intdash-agentctl [--version | -v]
intdash-agentctl [--help | -h]
intdash-agentctl <command> [<command_options>] [<arguments>]
```

--version, -v

バージョンを表示します。

--help, -h

ヘルプを表示します。

<command>ごとの使⽤方法については、以下を参照してください。

注釈: コマンドや引数には、省略形が用意されている場合があります。例えば、`intdash-agentctl config upstream --list` は、`intdash-agentctl c u -l` のように省略することができます。
使⽤できる省略形を確認するには、各コマンドに `--help` オプションを付けて実行してください。

22.1 intdash-agentctl run

```
intdash-agentctl run [<run_command_options>]
```

ストリーマーを起動し、計測を開始します。

<run_command_options> では、以下を⽤することができ⽬す。

--log <level>, -l <level>

ログレベルを設定します。level には、以下を設定できます: `t[race]` | `d[ebug]` | `i[nformation]` | `w[arn-
ing]` | `e[rror]` | `q[uiet]`

ログレベルは、環境変数 `AGENT_LOG` でも設定できます。環境変数 `AGENT_LOG` よりもこのコマンドオプションが優先されます。環境変数 `AGENT_LOG` が与えられておらず、このコマンドオプションも与えられていない場合、ログレベルは `information` となります。

--address <value>

intdash-agentd の gRPC サーバーのアドレスを指定します。通常は、デフォルトのままでよいため指定する必要はありません。(デフォルト: `localhost:50051`)

--name <name>, -n <name>

計測の名前を設定します。

--description <description>, -d <description>

計測の説明を設定します。

--timeout <seconds>, -t <seconds>

計測終了までの秒数を指定します。指定しない場合、`ctrl+C` で終了するまで継続します。

--help, -h

ヘルプを表示します。

22.2 intdash-agentctl config

```
intdash-agentctl config [<config_command_options>] <command> <command_option> [<arguments>]
```

intdash Edge Agent 2 の設定を変更します。

<command> では、connection、upstream、filter upstream などの、設定対象を示すサブコマンドを指定します。詳細については、設定対象ごとの説明を参照してください。

注釈: --modify <config_string> -m <config_string> --patch <config_string> を使用する場合、<config_string> で指定されたフィールドのみが更新され、他のフィールドは変更されません。

削除できるフィールドの場合は、null を指定することで削除できます。

```
intdash-agentctl config device-connector upstream --modify up-hello '
  launch: null
'
```

config コマンドに共通の <config_command_options> では、以下を使用することができます。

--log <level>, -l <level>

ログレベルを設定します。level には、以下を設定できます: t[race]|d[ebug]|i[nformation]|w[arning]|e[rrror]|q[uiet]

ログレベルは、環境変数 AGENT_LOG でも設定できます。環境変数 AGENT_LOG よりもこのコマンドオプションが優先されます。環境変数 AGENT_LOG が与えられておらず、このコマンドオプションも与えられていない場合、information レベルとなります。

--json, -j

設定の入出力を JSON 形式で行います。

--help, -h

ヘルプを表示します。

注釈: intdash-agentctl config コマンドで設定の取得、変更、削除などを実行すると、intdash Edge Agent 2 が持つ REST API に対してリクエストが行われ、設定の取得、変更、削除などが行われます。

22.2.1 connection サブコマンド

```
intdash-agentctl config [<config_command_options>] connection <command_option> [<arguments>]
```

intdash Edge Agent 2 と intdash サーバー間の接続について設定します。

設定操作

<command_option> [<arguments>] で以下のように指定します。

--get, -g

接続の設定を表示します。

--modify <config_string>, -m <config_string>, --patch <config_string>

接続の設定を更新します。<config_string> には YAML 形式の文字列 (--json を指定した場合は JSON 型式) を与えます。

例:


```
$ intdash-agentctl config connection --modify '
  server_url: https://abc.example.com
  project_uuid: 01234567-89ab-cdef-0123-456789abcdef
'
```

注釈:

- connection サブコマンドでは、REST API の /connection エンドポイントが使用されます。設定の詳細については、API リファレンスの [Update Connection Settings](#) を参照してください。
- <config_string> では、API リファレンスに記載されているリクエストボディの JSON と同等の内容を YAML 形式で指定してください（または --json オプションを使用して JSON 形式で指定してください）。

22.2.2 transport サブコマンド

```
intdash-agentctl config [<config_command_options>] transport <command_option> [<arguments>]
```

トランスポートについて設定します。

設定操作

<command_option> [<arguments>] で以下のように指定します。

--get, -g

トランスポートの設定を表示します。

--modify <config_string>, -m <config_string>, --patch <config_string>

トランスポートの設定を更新します。<config_string>には YAML 形式の文字列（--json を指定した場合は JSON 型式）を与えます。<config_string>で指定されたフィールドのみが更新され、他のフィールドは変更されません。

例:

```
$ intdash-agentctl config transport --modify 'protocol: websocket'
```

注釈:

- transport サブコマンドでは、REST API の /transport エンドポイントが使用されます。設定の詳細については、API リファレンスの [Update Transport Settings](#) を参照してください。
- <config_string> では、API リファレンスに記載されているリクエストボディの JSON と同等の内容を YAML 形式で指定してください（または --json オプションを使用して JSON 形式で指定してください）。

22.2.3 upstream、downstream サブコマンド

```
intdash-agentctl config [<config_command_options>] upstream <command_option> [<arguments>]  
intdash-agentctl config [<config_command_options>] downstream <command_option> [<arguments>]
```

アップストリームまたはダウンストリームについて設定します。

設定操作

<command_option> [<arguments>] で以下のように指定します。

--create <config_string>, -c <config_string>, --post <config_string>

アップストリームまたはダウンストリームの設定を新規作成します。<config_string>には YAML 形式の文字列 (--json を指定した場合は JSON 型式) を与えます。指定しなかった設定値はデフォルトが使用されます。

例:

```
$ intdash-agentctl config upstream --create '  
  id: recoverable  
  enabled: true  
  recover: true  
  persist: true  
  qos: unreliable  
  flush_policy: interval  
  flush_interval: 5  
'
```

--list, -l

アップストリーム／ダウンストリーム設定の一覧を表示します。

--get <id>, -g <id>

指定された ID のアップストリーム／ダウンストリームの設定を表示します。

例:

```
$ intdash-agentctl config upstream --get recoverable
```

--modify <id> <config_string>, -m <id> <config_string>, --patch <id> <config_string>

指定された ID のアップストリーム／ダウンストリームの設定を更新します。<config_string>には YAML 形式の文字列 (--json を指定した場合は JSON 型式) を与えます。<config_string>で指定されたフィールドのみが更新され、他のフィールドは変更されません。

例:

```
$ intdash-agentctl config upstream --modify recoverable 'enabled: false'
```

--delete <id>, -d <id>

指定されたストリームを削除します。引数としてアップストリーム／ダウンストリームの ID を指定します。

注釈:

- upstream サブコマンドでは、REST API の /upstreams エンドポイントが使用されます。設定の詳細については、API リファレンスの [Create Upstream Settings](#) を参照してください。
- downstream サブコマンドでは、REST API の /downstreams エンドポイントが使用されます。設定の詳細については、API リファレンスの [Create Downstream Settings](#) を参照してください。
- <config_string> では、API リファレンスに記載されているリクエストボディの JSON と同等の内容を YAML 形式で指定してください（または --json オプションを使用して JSON 形式で指定してください）。

22.2.4 filter サブコマンド

```
intdash-agentctl config [<config_command_options>] filter upstream <command_option> [<arguments>]  
intdash-agentctl config [<config_command_options>] filter downstream <command_option> [<arguments>]
```

フィルターについて設定します。

設定操作

<command_option> [<arguments>] で以下のように指定します。

--create <config_string>, -c <config_string>, --post <config_string>

フィルターの設定を新規作成します。<config_string>には YAML 形式の文字列（--json を指定した場合は JSON 型式）を与えます。指定しなかった設定値はデフォルトが使用されます。

例

```
$ intdash-agentctl config filter upstream --create '  
  id: example  
  enabled: true  
  type: name  
  target:  
    dest_ids:  
      - recoverable  
  name: v1/.+  
  change_to:  
    dest_id: deferred  
'
```

--list, -l

フィルター設定の一覧を表示します。

--get <id>, -g <id>

指定されたフィルターを表示します。引数としてフィルターの ID を指定します。

--modify <id> <config_string>, -m <id> <config_string>, --patch <id> <config_string>

指定された ID のフィルター設定を更新します。<config_string>には YAML 形式の文字列（--json を指定した場合は JSON 型式）を与えます。<config_string>で指定されたフィールドのみが更新され、他のフィールドは変更されません。

例:

```
$ intdash-agentctl config filter upstream --modify data_sampling 'enabled: false'
```

--delete <id>, -d <id>

指定されたフィルター設定を削除します。

注釈:

- filter upstream サブコマンドでは、REST API の /filters_upstream エンドポイントが使用されます。設定の詳細については、API リファレンスの [Create Filter for Upstream](#) を参照してください。
- filter downstream サブコマンドでは、REST API の /filters_downstream エンドポイントが使用されます。設定の詳細については、API リファレンス [Create Filter for Downstream](#) を参照してください。
- <config_string> では、API リファレンスに記載されているリクエストボディの JSON と同等の内容を YAML 形式で指定してください（または --json オプションを使用して JSON 形式で指定してください）。

22.2.5 device-connector コマンド

```
intdash-agentctl config [<config_command_options>] device-connector upstream <command_option> [<arguments>]  
intdash-agentctl config [<config_command_options>] device-connector downstream <command_option> [<arguments>]
```

デバイスコネクター IPC について設定します。

設定操作

<command_option> [<arguments>] で以下のように指定します。

--create <config_string>, -c <config_string>, --post <config_string>

デバイスコネクター IPC 設定を新規作成します。<config_string>には YAML 形式の文字列（--json を指定した場合は JSON 型式）を与えます。指定しなかった設定値はデフォルトが使用されます。

例

```
$ intdash-agentctl config device-connector upstream --create '  
  id: dc1  
  data_name: group/subgroup  
  dest_ids:  
  - recoverable  
  enabled: true  
  data_name_prefix: v1/1  
  format: iscp-v2-compatible  
  ipc:  
    path: /var/run/intdash/device_connector.fifo  
    type: fifo  
  launch:  
    args:  
    - --config  
    cmd: device-connector-intdash  
    environment:  
    - KEY=value  
'
```

--list, -l

デバイスコネクター IPC 設定の一覧を表示します。

--get <id>

指定されたデバイスコネクター IPC を表示します。

--modify <id> <config_string>, -m <id> <config_string>, --patch <id> <config_string>

指定された ID のデバイスコネクター IPC 設定を更新します。<config_string>には YAML 形式の文字列（--json を指定した場合は JSON 型式）を与えます。<config_string>で指定されたフィールドのみが更

新され、他のフィールドは変更されません。

例:

```
$ intdash-agentctl config device-connector upstream --modify dc1 'enabled: false'
```

--delete <id>, -d <id>

指定された ID のデバイスコネクタ IPC を削除します。

注釈:

- device-connector upstream サブコマンドでは、REST API の /device_connectors_upstream エンドポイントが使用されます。設定の詳細については、API リファレンスの [Create Device Connector IPC Settings for Upstream](#) を参照してください。
- device-connector downstream サブコマンドでは、REST API の /device_connectors_downstream エンドポイントが使用されます。設定の詳細については、API リファレンス [Create Device Connector IPC Settings for Downstream](#) を参照してください。
- <config_string> では、API リファレンスに記載されているリクエストボディの JSON と同等の内容を YAML 形式で指定してください（または --json オプションを使用して JSON 形式で指定してください）。

22.2.6 deferred-upload コマンド

```
intdash-agentctl config [<config_command_options>] deferred-upload <command_option> [<arguments>]
```

遅延アップロードについて設定します。

設定操作

<command_option> [<arguments>] で以下のように指定します。

--get, -g

遅延アップロードの設定を表示します。

--modify <config_string>, -m <config_string>, --patch <config_string>

遅延アップロードの設定を更新します。<config_string>には YAML 形式の文字列（--json を指定した場合は JSON 型式）を与えます。<config_string>で指定されたフィールドのみが更新され、他のフィールドは変更されません。

例:

```
$ intdash-agentctl config deferred-upload --modify 'priority: higher_than_realtime'
```

注釈:

- deferred-upload サブコマンドでは、REST API の /deferred_upload エンドポイントが使用されます。設定の詳細については、API リファレンスの [Update Deferred Upload Settings](#) を参照してください。
- <config_string> では、API リファレンスに記載されているリクエストボディの JSON と同等の内容を YAML 形式で指定してください（または --json オプションを使用して JSON 形式で指定してください）。

22.3 intdash-agentctl config-file

```
intdash-agentctl config-file [<config_file_command_options>] <command> <command_option> [<arguments>]
```

config コマンドに共通の <config_file_command_options> では、以下を使用することができます。

--log <level>, -l <level>

ログレベルを設定します。level には、以下を設定できます: t[race]|d[ebug]|i[nformation]|w[arning]|e[rror]|q[uiet]

ログレベルは、環境変数 AGENT_LOG でも設定できます。環境変数 AGENT_LOG よりもこのコマンドオプションが優先されます。環境変数 AGENT_LOG が与えられておらず、このコマンドオプションも与えられていない場合、information となります。

--help, -h

ヘルプを表示します。

22.3.1 show サブコマンド

```
intdash-agentctl config-file [<config_file_command_options>] show [<command_options>]
```

intdash Edge Agent 2 の現在の設定を標準出力に書き出します。

[<command_options>] で以下のように指定します。

--default

現在の設定ではなくデフォルトの設定を表示します。

--help, -h

ヘルプを表示します。

22.3.2 apply サブコマンド

```
intdash-agentctl config-file [<config_file_command_options>] apply <command_options>
```

指定されたファイルから設定を読み込み、適用します。

また、読み込まれた設定は現在使用中の設定ファイル（デフォルトでは /var/lib/intdash/agent.yaml、ただし環境変数 AGENT_LIB_DIR が設定されている場合は \${AGENT_LIB_DIR}/agent.yaml）に書き込まれます。

<command_options> で以下のように指定します。

--user-agent-config <value>, -c <value> (必須)

使用する設定ファイルを指定します。

--help, -h

ヘルプを表示します。

22.4 intdash-agentctl measurement

```
intdash-agentctl measurement [<measurement_command_options>] [<arguments>]
```

intdash Edge Agent 2 内にある計測のリストを古い順に表示します（一番下が最も新しいもの）。`--delete` オプションを使用することで、intdash Edge Agent 2 内のその計測に関するデータ（遅延アップロード用の送信待ちデータ含む）を削除することができます。

<measurement_command_options> では、以下を使用することができます。

--log <level>, -l <level>

ログレベルを設定します。level には、以下を設定できます: `t[race]` | `d[ebug]` | `i[nformation]` | `w[arning]` | `e[rror]` | `q[uiet]`

ログレベルは、環境変数 `AGENT_LOG` でも設定できます。環境変数 `AGENT_LOG` よりもこのコマンドオプションが優先されます。環境変数 `AGENT_LOG` が与えられておらず、このコマンドオプションも与えられていない場合、`information` となります。

--address <value>

intdash-agentd の gRPC サーバーのアドレスを指定します。通常は、デフォルトのままでよいため指定する必要はありません。(デフォルト: `localhost:50051`)

--json, -j

計測のリストを JSON 型式で表示します。

--delete, -d

intdash Edge Agent 2 内の計測データを削除します。引数として、削除したい計測の UUID を指定します。

--help, -h

ヘルプを表示します。

22.5 intdash-agentctl status

```
intdash-agentctl status [<status_command_options>]
```

アップストリーム、ダウストリーム、遅延アップロードの現在の状態を表示します。

<status_command_options> では、以下を使用することができます。

--log <level>, -l <level>

ログレベルを設定します。level には、以下を設定できます: `t[race]` | `d[ebug]` | `i[nformation]` | `w[arning]` | `e[rror]` | `q[uiet]`

ログレベルは、環境変数 `AGENT_LOG` でも設定できます。環境変数 `AGENT_LOG` よりもこのコマンドオプションが優先されます。環境変数 `AGENT_LOG` が与えられておらず、このコマンドオプションも与えられていない場合、`information` となります。

--json, -j

ステータスのリストを JSON 型式で表示します。

--help, -h

ヘルプを表示します。

22.6 intdash-agentctl ping

```
intdash-agentctl ping [<ping_command_options>]
```

intdash-agentd が正常に動作しているかどうかを確認します。

<ping_command_options> では、以下を使用することができます。

--log <level>, -l <level>

ログレベルを設定します。level には、以下を設定できます: t[race]|d[ebug]|i[nformation]|w[arning]|e[rrror]|q[uiet]

ログレベルは、環境変数 AGENT_LOG でも設定できます。環境変数 AGENT_LOG よりもこのコマンドオプションが優先されます。環境変数 AGENT_LOG が与えられておらず、このコマンドオプションも与えられていない場合、information となります。

--address <value>

intdash-agentd の gRPC サーバーのアドレスを指定します。通常は、デフォルトのままでよいため指定する必要はありません。(デフォルト: localhost:50051)

--timeout <value>, -t <value>

確認のタイムアウトを指定します (単位: 秒)。(デフォルト: 5)

--help, -h

ヘルプを表示します。

22.7 intdash-agentctl about

```
intdash-agentctl about [<command_options>]
```

バージョン、各プログラムのバージョン、intdash Edge Agent 2 用の環境変数の値を表示します。

<about_command_options> では、以下を使用することができます。

--help, -h

ヘルプを表示します。

22.8 intdash-agentctl help

```
intdash-agentctl help
```

intdash-agentctl コマンドの使い方を表示します。

23 intdash-agentd コマンド

注釈: intdash-agentd は、intdash Edge Agent 2 のデーモンとして使用されるプログラムです。通常、intdash-agentd の起動や終了は init スクリプトを使って行うので直接 intdash-agentd コマンドを実行する必要はありません。

intdash-agentd コマンドは以下の書式で使います。

```
intdash-agentd [--version | -v]
intdash-agentd [--help | -h]
intdash-agentd <command> [<command_options>] [<arguments>]
```

--version, -v

バージョンを表示します。

--help, -h

ヘルプを表示します。

<command>ごとの使用方法については、以下を参照してください。

23.1 intdash-agentd serve

```
intdash-agentd serve [<serve_command_options>] <arguments>
```

intdash-agentd を起動します。

注釈: 通常、intdash-agentd の起動は init スクリプトを使って行うため、直接 intdash-agentd serve コマンドを実行する必要はありません。

[<serve_command_options>] では、以下を使用することができます。

--log <level>, -l <level>

ログレベルを設定します。level には、以下を設定できます: t[race]|d[ebug]|i[nformation]|w[arning]|e[rror]|q[uiet]

ログレベルは、環境変数 AGENT_LOG でも設定できます。環境変数 AGENT_LOG よりもこのコマンドオプションが優先されます。環境変数 AGENT_LOG が与えられておらず、このコマンドオプションも与えられていない場合、information レベルとなります。

--port <value>

intdash-agentd の gRPC サーバーのポート番号を指定します。intdash-agentctl コマンドから計測情報にアクセスするために利用されます。通常は、デフォルトのままでよいため指定する必要はありません。(デフォルト: 50051)

--uploader-request-size <value>

intdash-agentd から intdash サーバーに向けて遅延アップロードを行う際の、1 リクエストに含めるチャンクの最大サイズを指定します (単位: バイト)。(デフォルト: 1048576)

--help, -h

ヘルプを表示します。

23.2 intdash-agentd help

```
intdash-agentd help
```

intdash-agentd コマンドの使い方を表示します。

24 設定一覧

intdash-agentctl config-file show コマンドにより出力することができる、intdash Edge Agent 2 の全設定は以下のとおりです。

注釈:

- 設定を出力する方法については [設定の書き出し／読み込み](#) (p. 84) を参照してください。
- 設定可能な項目は、REST API の各エンドポイントと同じです。REST API のメッセージボディは JSON ですが、[設定の書き出し／読み込み](#) (p. 84) では同内容の YAML が使用されます。

設定は以下の部分に大別されます。

キー	型	説明
connection (p. 101)	object	接続先サーバーと認証情報の設定
transport (p. 101)	object	トランスポートの設定
upstream (p. 102)	object	アップストリームの設定
downstream (p. 102)	object	ダウンストリームの設定
filters_upstream (p. 103)	[object]	上り方向のフィルター設定のリスト
filters_downstream (p. 103)	[object]	下り方向のフィルター設定のリスト
device_connectors_upstream (p. 104)	[object]	上り方向のデバイスコネクタ IPC 設定のリスト
device_connectors_downstream (p. 104)	[object]	下り方向のデバイスコネクタ IPC 設定のリスト
deferred_upload (p. 105)	object	未送信データの遅延アップロードに関する設定

24.1 設定の例

```
connection:
  server_url: https://xxxxx.intdash.jp
  project_uuid: c48e3eee-0242-462f-xxxx-xxxxxxxxxxxx
  edge_uuid: 03ace3b1-d208-4fc3-xxxx-xxxxxxxxxxxx
  client_secret: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
transport:
  protocol: quic
upstream:
- id: recoverable
  enabled: true
  recover: true
  persist: true
  qos: unreliable
  flush_policy: interval
  flush_interval: 5
- id: h264_nal_unit_key_units
  enabled: true
  recover: true
  persist: true
  qos: partial
  flush_policy: immediately
  flush_interval: 5
- id: h264_nal_unit_extra_units
  enabled: true
```

(次のページに続く)

(前のページからの続き)

```
recover: true
persist: true
qos: unreliable
flush_policy: immediately
flush_interval: 5
downstream:
- id: down
  enabled: true
  dest_ids:
  - down-hello
  qos: unreliable
  filters:
  - src_edge_uuid: 03ace3b1-d208-xxxx-xxxxxxxxxxxx
    data_filters:
    - type: string
      name: v1/1/ab
device_connectors_upstream:
- id: up-hello
  data_name_prefix: v1/1/
  dest_ids:
  - recoverable
  enabled: true
  format: iscp-v2-compatible
  ipc:
    type: fifo
    path: /var/run/intdash/up-hello.fifo
  launch:
    cmd: device-connector-intdash
    args:
    - --config
    - /tmp/dc-hello.yaml
- id: h264_nal_unit
  data_name_prefix: 101/h264_nal_unit
  dest_ids:
  - recoverable
  enabled: true
  format: iscp-v2-compatible
  ipc:
    type: fifo
    path: /var/run/intdash/up-h264_nal.fifo
device_connectors_downstream:
- id: down-hello
  enabled: true
  format: iscp-v2-compatible
  ipc:
    type: fifo
    path: /var/run/intdash/down-hello.fifo
filters_upstream:
- id: string_to_deferred
  enabled: true
  type: type
  target:
    dest_ids:
    - recoverable
  type: string
  change_to:
    dest_id: deferred
```

(次のページに続く)

(前のページからの続き)

```
- id: h264_nal_unit_filter
  enabled: true
  type: type
  target:
    dest_ids:
      - recoverable
    type: h264_nal_unit
  change_to:
    dest_id: h264_nal_unit_extra_units
- id: h264_nal_unit_key_filter
  enabled: true
  type: h264-essential-nal-units
  target:
    dest_ids:
      - h264_nal_unit_extra_units
  change_to:
    dest_id: h264_nal_unit_key_units
filters_downstream: []
deferred_upload:
  priority: same_as_realtime
  limit_data_storage: true
  data_storage_capacity: 102400
```

24.2 connection

設定例：

```
connection:
  server_url: https://xxxxx.intdash.jp
  project_uuid: c48e3eee-0242-462f-xxxx-xxxxxxxxxxxxx
  edge_uuid: 03ace3b1-d208-4fc3-xxxx-xxxxxxxxxxxxx
  client_secret: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

注釈: 設定可能な項目は、REST API の /connection エンドポイントと同じです。設定の詳細については、API リファレンスの [Update Connection Settings](#) を参照してください。

24.3 transport

設定例：

```
transport:
  protocol: quic
```

注釈: 設定可能な項目は、REST API の /transport エンドポイントと同じです。設定の詳細については、API リファレンスの [Update Transport Settings](#) を参照してください。

24.4 upstream

設定例：

```
upstream:
- id: recoverable
  enabled: true
  recover: true
  persist: true
  qos: unreliable
  flush_policy: interval
  flush_interval: 5
- id: h264_nal_unit_key_units
  enabled: true
  recover: true
  persist: true
  qos: partial
  flush_policy: immediately
  flush_interval: 5
- id: h264_nal_unit_extra_units
  enabled: true
  recover: true
  persist: true
  qos: unreliable
  flush_policy: immediately
  flush_interval: 5
```

注釈: 設定可能な項目は、REST API の /upstreams エンドポイントと同じです。設定の詳細については、API リファレンスの [Create Upstream Settings](#) を参照してください。

24.5 downstream

設定例：

```
downstream:
- id: down
  enabled: true
  dest_ids:
  - down-hello
  qos: unreliable
  filters:
  - src_edge_uuid: 03ace3b1-d208-4fc3-xxxx-xxxxxxxxxxxxx
    data_filters:
    - type: string
      name: v1/1/ab
```

データ ID のワイルドカード指定を含む、データ ID の詳細については iSCP の仕様を参照してください。

注釈: 設定可能な項目は、REST API の /downstreams エンドポイントと同じです。設定の詳細については、API リファレンスの [Create Downstream Settings](#) を参照してください。

24.6 filters_upstream

設定例：

```
filters_upstream:
- id: string_to_deferred
  enabled: true
  type: type
  target:
    dest_ids:
      - recoverable
    type: string
  change_to:
    dest_id: deferred
- id: h264_nal_unit_filter
  enabled: true
  type: type
  target:
    dest_ids:
      - recoverable
    type: h264_nal_unit
  change_to:
    dest_id: h264_nal_unit_extra_units
- id: h264_nal_unit_key_filter
  enabled: true
  type: h264-essential-nal-units
  target:
    dest_ids:
      - h264_nal_unit_extra_units
  change_to:
    dest_id: h264_nal_unit_key_units
```

注釈: 設定可能な項目は、REST API の `/filters_upstream` エンドポイントと同じです。設定の詳細については、API リファレンスの [Create Filter for Upstream](#) を参照してください。

24.7 filters_downstream

設定例：

```
filters_downstream:
- id: string_to_dc2
  enabled: true
  type: type
  target:
    dest_ids:
      - recoverable
    type: string
  change_to:
    dest_id: deferred
```

注釈: 設定可能な項目は、REST API の `/filters_downstream` エンドポイントと同じです。設定の詳細については、API リファレンス [Create Filter for Downstream](#) を参照してください。

24.8 device_connectors_upstream

設定例：

```
device_connectors_upstream
- id: up-hello
  enabled: true
  data_name_prefix: v1/1/
  format: iscp-v2-compatible
  ipc:
    type: fifo
    path: /var/run/intdash/up-hello.fifo
  launch:
    cmd: device-connector-intdash
    args:
      - --config
      - /tmp/dc-hello.yaml
    environment:
      - TEST_VAR=1
```

注釈: 設定可能な項目は、REST API の /device_connectors_upstream エンドポイントと同じです。設定の詳細については、API リファレンスの [Create Device Connector IPC Settings for Upstream](#) を参照してください。

24.9 device_connectors_downstream

設定例：

```
device_connectors_downstream
- id: down-hello
  enabled: true
  format: iscp-v2-compatible
  ipc:
    type: fifo
    path: /var/run/intdash/down-hello.fifo
  launch:
    cmd: device-connector-intdash
    args:
      - --config
      - /tmp/down-hello.yaml
    environment:
      - TEST_VAR=1
```

注釈: 設定可能な項目は、REST API の /device_connectors_downstream エンドポイントと同じです。設定の詳細については、API リファレンス [Create Device Connector IPC Settings for Downstream](#) を参照してください。

24.10 deferred_upload

設定例：

```
deferred_upload:  
  priority: same_as_realtime  
  limit_data_storage: true  
  data_storage_capacity: 102400
```

注釈: 設定可能な項目は、REST API の /deferred_upload エンドポイントと同じです。設定の詳細については、API リファレンスの [Update Deferred Upload Settings](#) を参照してください。

25 付与されるデータ ID

intdash Edge Agent 2 がデバイスコネクタから受け取ったデータポイントには、intdash サーバーへ送信される際に、iSCP 仕様に沿ったデータ ID が付与されます。

25.1 データポイントへの ID 付与

注釈: ここでは、アップストリームの場合を例にして、FIFO 用データフォーマット→iSCP の方向で説明しますが、ダウンストリームの場合も対応関係は同様です。

iSCP のデータ ID は、データ型とデータ名称により構成されます。デバイスコネクタからデータを取得すると以下のようにデータ ID が付与されます（推奨される iscp-v2-compatible 形式の FIFO 用データフォーマットを使用する場合）。

- iSCP のデータ型としては、FIFO 用データフォーマット内の Data Type フィールドの値が使用されます。
- iSCP のデータ名称としては、デバイスコネクタ IPC 設定に含まれる `data_name_prefix` と、FIFO 用データフォーマットに含まれる Data Name を / で結合したものが使用されます。

例えば、デバイスコネクタ IPC 設定の `data_name_prefix` として `abc` が設定され、FIFO 用データフォーマット内の Data Type フィールドが `string`、Data Name が `def` だった場合、iSCP でのデータ型は `string`、データ名称は `abc/def` となります。

注釈: FIFO 用データフォーマットとして logger-msg 形式（非推奨）を使用する場合、上記とは異なります。

- データ型は、logger-msg 形式のデータ型に一对一で対応する所定の iSCP のデータ型として扱われます。logger-msg のデータ型と iSCP のデータ型の対応については以下の表を参照してください。
- データ名称は、ストリーマーにて以下の対応表のように生成されます（logger-msg フォーマットには Data Name フィールドがないため）。NMEA、CAN/CAN-FD、String 等ではペイロード内の情報を使ってデータ名称が生成されます。

logger-msg のデータタイプ	付与される iSCP のデータ ID	
	データ型	データ名称
NMEA (0x10)	string/nmea	<data_name_prefix>/<トーカー>/<メッセージ> ペイロード内の NMEA センテンス（例:「\$GNRMC,...」）から、2～3 文字目のトーカー（例:「GN」）と、4～6 文字目のメッセージ（例:「RMC」）が抽出されます。
CAN/CAN-FD (0x11)	can_frame	<data_name_prefix>/<CAN ID> ペイロード内の CAN フレームから CAN ID が抽出されます。
JPEG (0x12)	jpeg	<data_name_prefix>
H264 (0x1C)	h264_annex_b	<data_name_prefix>
String (0x1D)	string	<data_name_prefix>/<logger-msg の ID フィールド>
Float (0x1E)	float64	<data_name_prefix>/<logger-msg の ID フィールド>
Int (0x1F)	int64	<data_name_prefix>/<logger-msg の ID フィールド>
Bytes (0x20)	bytes	<data_name_prefix>/<logger-msg の ID フィールド>
PCM (0x22)	pcm	<data_name_prefix>

25.2 iSCP v1 互換のデータ ID

iSCP v2 と iSCP v1 とでは、データを特定する方法が異なります。iSCP v1 にはチャンネルがありますが、v2 にはチャンネルがありません。また、データ ID の仕組みも異なります。

しかし、iSCP v2 でアップストリームを行う際に以下のようなデータ名称を付与してサーバーに送信すると、サーバー側の仕組みにより、そのデータは iSCP v1 でダウンストリームすることができます。

iSCP v2 のデータ型	iSCP v1 と互換性を持たせるためのデータ名称	データ名称例
string/nmea	v1/<チャンネル>/<トーカ><メッセージ>	v1/1/GPRMC
can_frame	v1/<チャンネル>/<CAN ID (最上位ビットは IDE)>	v1/1/80000001
jpeg	v1/<チャンネル>/jpeg	v1/1/jpeg
h264_annex_b	v1/<チャンネル>/<iSCP v1 の H.264 データのタイプ>	v1/1/01
string	v1/<チャンネル>/<iSCP v1 のデータ ID>	v1/1/hello
float64	v1/<チャンネル>/<iSCP v1 のデータ ID>	v1/1/hello
int64	v1/<チャンネル>/<iSCP v1 のデータ ID>	v1/1/hello
bytes	v1/<チャンネル>/<iSCP v1 のデータ ID>	v1/1/hello
pcm	v1/<チャンネル>/pcm	v1/1/pcm

<チャンネル> は、10 進数 0～255 で指定してください。

注釈: intdash サーバーは、データ名称の最初が v1/<10 進数 0～255> になっている場合、10 進数の部分を iSCP v1 用のチャンネル番号として解釈します。また、v1/<10 進数 0～255>/ の後の文字列を、データ型に応じて、トーカ、メッセージ、CAN ID、または iSCP v1 のデータ ID として解釈します。

25.3 iSCP v1 互換のデータ名称

intdash Edge Agent 2 において、上の表のように iSCP v1 と互換性を持たせるためのデータ名称を与えるためには、デバイスコネクタ IPC 設定で、data_name_prefix を v1/<チャンネル> としてください。その後ろの部分については、以下のとおりです。

重要: ただし、iscp-v2-compatible 形式で h264_nal_unit 型のデータを送信したい場合は、data_name_prefix を <チャンネル番号> とし（v1/ は不要）、iscp-v2-compatible の Data Name フィールドを h264_nal_unit としてください。h264_nal_unit 型は iSCP v1 に存在しないデータ型であるため、特別な指定方法となっています。

25.3.1 iscp-v2-compatible 使用時

string/nmea、can_frame、jpeg、h264_annex_b、pcm 型の場合、data_name_prefix と、iscp-v2-compatible フォーマットの Data Name フィールドを /（スラッシュ）で結合した文字列の先頭が v1/<チャンネル> であるとき¹、ストリーマーにて以下のように特別な処理が行われます。

string/nmea 型

ペイロード内の NMEA センテンス（例:「\$GNRMC,...」）から 2～3 文字目のトーカ（例:「GN」）と、4～6 文字目のメッセージ（例:「RMC」）が抽出され、データ名称 v1/<チャンネル>/<トーカ><メッセージ> が付与されます。

can_frame 型

ペイロード内から CAN ID が抽出され、データ名称 v1/<チャンネル>/<CAN ID (最上位ビットは IDE)> が付与されます。

jpeg 型

常にデータ名称 v1/<チャンネル>/jpeg が付与されます。

¹ 以下の場合はいずれも該当します。

- data_name_prefix が v1/1 で、iscp-v2-compatible フォーマットの Data Name フィールドが abc である場合 → v1/1/abc
- data_name_prefix が v1 で、iscp-v2-compatible フォーマットの Data Name フィールドが 1 である場合 → v1/1
- data_name_prefix が v1 で、iscp-v2-compatible フォーマットの Data Name フィールドが 1/abc である場合 → v1/1/abc

h264_annex_b 型

ペイロード内から H.264 データのタイプが抽出され、データ名称 v1/<チャンネル>/<iSCP v1 の H.264 データのタイプ> が付与されます。

pcm 型

常にデータ名称 v1/<チャンネル>/pcm が付与されます。

string、float64、int64、bytes 型の場合、ストリーマーでの特別な処理は行われません。アップストリーム時のデータ名称は <data_name_prefix>/<iscp-v2-compat フォーマットの Data Name フィールド> のようになりますので、iscp-v2-compat フォーマットの Data Name フィールドに、iSCP v1 のデータ ID として使用したい文字列を指定してください。

例えば、data_name_prefix を v1/1 とし、iscp-v2-compat フォーマットの Data Name を abc とすれば、v1/1/abc としてアップストリームが行われ、これを iSCP v1 でダウンストリームする場合は、チャンネル 1、データ ID abc を指定すればよいことになります。

25.3.2 logger-msg (非推奨) 使用時

string/nmea、can_frame、jpeg、h264_annex_b、pcm 型の場合、data_name_prefix が v1/<チャンネル>になっていると、ストリーマーにて以下のように特別な処理が行われます。

string/nmea 型

ペイロード内の NMEA センテンス (例:「\$GNRMC,...」) から 2~3 文字目のトーカ (例:「GN」) と、4~6 文字目のメッセージ (例:「RMC」) が抽出され、データ名称 v1/<チャンネル>/<トーカ><メッセージ> が付与されます。

can_frame 型

ペイロード内から CAN ID が抽出され、データ名称 v1/<チャンネル>/<CAN ID (最上位ビットは IDE)> が付与されます。

jpeg 型

常にデータ名称 v1/<チャンネル>/jpeg が付与されます。

h264_annex_b 型

ペイロード内から H.264 データのタイプが抽出され、データ名称 v1/<チャンネル>/<iSCP v1 の H.264 データのタイプ> が付与されます。

pcm 型

常にデータ名称 v1/<チャンネル>/pcm が付与されます。

string、float64、int64、bytes 型の場合、ストリーマーでの特別な処理は行われません。アップストリーム時のデータ名称は <data_name_prefix>/<logger-msg フォーマットの ID フィールド> のようになります。

例えば、data_name_prefix を v1/1 とし、logger-msg フォーマットの ID フィールドを abc とすれば、v1/1/abc としてアップストリームが行われ、これを iSCP v1 でダウンストリームする場合は、チャンネル 1、データ ID abc を指定すればよいことになります。

26 FIFO 用データフォーマット

intdash Edge Agent 2 とデバイスコネクタの間（デバイコネクタ IPC）で使われる FIFO 用データフォーマットについて説明します。

重要: intdash Edge Agent 2 とデバイスコネクタの間で使われる FIFO 用データフォーマットには 2 種類あります。どちらを使用するかはデバイスコネクタ IPC 設定で指定します。

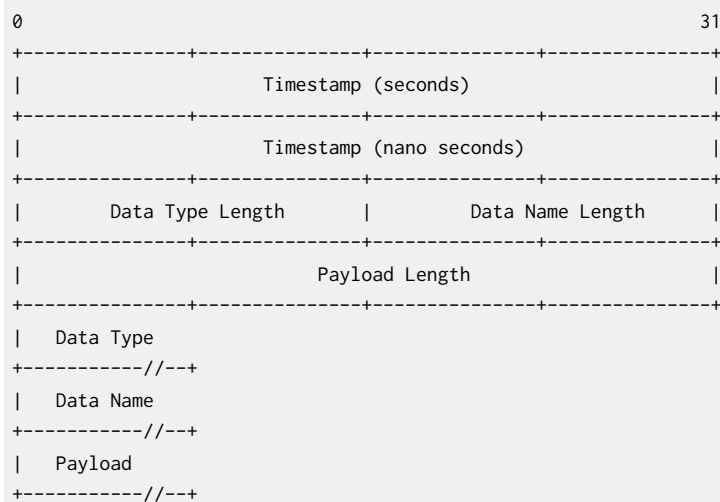
- iscp-v2-compat: iSCP v2 のペイロードフォーマットに沿ったフォーマット
- logger-msg: 旧 intdash Edge Agent で使用されていたフォーマット（非推奨）

本章で説明するのは iscp-v2-compat フォーマットです。

旧フォーマット logger-msg については、[旧 intdash Edge Agent のデベロッパーガイド](#) を参照してください。

26.1 iscp-v2-compat の全体像

iscp-v2-compat フォーマットは以下のとおりです。



フィールド名	バイト長	エンディアン	符号	値範囲	説明
Timestamp (seconds)	4	LE	U	0 ~ 4294967295	このデータのタイムスタンプ (秒)。アップストリームの場合とダウンストリームの場合で意味が異なります。 Timestamp フィールドの意味 (p. 111) を参照してください。
Timestamp (nano seconds)	4	LE	U	0 ~ 999999999	このデータのタイムスタンプ (ナノ秒)。アップストリームの場合とダウンストリームの場合で意味が異なります。 Timestamp フィールドの意味 (p. 111) を参照してください。
Data Type Length	2	LE	U	0 ~ 65535	Data Type フィールドのバイト長
Data Name Length	2	LE	U	0 ~ 65535	Data Name フィールドのバイト長
Payload Length	4	LE	U	0 ~ 4294967295	Payload フィールドのバイト長
Data Type	可変	-	-	-	iSCPV2 のデータ型の文字列
Data Name	可変	-	-	-	iSCPV2 のデータ名称 (もしくはデータ名称の末尾) の文字列。アップストリームの場合とダウンストリームの場合で意味が異なります。 Data Name フィールドの意味 (p. 114) を参照してください。
Payload	可変	-	-	-	iSCPV2 拡張仕様のペイロード

26.2 Timestamp フィールドの意味

Timestamp フィールドの意味は、アップストリームの場合とダウンストリーム場合で異なります。

26.2.1 アップストリームの場合

POSIX の `clock_gettime()` の引数に `CLOCK_MONOTONIC` (または `CLOCK_MONOTONIC_RAW`) を指定して取得した時間を設定します。

注釈: intdash Edge Agent 2 のデフォルト設定では、`CLOCK_MONOTONIC` が使用されます。`CLOCK_MONOTONIC_RAW` を使用する場合は、Agent コマンドでの計測を実行時に以下のように環境変数を設定してください。

```
AGENT_CLOCK_ID=CLOCK_MONOTONIC_RAW intdash-agentctl run
```

`CLOCK_MONOTONIC` の取得方法の例

C

```
#include <time.h>
#include <stdio.h>

int main(int argc, const char* argv[])
{
    struct timespec ts;
    clock_gettime(CLOCK_MONOTONIC, &ts);
    printf("%ld%09ld ns\n", ts.tv_sec, ts.tv_nsec);
    return 0;
}
```

C++11

```
#include <stdio.h>
#include <chrono>

int main(int argc, const char* argv[])
{
    auto steady_ts = std::chrono::steady_clock::now();
    printf("%lld ns\n",
        static_cast<long long>(std::chrono::duration_cast<std::chrono::nanoseconds>(steady_ts.time_since_
epoch()).count()));
    return 0;
}
```

Go

```
package main

import "fmt"

/*
#include <time.h>
static unsigned long long get_nsecs(void)
{
    struct timespec ts;
    clock_gettime(CLOCK_MONOTONIC, &ts);
    return (unsigned long long)ts.tv_sec * 1000000000UL + ts.tv_nsec;
}
*/
```

(次のページに続く)

(前のページからの続き)

```
import "C"

func main() {
    monotonic := uint64(C.get_nsecs())
    fmt.Println(monotonic, "ns")
}
```

Python

```
#!/usr/bin/env python3
import time

if __name__ == '__main__':
    now = int(time.clock_gettime(time.CLOCK_MONOTONIC) * 1_000_000_000)
    print("%d ns" % (now))
```

Rust

```
use libc;

fn main() {
    let mut ts = libc::timespec {
        tv_sec: 0,
        tv_nsec: 0,
    };
    unsafe {
        libc::clock_gettime(libc::CLOCK_MONOTONIC, &mut ts);
    };
    println!(
        "{} ns",
        (ts.tv_sec as u64) * 1000000000 + (ts.tv_nsec as u64)
    );
}
```

26.2.2 ダウンストリームの場合

ダウンストリームの際の FIFO データに含まれる Timestamp フィールドには、計測の基準時刻に経過時間を足すことで得られたタイムスタンプが格納されます。

このとき、基準時刻としては、アップストリームを行うエッジがサーバーに送信した基準時刻（エッジのリアルタイムクロックによるもの、NTP によるものなど複数の基準時刻）のうち、最も高い優先度を持つものが使用されます。計測の途中で優先度が高い基準時刻が得られると、基準時刻はそれに置き換わり、それ以降のタイムスタンプは新しい基準時刻を基に計算されます。

複数のエッジ（ここではエッジ A、エッジ B とします）からのデータを 1 つの intdash Edge Agent 2 でダウンストリームする場合、エッジ A からのデータポイントの Timestamp フィールドはエッジ A が送信した基準時刻を基にしたタイムスタンプ、エッジ B からのデータポイントの Timestamp フィールドはエッジ B が送信した基準時刻を基にしたタイムスタンプです。

26.3 Data Name フィールドの意味

Data Name フィールドの意味は、アップストリームの場合とダウンストリーム場合で異なります。

26.3.1 アップストリームの場合

アップストリームの場合の Data Name フィールドは、iSCP のデータ名称の後半部分に相当します（前半部分は intdash Edge Agent 2 の設定項目 data_name_prefix によって決定されます）。データ型ごとの設定値については [付与されるデータ ID](#) (p. 106) を参照してください。

```
v1/123/00000001
-----
|          |
|          |'- アップストリーム時の Data Name フィールドの内容
|--intdash Edge Agent 2 の data_name_prefix
```

26.3.2 ダウンストリームの場合

ダウンストリームの場合の Data Name フィールドの内容は、iSCP のデータ名称です。アップストリームを行う intdash Edge Agent 2 における「data_name_prefix + Data Name フィールド」に相当します。

```
v1/123/00000001
-----
|
└─ ダウンストリーム時の Data Name フィールドの内容
```

27 環境変数一覧

環境変数	説明
AGENT_LOG	ログレベル (trace debug information warning error)
AGENT_CLOCK_ID	基準時刻取得に使用する時計の種類 (CLOCK_MONOTONIC CLOCK_MONOTONIC_RAW)
AGENT_VERSION	intdash Edge Agent 2 のバージョン情報を記載したファイルのパス
AGENT_DAEMON	intdash-agentd の実行バイナリのパス
AGENT_STREAMER	ストリーマーの実行バイナリのパス
AGENT_INFO_DIR	起動状態を記載した情報ファイルの保存ディレクトリ
AGENT_LIB_DIR	設定ファイル agent.yaml の保存ディレクトリ (agent.yaml の変更を反映するにはデーモンの再起動が必要です。)

28 設定レシピ集

この章では、intdash Edge Agent 2 を使用するための代表的な設定例を紹介します。

28.1 H.264 NAL Unit を使って H.264 データを送信する

iSCP で定義されている H.264 NAL Unit データ型を使って、H.264 の動画をリアルタイム送信するための設定例です。

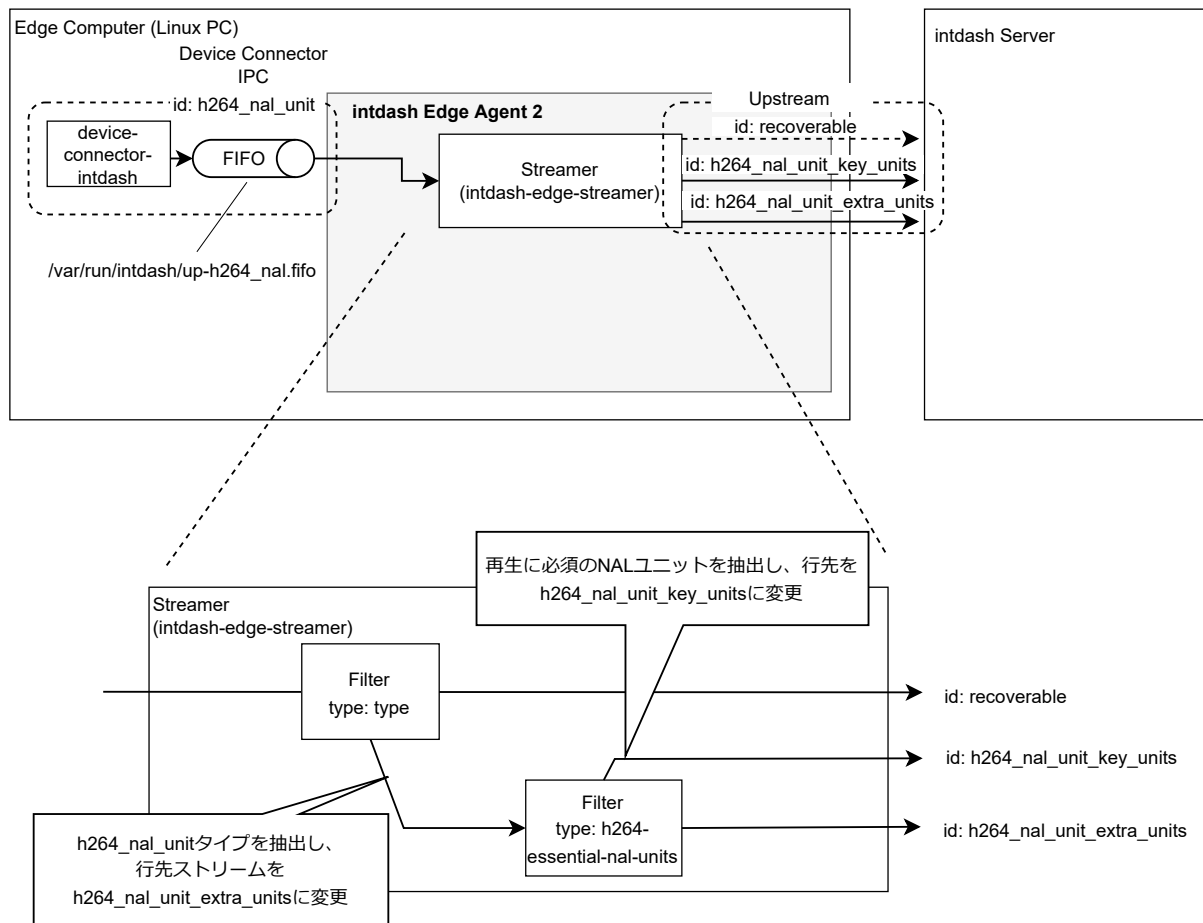


図 23 H.264 NAL Unit データ型を使って、H.264 の動画をリアルタイム送信するための設定例

H.264 NAL Unit データ型を使い適切に設定されたストリームで送信することで、受信側の Visual M2M Data Visualizer では、伝送欠損せず受信できた最低限のデータのみを使った動画の再生が可能になります。データが欠損した個所はブロックノイズになったり、緑色の表示になったりしますが、これにより、送信エッジ側の送信帯域が狭い場合でも、リアルタイム性が高い状態で動画を再生できます。

重要:

- この設定例は、アップストリーム `h264_nal_unit_extra_units` で、信頼性のない接続（`qos: unreliable`）を使用することで効果が発揮されます。そのため、[トランスポート](#) (p. 27) として、QUIC を使用する必要があります。QUIC を使用するためにはサーバー側で設定が必要です。
- H.264 NAL Unit データ型は、H.264 の NALUnit 単位でデータポイントを送信するためのものです。Data Visualizer はこの方法で送信されたデータの再生が可能です。Data Visualizer 以外のアプリケーションを使って再生するには、この送信方法に合わせた再生処理を実装する必要があります。

28.1.1 トランスポートの設定

H.264 NAL Unit に適した送信をするために、トランスポートを設定します。

```
$ intdash-agentctl config transport --modify '  
  protocol: quic  
,
```

28.1.2 アップストリームの設定

H.264 NAL Unit に適した送信をするために、以下の 2 種類のアップストリームを作成します。

アップストリームの ID	概要
<code>h264_nal_unit_key_units</code>	信頼性のあるアップストリームを使用して NAL Unit 単位でデータを送信します。このストリームを使用して、再生に必須となる SPS/PPS/IDR の NAL Unit を送信します。
<code>h264_nal_unit_extra_units</code>	信頼性のないアップストリームを使用して NAL Unit 単位でデータを送信します。このストリームを使用して、nonIDR の NAL Unit を送信します。

これら 2 つのアップストリームを作成するには、以下のコマンドを実行してください。

```
$ intdash-agentctl config up --create '  
  id: h264_nal_unit_key_units  
  enabled: true  
  recover: true  
  persist: true  
  qos: partial  
  flush_policy: immediately  
,  
$ intdash-agentctl config up --create '  
  id: h264_nal_unit_extra_units  
  enabled: true  
  recover: true  
  persist: true  
  qos: unreliable  
  flush_policy: immediately  
,
```

28.1.3 デバイスコネクター IPC の設定

デバイスコネクターからのデータを受け取るためにデバイスコネクター IPC を追加します。以下のコマンドを実行してください。

```
$ intdash-agentctl config device-connector up --create '  
  id: h264_nal_unit  
  data_name_prefix: 101/h264_nal_unit  
  dest_ids:  
    - recoverable  
  format: iscp-v2-compat  
  ipc:  
    type: fifo  
    path: /var/run/intdash/up-h264_nal.fifo  
,
```

/var/run/intdash/up-h264_nal.fifo という FIFO から取得したデータに、101/h264_nal_unit という data_name_prefix を付与し、recoverable というストリームに送る設定です。

注釈: h264_nal_unit は iSCP v1 には存在しないデータ型であるため、Data Visualizer で表示できるようにするために、特別な data_name_prefix を設定しています。

28.1.4 フィルターの設定

h264_nal_unit データ型のデータポイントの行先を、さきほど作成した別のアップストリーム h264_nal_unit_extra_units に変更するフィルターを設定します。

```
$ intdash-agentctl config filter up --create '  
  id: h264_nal_unit_filter  
  enabled: true  
  type: type  
  target:  
    type: h264_nal_unit  
    dest_ids:  
      - recoverable  
  change_to:  
    dest_id: h264_nal_unit_extra_units  
,
```

次に、再生に必須であるデータポイントの行先を、さらに別のアップストリーム h264_nal_unit_key_units に変更するフィルターを設定します。

```
$ intdash-agentctl config filter up --create '  
  id: h264_nal_unit_key_filter  
  enabled: true  
  type: h264-essential-nal-units  
  target:  
    dest_ids:  
      - h264_nal_unit_extra_units  
  change_to:  
    dest_id: h264_nal_unit_key_units  
,
```

注釈: h264-essential-nal-units タイプのフィルターは、再生に必須の NAL ユニット (SPS/PPS/IDR) を抽出するために存在する特別なフィルターです。

28.1.5 ストリーマーの起動

1. 以上の設定ができればストリーマーを起動します。

```
$ intdash-agentctl run
```

2. 別のコンソールでデバイスコネクタを起動して、h264_nal_unit タイプのデータを /var/run/intdash/up-h264_nal.fifo に書き込んでください。

注釈: EDGEPLANT T1 用で device-connector-intdash を使用する場合はパイプライン設定ファイルは以下のとおりです。

/dev/video0 として認識されている UVC カメラから GStreamer を使って画像を取得し、h264_nal_unit として FIFO に書き出します。

```
tasks:
- id: 1
  element: process-src
  conf:
    command: gst-launch-1.0 -q v4l2src device=/dev/video0 ! videorate ! image/jpeg,width=1920,
    ↳height=1080,framerate=15/1 ! jpegdec ! omxh264enc iframeinterval=15 bitrate=3000000 bit-
    ↳packetization=true slice-header-spacing=1200 insert-sps-pps=true insert-vui=true ! video/x-h264,
    ↳stream-format=byte-stream ! queue ! h264parse ! queue ! fdsink fd=1

- id: 2
  element: h264-nalunit-split-filter
  from: [ [1] ]
  conf:
    clock_id: CLOCK_MONOTONIC
    delay_ms: 100

- id: 3
  element: print-log-filter
  from: [ [2] ]
  conf:
    interval_ms: 10000
    tag: video0
    output: stderr

- id: 4
  element: file-sink
  from: [ [3] ]
  conf:
    flush_size: 100
    path: /var/run/intdash/up-h264_nal.fifo
```

VTC 1910-S/VTC 1010 で使用できる omxh264 のパイプラインについては、設定ファイルのサンプル (/etc/dc_conf/gstreamer_h264_nalunit.yml) を参照してください。

3. Data Visualizer でリアルタイムデータを確認します。

送信エッジ側の送信帯域が狭い場合でもリアルタイム性が高い状態で動画を再生できます。

注釈: 送信帯域が狭くなった場合にどうなるかを実際に試す場合には、Linux の tc コマンドを使用するのが便利です。NIC eth0 の送信帯域を 2700kbit/sec に制限する例:

```
$ tc qdisc add dev eth0 handle 10: root tbf rate 2700kbit burst 10kb limit 10kb
```

上のコマンドで設定した帯域制限を解除するには以下を実行します:

```
$ tc qdisc del dev eth0 root
```

28.2 カメラから V4L2 経由で取得した JPEG データを送信する

付属デバイスコネクタ `device-connector-intdash` を使用してカメラから JPEG 画像を取得し、リアルタイム送信するための設定例です。

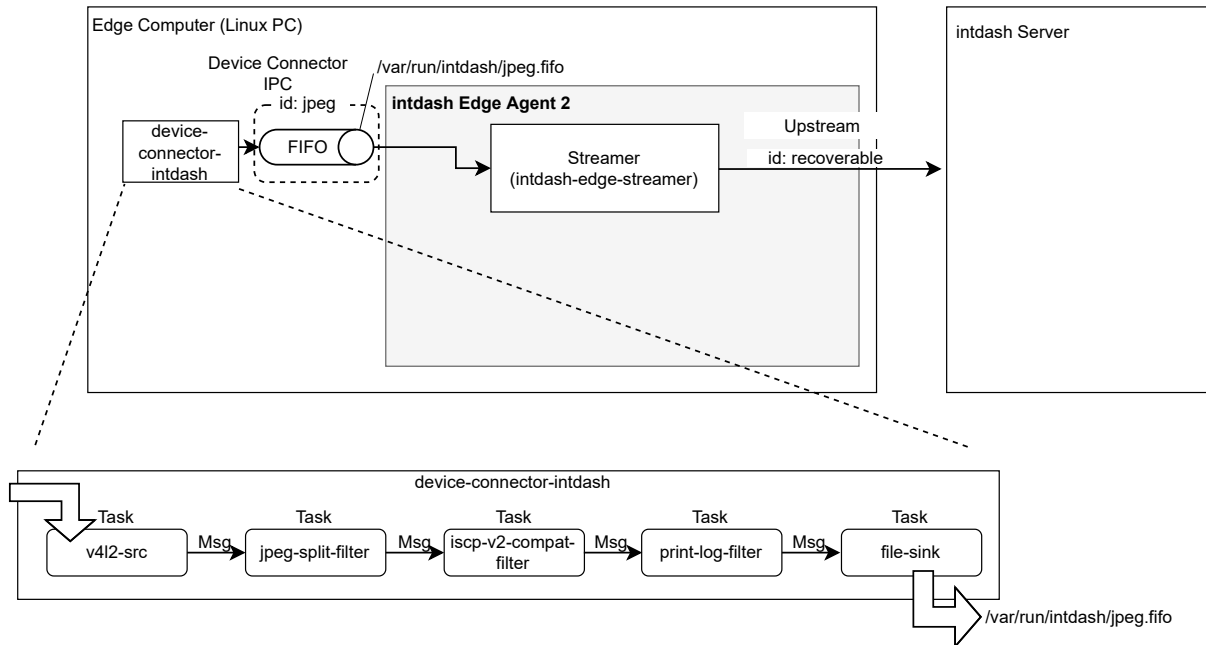


図 24 JPEG 画像を送信するための設定例

28.2.1 アップストリームの設定

以下のコマンドを実行して、`recoverable` という ID を持つアップストリームを作成します。指定しているのは ID のみのため、他の設定値はデフォルトのとおりになります。

```
$ intdash-agentctl config up --create '
  id: recoverable
'
```

28.2.2 デバイスコネクタ IPC の設定

デバイスコネクタからのデータを受け取るためにデバイスコネクタ IPC を追加します。以下のコマンドを実行してください。

```
$ intdash-agentctl config device-connector up --create '
  id: jpeg
  data_name_prefix: v1/101/
  dest_ids:
    - recoverable
  format: iscp-v2-compat
  ipc:
    type: fifo
    path: /var/run/intdash/jpeg.fifo
  launch:
    cmd: device-connector-intdash
  args:
```

(次のページに続く)

(前のページからの続き)

```
- --config
- /etc/dc_conf/jpeg-up.yml
environment:
- DC_V4L2_SRC_CONF_PATH=/dev/video0
- DC_V4L2_SRC_WIDTH=320
- DC_V4L2_SRC_HEIGHT=240
- DC_V4L2_SRC_FPS=5
- DC_PRINT_LOG_FILTER_CONF_TAG=jpeg
- DC_FILE_SINK_CONF_PATH=/var/run/intdash/jpeg.fifo
```

- launch で、device-connector-intdash を起動するように設定しています。
- device-connector-intdash を実行するための設定は、パイプライン設定ファイル /etc/dc_conf/jpeg-up.yml として与えています（このファイルは次の手順で作成します）。また、パイプライン設定ファイルで使用するための環境変数を environment で与えています。
- device-connector-intdash から /var/run/intdash/jpeg.fifo を介して得られたデータは、v1/101/ というデータ名称プリフィックスを与えられて、recoverable という ID を持つアップストリームに送信されます。

28.2.3 device-connector-intdash のパイプライン設定

device-connector-intdash 用のパイプライン設定ファイルを以下の内容で作成し、/etc/dc_conf/jpeg-up.yml として保存します。

```
tasks:
- id: 1
  element: v4l2-src
  conf:
    path: $(DC_V4L2_SRC_CONF_PATH)
    type: jpeg
    width: $(DC_V4L2_SRC_WIDTH)
    height: $(DC_V4L2_SRC_HEIGHT)
    fps: $(DC_V4L2_SRC_FPS)

- id: 2
  element: jpeg-split-filter
  from: [ [1] ]

- id: 3
  element: iscp-v2-compat-filter
  from: [ [2] ]
  conf:
    timestamp:
      stamp:
        clock_id: CLOCK_MONOTONIC
    convert_rule: jpeg

- id: 4
  element: print-log-filter
  from: [ [3] ]
  conf:
    interval_ms: 10000
    tag: $(DC_PRINT_LOG_FILTER_CONF_TAG)
    output: stderr
```

(次のページに続く)

(前のページからの続き)

```
- id: 5
  element: file-sink
  from: [ [4] ]
  conf:
    flush_size: 100
    path: $(DC_FILE_SINK_CONF_PATH)
```

- v4l2-src エLEMENTでは、環境変数として与えられた値（DC_V4L2_SRC_CONF_PATH、DC_V4L2_SRC_WIDTH など）を使って使ってカメラから画像を取得します。（参考: [v4l2-src](#) (p. 54)）
- jpeg-split-filter ではそれを 1 フレームごとの JPEG データに分割します。（参考: [jpeg-split-filter](#) (p. 62)）
- iscp-v2-compat-filter ではそれを FIFO 用データフォーマットに変換するとともに、タイムスタンプを与えます。（参考: [iscp-v2-compat-filter](#) (p. 55)）
- print-log-filter では標準エラー出力にログを出力します。（参考: [print-log-filter](#) (p. 65)）
- file-sink では、\$(DC_FILE_SINK_CONF_PATH) に FIFO 用データフォーマットのデータが書き出されます。これを intdash Edge Agent 2 が読み取ります。（参考: [file-sink](#) (p. 67)）

28.2.4 ストリーマーの起動

以上の設定ができればストリーマーを起動します。

```
$ intdash-agentctl run
```

v1 互換データとして送っているため、Edge Finder で確認することができます。

28.3 EDGEPLANT ANALOG-USB Interface のデータを送信する

付属デバイスコネクタ device-connector-intdash を使用して EDGEPLANT ANALOG-USB Interface からアナログデータを取得し、リアルタイム送信するための設定例です。

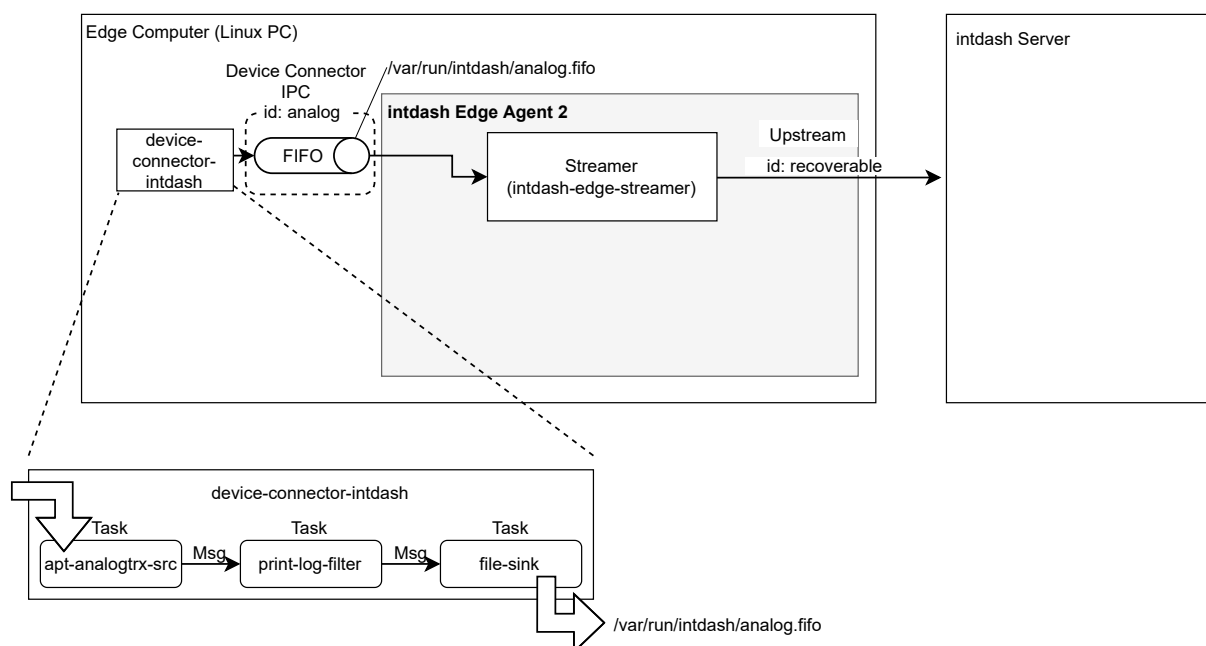


図 25 EDGEPLANT ANALOG-USB Interface からのデータを送信するための設定例

28.3.1 ストリームの設定

以下のコマンドを実行して、recoverable という ID を持つアップストリームを作成します。指定しているのは ID のみのため、他の設定値はデフォルトのとおりになります。

```
$ intdash-agentctl config up --create '  
  id: recoverable  
,
```

28.3.2 デバイスコネクタ IPC の設定

デバイスコネクタからのデータを受け取るためにデバイスコネクタ IPC を追加します。以下のコマンドを実行してください。

```
$ intdash-agentctl config device-connector up --create '  
  id: analog  
  data_name_prefix: v1/101/  
  dest_ids:  
    - recoverable  
  format: iscp-v2-compat  
  ipc:  
    type: fifo  
    path: /var/run/intdash/analog.fifo  
  launch:  
    cmd: device-connector-intdash  
    args:  
      - --config  
      - /etc/dc_conf/analog-up.yml  
  environment:  
    - DC_APT_ANALOGTRX_SRC_CONF_PATH=/dev/apt-usb/by-id/usb-xxx  
    - DC_APT_ANALOGTRX_SRC_CONF_TIMESTAMP_MODE=device  
    - DC_APT_ANALOGTRX_SRC_CONF_INPUT_SEND_RATE=1250000  
    - DC_APT_ANALOGTRX_SRC_CONF_INPUT_ENABLED_0=true  
    - DC_APT_ANALOGTRX_SRC_CONF_INPUT_ENABLED_1=true  
    - DC_APT_ANALOGTRX_SRC_CONF_INPUT_ENABLED_2=true  
    - DC_APT_ANALOGTRX_SRC_CONF_INPUT_ENABLED_3=true  
    - DC_APT_ANALOGTRX_SRC_CONF_INPUT_ENABLED_4=true  
    - DC_APT_ANALOGTRX_SRC_CONF_INPUT_ENABLED_5=true  
    - DC_APT_ANALOGTRX_SRC_CONF_INPUT_ENABLED_6=true  
    - DC_APT_ANALOGTRX_SRC_CONF_INPUT_ENABLED_7=true  
    - DC_APT_ANALOGTRX_SRC_CONF_INPUT_VOLTAGE_MIN_0=-5000  
    - DC_APT_ANALOGTRX_SRC_CONF_INPUT_VOLTAGE_MIN_1=-5000  
    - DC_APT_ANALOGTRX_SRC_CONF_INPUT_VOLTAGE_MIN_2=-5000  
    - DC_APT_ANALOGTRX_SRC_CONF_INPUT_VOLTAGE_MIN_3=-5000  
    - DC_APT_ANALOGTRX_SRC_CONF_INPUT_VOLTAGE_MIN_4=-5000  
    - DC_APT_ANALOGTRX_SRC_CONF_INPUT_VOLTAGE_MIN_5=-5000  
    - DC_APT_ANALOGTRX_SRC_CONF_INPUT_VOLTAGE_MIN_6=-5000  
    - DC_APT_ANALOGTRX_SRC_CONF_INPUT_VOLTAGE_MIN_7=-5000  
    - DC_APT_ANALOGTRX_SRC_CONF_INPUT_VOLTAGE_MAX_0=5000  
    - DC_APT_ANALOGTRX_SRC_CONF_INPUT_VOLTAGE_MAX_1=5000  
    - DC_APT_ANALOGTRX_SRC_CONF_INPUT_VOLTAGE_MAX_2=5000  
    - DC_APT_ANALOGTRX_SRC_CONF_INPUT_VOLTAGE_MAX_3=5000  
    - DC_APT_ANALOGTRX_SRC_CONF_INPUT_VOLTAGE_MAX_4=5000  
    - DC_APT_ANALOGTRX_SRC_CONF_INPUT_VOLTAGE_MAX_5=5000  
    - DC_APT_ANALOGTRX_SRC_CONF_INPUT_VOLTAGE_MAX_6=5000  
    - DC_APT_ANALOGTRX_SRC_CONF_INPUT_VOLTAGE_MAX_7=5000
```

(次のページに続く)

(前のページからの続き)

```

- DC_APT_ANALOGTRX_SRC_CONF_OUTPUT_ENABLED=false
- DC_APT_ANALOGTRX_SRC_CONF_OUTPUT_WAVEFORM_TYPE=0
- DC_APT_ANALOGTRX_SRC_CONF_OUTPUT_VOLTAGE=20
- DC_APT_ANALOGTRX_SRC_CONF_OUTPUT_FREQUENCY=1000
- DC_PRINT_LOG_FILTER_CONF_TAG=analog
- DC_FILE_SINK_CONF_PATH=/var/run/intdash/analog.fifo

```

- launch で、device-connector-intdash を起動するように設定しています。
- device-connector-intdash を実行するための設定は、パイプライン設定ファイル /etc/dc_conf/analog-up.yml として与えています（このファイルは次の手順で作成します）。また、パイプライン設定ファイルで使用するための環境変数を与えています（environment）。
- device-connector-intdash から /var/run/intdash/analog.fifo を介して得られたデータは、v1/101/ というデータ名称プリフィックスを与えられて、recoverable という ID を持つストリームに送信されます。

28.3.3 device-connector-intdash のパイプライン設定

次に device-connector-intdash のパイプライン設定ファイルを以下の内容で作成し、/etc/dc_conf/analog-up.yml として保存します。

```

before_task:
  # sync timestamp
  - mkdir -p /var/lock/intdash
  - |
    BASETIME_CLOCK_ID=$DC_CLOCK_ID
    meas-hook --lockfile /var/lock/intdash/dc_apt_usbtrx.lock --command "
      if command -v apt_usbtrx_timesync.sh > /dev/null 2>&1 ; then apt_usbtrx_timesync.sh; exit 0; fi;
      if command -v apt_usbtrx_timesync_all.sh > /dev/null 2>&1 ; then apt_usbtrx_timesync_all.sh; exit 0; fi;
      echo \"ERROR: timestamp script not found\";
      exit 1;
    "

after_task:
  - rm -f /var/lock/intdash/dc_apt_usbtrx.lock

tasks:
  - id: 1
    element: apt-analogtrx-src
    conf:
      clock_id: CLOCK_MONOTONIC
      path: $(DC_APT_ANALOGTRX_SRC_CONF_PATH)
      timestamp_mode: $(DC_APT_ANALOGTRX_SRC_CONF_TIMESTAMP_MODE)
      input_send_rate: $(DC_APT_ANALOGTRX_SRC_CONF_INPUT_SEND_RATE)
      input_enabled:
        - $(DC_APT_ANALOGTRX_SRC_CONF_INPUT_ENABLED_0)
        - $(DC_APT_ANALOGTRX_SRC_CONF_INPUT_ENABLED_1)
        - $(DC_APT_ANALOGTRX_SRC_CONF_INPUT_ENABLED_2)
        - $(DC_APT_ANALOGTRX_SRC_CONF_INPUT_ENABLED_3)
        - $(DC_APT_ANALOGTRX_SRC_CONF_INPUT_ENABLED_4)
        - $(DC_APT_ANALOGTRX_SRC_CONF_INPUT_ENABLED_5)
        - $(DC_APT_ANALOGTRX_SRC_CONF_INPUT_ENABLED_6)
        - $(DC_APT_ANALOGTRX_SRC_CONF_INPUT_ENABLED_7)
      input_voltage_min:
        - $(DC_APT_ANALOGTRX_SRC_CONF_INPUT_VOLTAGE_MIN_0)

```

(次のページに続く)

(前のページからの続き)

```
- $(DC_APT_ANALOGTRX_SRC_CONF_INPUT_VOLTAGE_MIN_1)
- $(DC_APT_ANALOGTRX_SRC_CONF_INPUT_VOLTAGE_MIN_2)
- $(DC_APT_ANALOGTRX_SRC_CONF_INPUT_VOLTAGE_MIN_3)
- $(DC_APT_ANALOGTRX_SRC_CONF_INPUT_VOLTAGE_MIN_4)
- $(DC_APT_ANALOGTRX_SRC_CONF_INPUT_VOLTAGE_MIN_5)
- $(DC_APT_ANALOGTRX_SRC_CONF_INPUT_VOLTAGE_MIN_6)
- $(DC_APT_ANALOGTRX_SRC_CONF_INPUT_VOLTAGE_MIN_7)
input_voltage_max:
- $(DC_APT_ANALOGTRX_SRC_CONF_INPUT_VOLTAGE_MAX_0)
- $(DC_APT_ANALOGTRX_SRC_CONF_INPUT_VOLTAGE_MAX_1)
- $(DC_APT_ANALOGTRX_SRC_CONF_INPUT_VOLTAGE_MAX_2)
- $(DC_APT_ANALOGTRX_SRC_CONF_INPUT_VOLTAGE_MAX_3)
- $(DC_APT_ANALOGTRX_SRC_CONF_INPUT_VOLTAGE_MAX_4)
- $(DC_APT_ANALOGTRX_SRC_CONF_INPUT_VOLTAGE_MAX_5)
- $(DC_APT_ANALOGTRX_SRC_CONF_INPUT_VOLTAGE_MAX_6)
- $(DC_APT_ANALOGTRX_SRC_CONF_INPUT_VOLTAGE_MAX_7)
output_enabled: $(DC_APT_ANALOGTRX_SRC_CONF_OUTPUT_ENABLED)
output_waveform_type: $(DC_APT_ANALOGTRX_SRC_CONF_OUTPUT_WAVEFORM_TYPE)
output_voltage: $(DC_APT_ANALOGTRX_SRC_CONF_OUTPUT_VOLTAGE)
output_frequency: $(DC_APT_ANALOGTRX_SRC_CONF_OUTPUT_FREQUENCY)

- id: 2
  element: print-log-filter
  from: [[1]]
  conf:
    interval_ms: 10000
    tag: $(DC_PRINT_LOG_FILTER_CONF_TAG)
    output: stderr

- id: 3
  element: file-sink
  from: [[2]]
  conf:
    flush_size: 10
    path: $(DC_FILE_SINK_CONF_PATH)
```

- apt-analogtrx-src エレメントでは、環境変数として与えられた値を使って EDGEPLANT ANALOG-USB Interface からデータを取得します。(参考: [apt-analogtrx-src](#) (p. 47))
- print-log-filter では標準エラー出力にログを出力します。(参考: [print-log-filter](#) (p. 65))
- file-sink では、\$(DC_FILE_SINK_CONF_PATH) に FIFO 用データフォーマットのデータが書き出されます。これを intdash Edge Agent 2 が読み取ります。(参考: [file-sink](#) (p. 67))

28.3.4 ストリーマーの起動

以上の設定ができればストリーマーを起動します。

```
$ intdash-agentctl run
```

28.4 EDGEPLANT CAN-USB Interface のデータを送信する

付属デバイスコネクタ `device-connector-intdash` を使用して EDGEPLANT CAN-USB Interface から CAN データを取得し、リアルタイム送信するための設定例です。

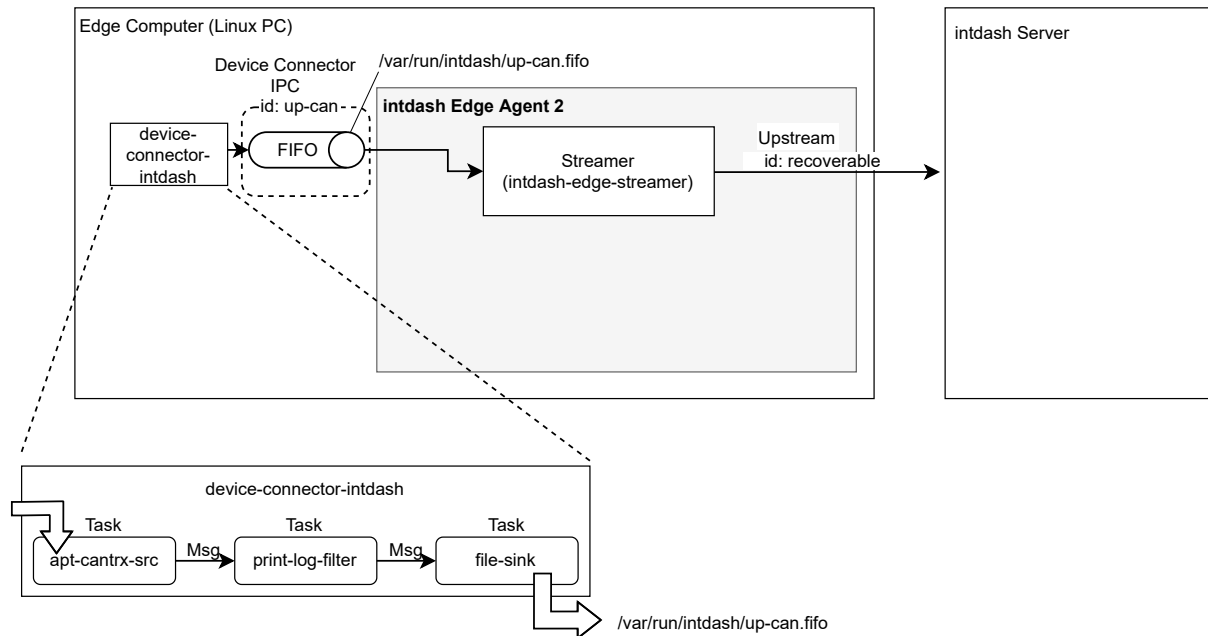


図 26 EDGEPLANT CAN-USB Interface からのデータを送信するための設定例

28.4.1 アップストリームの設定

以下のコマンドを実行して、`recoverable` という ID を持つアップストリームを作成します。指定しているのは ID のみのため、他の設定値はデフォルトのとおりになります。

```
$ intdash-agentctl config up --create '
  id: recoverable
'
```

28.4.2 デバイスコネクタ IPC の設定

デバイスコネクタからのデータを受け取るためにデバイスコネクタ IPC を追加します。以下のコマンドを実行してください。

```
$ intdash-agentctl config device-connector up --create '
  id: up-can
  data_name_prefix: can
  dest_ids:
    - recoverable
  format: iscp-v2-compatible
  ipc:
    type: fifo
    path: /var/run/intdash/up-can.fifo
  launch:
    cmd: device-connector-intdash
'
```

(次のページに続く)

(前のページからの続き)

```
args:
- --config
- /etc/dc_conf/apt_cantrx-up.yml
environment:
- DC_APT_CANTRX_SRC_CONF_PATH=/dev/apt-usb/by-id/usb-xxx
- DC_APT_CANTRX_SRC_CONF_LISTENONLY=1
- DC_APT_CANTRX_SRC_CONF_BAUDRATE=500
- DC_APT_CANTRX_SRC_CONF_TIMESTAMP_MODE=device
- DC_PRINT_LOG_FILTER_CONF_TAG=up-can
- DC_FILE_SINK_CONF_PATH=/var/run/intdash/up-can.fifo
```

- launch で、device-connector-intdash を起動するように設定しています。
- device-connector-intdash を実行するための設定は、パイプライン設定ファイル /etc/dc_conf/apt_cantrx-up.yml として与えています（このファイルは次の手順で作成します）。また、パイプライン設定ファイルで使用するための環境変数を与えています（environment）。
- device-connector-intdash から /var/run/intdash/up-can.fifo を介して得られたデータは、can というデータ名称プリフィックスを与えられて、recoverable という ID を持つアップストリームに送信されます。

28.4.3 device-connector-intdash のパイプライン設定

device-connector-intdash 用のパイプライン設定ファイルを以下の内容で作成し、/etc/dc_conf/apt_cantrx-up.yml として保存します。

```
before_task:
# sync timestamp
- mkdir -p /var/lock/intdash
- |
  BASETIME_CLOCK_ID=$DC_CLOCK_ID
  meas-hook --lockfile /var/lock/intdash/dc_apt_usbtrx.lock --command "
    if command -v apt_usbtrx_timesync.sh > /dev/null 2>&1 ; then apt_usbtrx_timesync.sh; exit 0; fi;
    if command -v apt_usbtrx_timesync_all.sh > /dev/null 2>&1 ; then apt_usbtrx_timesync_all.sh; exit 0; fi;
    echo \"ERROR: timestamp script not found\";
    exit 1;
  "

after_task:
- rm -f /var/lock/intdash/dc_apt_usbtrx.lock

tasks:
- id: 1
  element: apt-cantrx-src
  conf:
    clock_id: CLOCK_MONOTONIC
    path: $(DC_APT_CANTRX_SRC_CONF_PATH)
    listenonly: $(DC_APT_CANTRX_SRC_CONF_LISTENONLY)
    baudrate: $(DC_APT_CANTRX_SRC_CONF_BAUDRATE)
    timestamp_mode: $(DC_APT_CANTRX_SRC_CONF_TIMESTAMP_MODE)

- id: 2
  element: print-log-filter
  from: [[1]]
  conf:
    interval_ms: 10000
```

(次のページに続く)

(前のページからの続き)

```
tag: $(DC_PRINT_LOG_FILTER_CONF_TAG)
output: stderr

- id: 3
  element: file-sink
  from: [[2]]
  conf:
    flush_size: 10
    path: $(DC_FILE_SINK_CONF_PATH)
```

- `before_task` では、EDGEPLANT CAN-USB Interface の同期処理の初期化を行っています。複数の `device-connector-intdash` を使用する場合に対応するためにプロセス間排他を用いています。`after_script` でその後片付けをしています。(`before_task` では、定義されたタスクが開始される前に実行されるコマンドを、`after_task` では、タスク終了後に実行されるコマンドを指定します。)
- `apt-cantrx-src` エLEMENTでは、環境変数として与えられた値を使って EDGEPLANT CAN-USB Interface からデータを取得します。(参考: [apt-analogtrx-src](#) (p. 47))
- `print-log-filter` では標準エラー出力にログを出力します。(参考: [print-log-filter](#) (p. 65))
- `file-sink` では、`$(DC_FILE_SINK_CONF_PATH)` に FIFO 用データフォーマットのデータが書き出されます。これを intdash Edge Agent 2 が読み取ります。(参考: [file-sink](#) (p. 67))

28.4.4 ストリーマーの起動

以上の設定ができればストリーマーを起動します。

```
$ intdash-agentctl run
```

28.5 受信データを EDGEPLANT CAN-USB Interface に送信する

リアルタイムでダウンストリームした CAN データを、`device-connector-intdash` を使用して EDGEPLANT CAN-USB Interface に書き込むための設定例です。

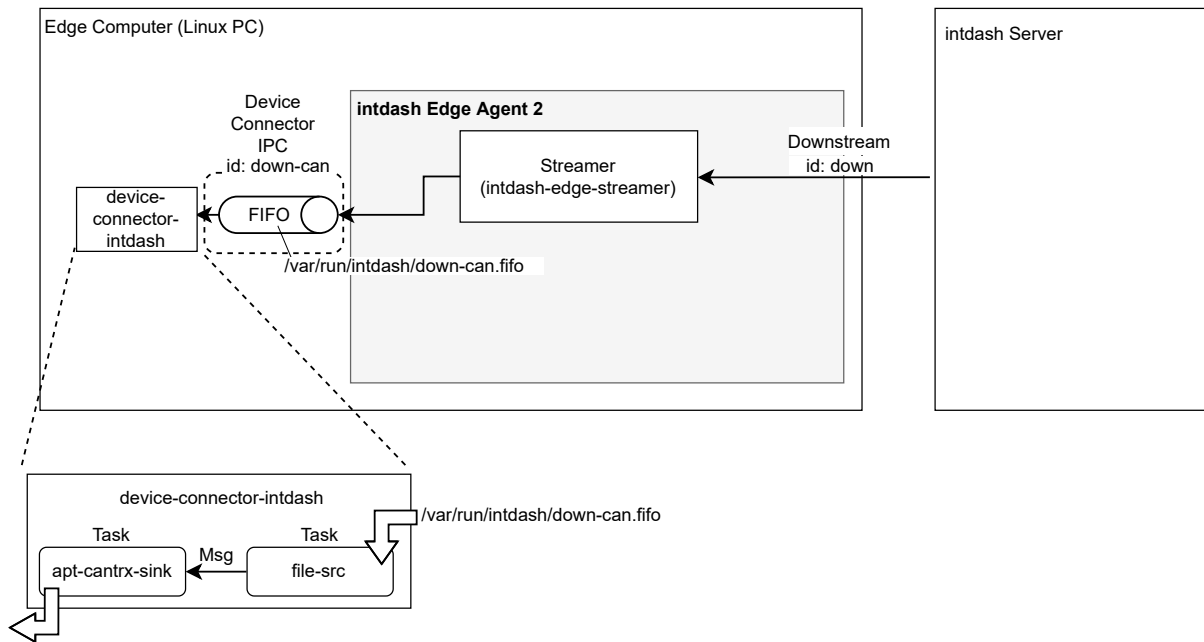


図 27 ダウンストリームした CAN データを EDGEPLANT CAN-USB Interface に書き込むための設定例

28.5.1 ダウンストリームの設定

以下のコマンドを実行して、down という ID を持つダウンストリームを作成します。

送信元が 03ace3b1-d208-4fc3-9763-a502923ca6ab で、データ名称が can のデータのみを受信するようダウンストリームフィルターが設定してあります。

```
$ intdash-agentctl config downstream --create '
  id: down
  enabled: true
  dest_ids:
    - down-can
  filters:
    - src_edge_uuid: 03ace3b1-d208-4fc3-9763-a502923ca6ab
      data_filters:
        - type: can_frame
          name: can
  ,
```

サーバーからダウンストリームされたデータは、デバイスコネクター IPC down-hello に送られます。デバイスコネクター IPC down-hello はこのあとの手順で設定します。

28.5.2 デバイスコネクター IPC の設定

デバイスコネクターにデータを渡すためにデバイスコネクター IPC を追加します。以下のコマンドを実行してください。

```
$ intdash-agentctl config device-connector down --create '
  id: down-can
  format: iscp-v2-compat
  ipc:
    type: fifo
    path: /var/run/intdash/down-can.fifo
  launch:
    cmd: device-connector-intdash
    args:
      - --config
      - /etc/dc_conf/apt_cantrx-down.yml
    environment:
      - DC_APT_CANTRX_SINK_CONF_PATH=/dev/apt-usb/by-id/usb-xxx
      - DC_APT_CANTRX_SINK_CONF_LISTENONLY=1
      - DC_APT_CANTRX_SINK_CONF_BAUDRATE=500
      - DC_APT_CANTRX_SINK_CONF_TIMESTAMP_MODE=device
      - DC_FILE_SRC_CONF_PATH=/var/run/intdash/down-can.fifo
  ,
```

- launch で、device-connector-intdash を起動するように設定しています。
- device-connector-intdash を実行するための設定は、パイプライン設定ファイル /etc/dc_conf/apt_cantrx-down.yml として与えています（このファイルは次の手順で作成します）。また、パイプライン設定ファイルで使用するための環境変数を与えています（environment）。
- ダウンストリームから /var/run/intdash/down-can.fifo を介してデバイスコネクターに送られます。

28.5.3 device-connector-intdash のパイプライン設定

device-connector-intdash 用のパイプライン設定ファイルを以下の内容で作成し、/etc/dc_conf/apt_cantrx-down.yml として保存します。

```
before_task:
  # sync timestamp
  - mkdir -p /var/lock/intdash
  - |
    Basetime_Clock_ID=$DC_Clock_ID
    meas-hook --lockfile /var/lock/intdash/dc_apt_usbtrx.lock --command "
      if command -v apt_usbtrx_timesync.sh > /dev/null 2>&1 ; then apt_usbtrx_timesync.sh; exit 0; fi;
      if command -v apt_usbtrx_timesync_all.sh > /dev/null 2>&1 ; then apt_usbtrx_timesync_all.sh; exit 0; fi;
      echo \"ERROR: timestamp script not found\";
      exit 1;
    "

after_task:
  - rm -f /var/lock/intdash/dc_apt_usbtrx.lock

tasks:
  - id: 10
    element: file-src
    conf:
      path: $(DC_FILE_SRC_CONF_PATH)

  - id: 11
```

(次のページに続く)

(前のページからの続き)

```
element: apt-cantrx-sink
from: [[10]]
conf:
  clock_id: CLOCK_MONOTONIC
  path: $(DC_APT_CANTRX_SINK_CONF_PATH)
  listenonly: $(DC_APT_CANTRX_SINK_CONF_LISTENONLY)
  baudrate: $(DC_APT_CANTRX_SINK_CONF_BAUDRATE)
  timestamp_mode: $(DC_APT_CANTRX_SINK_CONF_TIMESTAMP_MODE)
```

- file-src では、intdash Edge Agent 2 によって書き出されたデータを\$(DC_FILE_SINK_CONF_PATH)から読みます。(参考: [file-src](#) (p. 50))
- apt-cantrx-sink エlementでは、環境変数として与えられた値を使って EDGEPLANT CAN-USB Interface にデータを渡します。(参考: [apt-cantrx-src](#) (p. 49))

28.5.4 ストリーマーの起動

以上の設定ができたらストリーマーを起動します。

```
$ intdash-agentctl run
```

28.6 u-blox GNSS モジュールから取得した UBX プロトコルのメッセージを送信する

付属デバイスコネクタ device-connector-intdash を使用して UBX プロトコルのメッセージを取得し、リアルタイム送信するための設定例です。

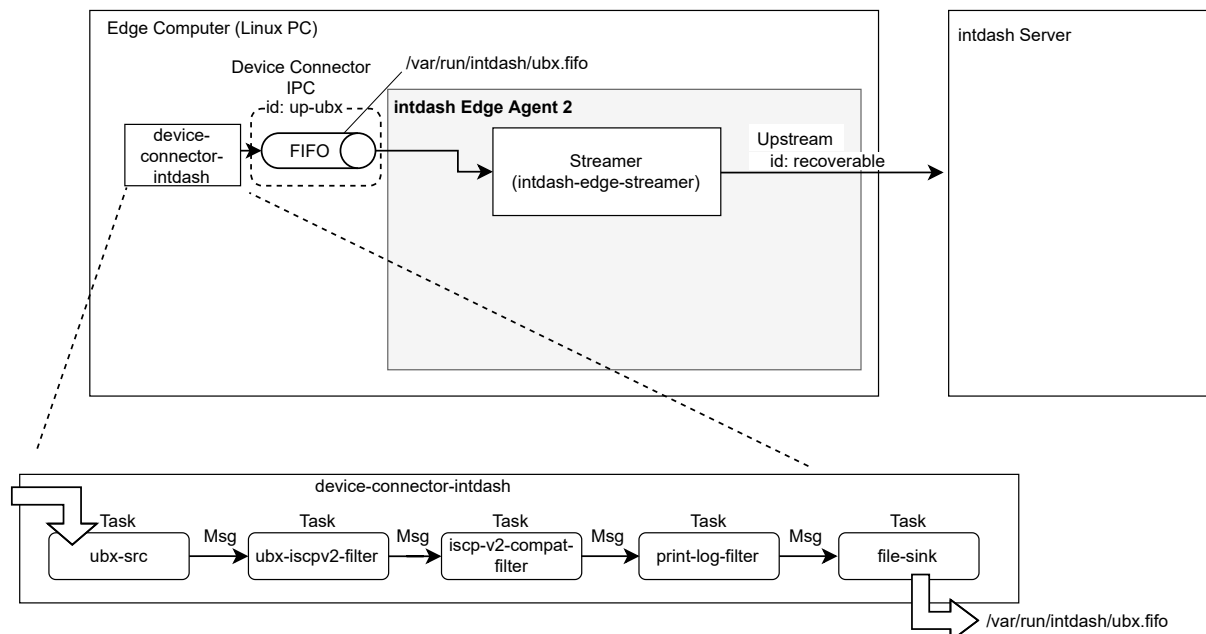


図 28 UBX プロトコルのメッセージを送信するための設定例

28.6.1 アップストリームの設定

以下のコマンドを実行して、recoverable という ID を持つアップストリームを作成します。指定しているのは ID のみのため、他の設定値はデフォルトのとおりになります。

```
$ intdash-agentctl config up --create '
  id: recoverable
'
```

28.6.2 デバイスコネクタ IPC の設定

デバイスコネクタからのデータを受け取るためにデバイスコネクタ IPC を追加します。以下のコマンドを実行してください。

```
$ intdash-agentctl config device-connector up --create '
  id: up-ubx
  data_name_prefix: ubx
  dest_ids:
    - recoverable
  format: iscp-v2-compat
  ipc:
    type: fifo
    path: /var/run/intdash/up-ubx.fifo
  launch:
    cmd: device-connector-intdash
    args:
      - --config
      - /etc/dc_conf/ubx.yml
    environment:
      - DC_UBX_SRC_CONF_PATH=/dev/ttyTHS1
      - DC_UBX_SRC_CONF_BAUD_RATE=57600
      - DC_UBX_SRC_CONF_MEAS_RATE_MS=200
      - DC_UBX_SRC_CONF_NAV_RATE=1
      - DC_UBX_SRC_CONF_HIGH_NAV_RATE_HZ=5
      - DC_UBX_SRC_CONF_ESF_STATUS_RATE=1
      - DC_UBX_SRC_CONF_HNR_ATT_RATE=1
      - DC_UBX_SRC_CONF_HNR_INS_RATE=1
      - DC_UBX_SRC_CONF_HNR_PVT_RATE=1
      - DC_UBX_SRC_CONF_NAV_STATUS_RATE=1
      - DC_PRINT_LOG_FILTER_CONF_TAG=ubx
      - DC_FILE_SINK_CONF_PATH=/var/run/intdash/up-ubx.fifo
'
```

- launch で、device-connector-intdash を起動するように設定しています。
- device-connector-intdash を実行するための設定は、パイプライン設定ファイル /etc/dc_conf/ubx.yml として与えています。また、パイプライン設定ファイルで使用するための環境変数を与えています (environment)。
- 特に、DC_UBX_SRC_CONF_PATH は u-blox GNSS モジュールのデバイスパス、DC_UBX_SRC_CONF_BAUD_RATE は使用されるボーレートです。使用するデバイスに合った値を設定してください。
- device-connector-intdash から /var/run/intdash/up-ubx.fifo を介して得られたデータは、ubx というデータ名称プリフィックスを与えられて、recoverable という ID を持つアップストリームに送信されます。

28.6.3 ストリーマーの起動

以上の設定ができればストリーマーを起動します。

```
$ intdash-agentctl run
```

28.7 NMEA データを送信する

付属デバイスコネクタ `device-connector-intdash` を使用して NMEA データを取得し、リアルタイム送信するための設定例です。

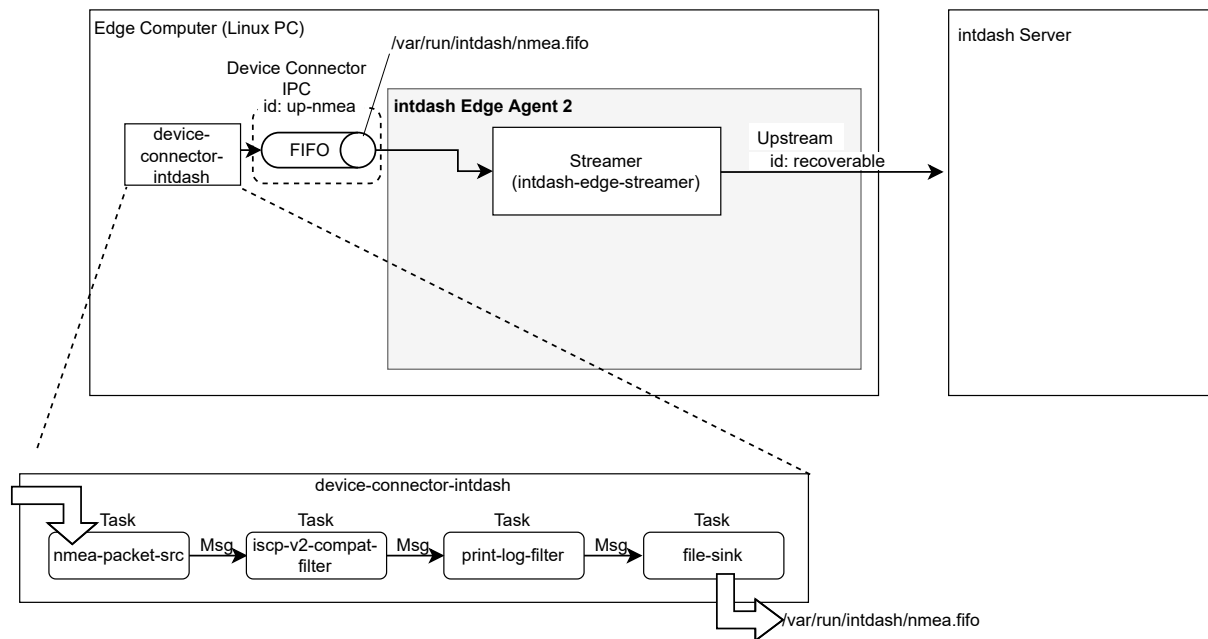


図 29 NMEA データを送信するための設定例

28.7.1 アップストリームの設定

以下のコマンドを実行して、`recoverable` という ID を持つアップストリームを作成します。指定しているのは ID のみのため、他の設定値はデフォルトのとおりになります。

```
$ intdash-agentctl config up --create '
  id: recoverable
'
```

28.7.2 デバイスコネクタ IPC の設定

デバイスコネクタからのデータを受け取るためにデバイスコネクタ IPC を追加します。以下のコマンドを実行してください。

```
$ intdash-agentctl config device-connector up --create '
  id: up-nmea
  data_name_prefix: nmea
  dest_ids:
```

(次のページに続く)

(前のページからの続き)

```

- recoverable
format: iscp-v2-compat
ipc:
  type: fifo
  path: /var/run/intdash/nmea.fifo
launch:
  cmd: device-connector-intdash
  args:
    - --config
    - /etc/dc_conf/nmea-up.yml
  environment:
    - DC_NMEA_PACKET_SRC_CONF_PATH=/dev/ttyTHS1
    - DC_NMEA_PACKET_SRC_CONF_BAUDRATE=57600
    - DC_PRINT_LOG_FILTER_CONF_TAG=nmea
    - DC_FILE_SINK_CONF_PATH=/var/run/intdash/nmea.fifo
,
```

- launch で、device-connector-intdash を起動するように設定しています。
- device-connector-intdash を実行するための設定は、パイプライン設定ファイル /etc/dc_conf/nmea-up.yml として与えています（このファイルは次の手順で作成します）。また、パイプライン設定ファイルで使用するための環境変数を与えています（environment）。
- 特に、DC_NMEA_PACKET_SRC_CONF_PATH は GPS デバイスのデバイスパス、DC_NMEA_PACKET_SRC_CONF_BAUDRATE は使用されるボーレートです。使用するデバイスに合った値を設定してください。
- device-connector-intdash から /var/run/intdash/nmea.fifo を介して得られたデータは、nmea というデータ名称プリフィックスを与えられて、recoverable という ID を持つアップストリームに送信されます。

28.7.3 device-connector-intdash のパイプライン設定

次に device-connector-intdash のパイプライン設定ファイルを /etc/dc_conf/nmea-up.yml として作成します。

```

tasks:
- id: 1
  element: nmea-packet-src
  conf:
    path: $(DC_NMEA_PACKET_SRC_CONF_PATH)
    baudrate: $(DC_NMEA_PACKET_SRC_CONF_BAUDRATE)

- id: 2
  element: iscp-v2-compat-filter
  from: [ [1] ]
  conf:
    timestamp:
      stamp:
        clock_id: CLOCK_MONOTONIC
    convert_rule: nmea

- id: 3
  element: print-log-filter
  from: [ [2] ]
  conf:
    interval_ms: 10000
    tag: $(DC_PRINT_LOG_FILTER_CONF_TAG)
    output: stderr
```

(次のページに続く)

(前のページからの続き)

```
- id: 4
  element: file-sink
  from: [ [3] ]
  conf:
    path: $(DC_FILE_SINK_CONF_PATH)
```

- nmea-packet-src エレメントでは、環境変数として与えられた値を使って NMEA メッセージを取得します。(参考: [nmea-packet-src](#) (p. 50))
- iscp-v2-compatible-filter ではそれを FIFO 用データフォーマットに変換するとともに、タイムスタンプを与えます。(参考: [iscp-v2-compatible-filter](#) (p. 55))
- print-log-filter では標準エラー出力にログを出力します。(参考: [print-log-filter](#) (p. 65))
- file-sink では、\$(DC_FILE_SINK_CONF_PATH) に FIFO 用データフォーマットのデータが書き出されます。これを intdash Edge Agent 2 が読み取ります。(参考: [file-sink](#) (p. 67))

注釈: nmea-packet-src は GPS デバイスの初期化は行いません。詳細については、[nmea-packet-src](#) (p. 50) の注釈を参照してください。

28.7.4 ストリーマーの起動

以上の設定ができたらストリーマーを起動します。

```
$ intdash-agentctl run
```

28.8 音声を送信する

EDGEPLANT T1 のオンボードの音声ジャックに接続されたマイクから、付属デバイスコネクタ device-connector-intdash を使用して音声データを取得し、リアルタイム送信するための設定例です。

重要: 本設定を使用する際は、オンボードの音声ジャック以外にはマイクを接続しないでください。USB 接続のマイクなどが接続されていると、オンボードの音声ジャック以外から音声を取得する可能性があります。

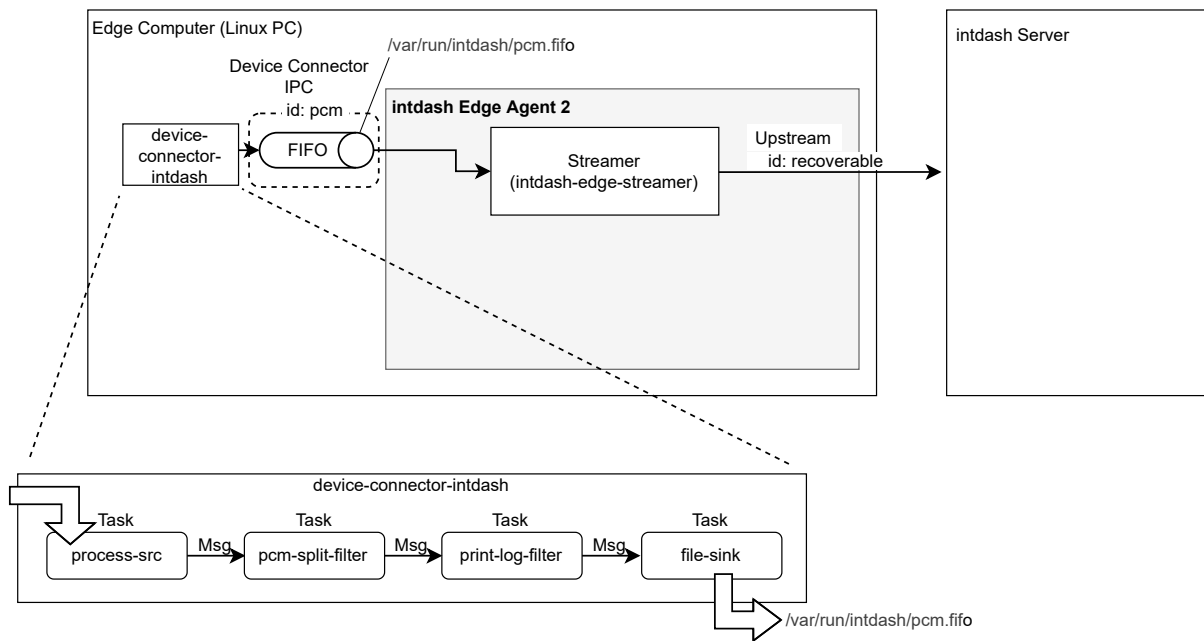


図 30 音声データを送信するための設定例

28.8.1 アップストリームの設定

以下のコマンドを実行して、recoverable という ID を持つアップストリームを作成します。指定しているのは ID のみのため、他の設定値はデフォルトのとおりになります。

```
$ intdash-agentctl config up --create '
  id: recoverable
'
```

28.8.2 デバイスコネクター IPC の設定

デバイスコネクターからのデータを受け取るためにデバイスコネクター IPC を追加します。以下のコマンドを実行してください。

```
$ intdash-agentctl config device-connector up --create '
  id: pcm
  data_name_prefix: v1/150
  format: iscp-v2-compat
  dest_ids:
    - recoverable
  ipc:
    type: fifo
    path: /var/run/intdash/pcm.fifo
  launch:
    cmd: device-connector-intdash
    args:
      - --config
      - /etc/dc_conf/pcm.yml
    environment:
      - DC_PROCESS_SRC_CONF_COMMAND="gst-launch-1.0 -q alsasrc device=hw:1 ! audioconvert ! audio/x-raw,
format=S32LE,rate=48000,channels=1 ! fdsink fd=1"
      - DC_PCM_SPLIT_FILTER_CONF_DELAY_MS=0
'
```

(次のページに続く)

(前のページからの続き)

```
- DC_PCM_SPLIT_FILTER_CONF_AUDIO_ELEMENT="y Jack-state"
- DC_PCM_SPLIT_FILTER_CONF_AUDIO_IFACE="mixer"
- DC_PCM_SPLIT_FILTER_CONF_AUDIO_FORMAT="S32LE"
- DC_PCM_SPLIT_FILTER_CONF_AUDIO_RATE=48000
- DC_PCM_SPLIT_FILTER_CONF_AUDIO_CHANNELS=1
- DC_PCM_SPLIT_FILTER_CONF_AUDIO_VOLUME_IFACE="mixer"
- DC_PCM_SPLIT_FILTER_CONF_AUDIO_VOLUME_ELEMENT="y Mic Capture Volume"
- DC_PCM_SPLIT_FILTER_CONF_AUDIO_VOLUME_VALUE=20
- DC_PCM_SPLIT_FILTER_CONF_AUDIO_BOOST_ELEMENT="y Mic Boost Capture Volume"
- DC_PCM_SPLIT_FILTER_CONF_AUDIO_BOOST_VALUE=1
- DC_PCM_SPLIT_FILTER_CONF_PATH=/dev/snd/by-path/platform-sound
- DC_PRINT_LOG_FILTER_CONF_TAG=pcm
- DC_FILE_SINK_CONF_PATH=/var/run/intdash/pcm.fifo
```

- launch で、device-connector-intdash を起動するように設定しています。
- device-connector-intdash を実行するための設定は、パイプライン設定ファイル /etc/dc_conf/pcm.yml として与えています（このファイルは次の手順で作成します）。また、パイプライン設定ファイルで使用するための環境変数を environment で与えています。
- device-connector-intdash から /var/run/intdash/pcm.fifo を介して得られたデータは、v1/150 というデータ名称プリフィックスを与えられて、recoverable という ID を持つアップストリームに送信されます。

28.8.3 device-connector-intdash のパイプライン設定

device-connector-intdash 用のパイプライン設定ファイルを以下の内容で作成し、/etc/dc_conf/pcm.yml として保存します。

```
tasks:
- id: 1
  element: process-src
  conf:
    command: $(DC_PROCESS_SRC_CONF_COMMAND)

- id: 2
  element: pcm-split-filter
  from: [ [1] ]
  conf:
    clock_id: CLOCK_MONOTONIC
    delay_ms: $(DC_PCM_SPLIT_FILTER_CONF_DELAY_MS)
    audio_element: $(DC_PCM_SPLIT_FILTER_CONF_AUDIO_ELEMENT)
    audio_iface: $(DC_PCM_SPLIT_FILTER_CONF_AUDIO_IFACE)
    audio_format: $(DC_PCM_SPLIT_FILTER_CONF_AUDIO_FORMAT)
    audio_rate: $(DC_PCM_SPLIT_FILTER_CONF_AUDIO_RATE)
    audio_channels: $(DC_PCM_SPLIT_FILTER_CONF_AUDIO_CHANNELS)
    audio_volume_iface: $(DC_PCM_SPLIT_FILTER_CONF_AUDIO_VOLUME_IFACE)
    audio_volume_element: $(DC_PCM_SPLIT_FILTER_CONF_AUDIO_VOLUME_ELEMENT)
    audio_volume_value: $(DC_PCM_SPLIT_FILTER_CONF_AUDIO_VOLUME_VALUE)
    audio_boost_element: $(DC_PCM_SPLIT_FILTER_CONF_AUDIO_BOOST_ELEMENT)
    audio_boost_value: $(DC_PCM_SPLIT_FILTER_CONF_AUDIO_BOOST_VALUE)
    path: $(DC_PCM_SPLIT_FILTER_CONF_PATH)

- id: 3
  element: print-log-filter
  from: [ [2] ]
```

(次のページに続く)

(前のページからの続き)

```
conf:
  interval_ms: 10000
  tag: $(DC_PRINT_LOG_FILTER_CONF_TAG)
  output: stderr

- id: 4
  element: file-sink
  from: [ [3] ]
  conf:
    create: true
    flush_size: 100
    path: $(DC_FILE_SINK_CONF_PATH)
```

- process-src エlementでは、環境変数として与えられた値（DC_PROCESS_SRC_CONF_COMMAND）を使ってマイクから音声データを取得します。（参考: [process-src](#) (p. 51)）
- pcm-split-filter ではそれを FIFO 用データフォーマットに分割・変換するとともに、タイムスタンプを与えます。（参考: [pcm-split-filter](#) (p. 62)）
- print-log-filter では標準エラー出力にログを出力します。（参考: [print-log-filter](#) (p. 65)）
- file-sink では、\$(DC_FILE_SINK_CONF_PATH) に FIFO 用データフォーマットのデータが書き出されます。これを intdash Edge Agent 2 が読み取ります。（参考: [file-sink](#) (p. 67)）

28.8.4 ストリーマーの起動

以上の設定ができればストリーマーを起動します。

```
$ intdash-agentctl run
```