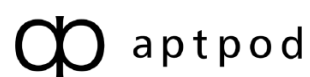


intdash Bridge

デベロッパーガイド

intdash Bridge Version 2.2.0

第 4 版 (2024 年 1 月)



目次

01 はじめに	3
1.1 intdash Bridge とは.....	3
1.2 intdash Bridge の主要機能	4
1.3 動作要件	4
02 使用方法	5
2.1 intdash Edge Agent をインストールする	5
2.2 intdash Bridge をインストールする	6
2.3 intdash Edge Agent の設定を行う	6
2.4 intdash Bridge の設定を行う	8
2.5 intdash Edge Agent と intdash Bridge を起動する	13
03 チュートリアル A: intdash Bridge を使って 2 つの ROS 空間を接続する	14
3.1 送信側エッジを準備する	14
3.2 受信側エッジを準備する	18
3.3 メッセージの送信を開始する	22
04 チュートリアル B: Data Visualizer で ROS メッセージを表示する	23
4.1 送信側エッジを準備する	23
4.2 Data Visualizer で JSON 用のデータ設定を準備する	27
4.3 Data Visualizer でビジュアルパーツを配置する.....	29
4.4 メッセージの送信を開始する	30
05 チュートリアル C: 画像の ROS メッセージを JPEG に変換して Data Visualizer で表示する	31
5.1 送信側エッジを準備する	31
5.2 Data Visualizer で JPEG 用のデータ設定を準備する	36
5.3 Data Visualizer でビジュアルパーツを配置する.....	38
5.4 メッセージの送信を開始する	39
06 チュートリアル D: 画像の ROS メッセージを H.264 に変換して Data Visualizer で表示する	40
6.1 送信側エッジを準備する	40
6.2 Data Visualizer で H.264 用のデータ設定を準備する	45
6.3 Data Visualizer でビジュアルパーツを配置する.....	47
6.4 メッセージの送信を開始する	48
07 制限事項	49
7.1 JSON の出力形式について	49
7.2 JSON での uint8 型の出力について	49

01 はじめに

重要:

- このドキュメントに記載されている仕様は予告なく変更されることがあります。このドキュメントは情報提供を目的としたものであり、仕様を保証するものではありません。
- 説明で使用している画面は一例です。ご使用の環境やアプリケーションのバージョンによって、表示や手順が一部異なる場合があります。

注釈: このドキュメントに記載されている会社名、サービス名、製品名等は、一般に、各社の登録商標または商標です。本文および図表中には、「™」、「®」は明記していません。

1.1 intdash Bridge とは

intdash Bridge は、エッジデバイスにおいて、ROS 空間と intdash Edge Agent の間を仲介するソフトウェアです。intdash Bridge を使用することにより、ROS 空間と intdash サーバーとの間で、ROS メッセージのやり取りが可能になります。

intdash Bridge は、intdash Edge Agent に対しては 1 つのデバイスコネクターとして振る舞い、ROS 空間では 1 つの ROS ノードとして振る舞います。これにより、ROS 空間で受信したメッセージを intdash サーバーへ送信することができ（アップストリーム）、また、intdash サーバーから受信したメッセージを ROS 空間に送信することができます（ダウンストリーム）。

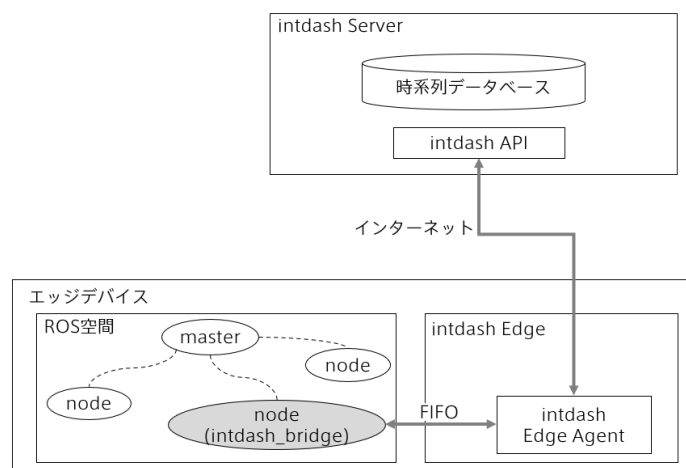


図 1 intdash Bridge の機能

1.2 intdash Bridge の主要機能

- ROS メッセージのサブスクライブと、サブスクライブした ROS メッセージの intdash Edge Agent への送信
- intdash Edge Agent からのデータの受信と、受信したメッセージの ROS 空間内へのパブリッシュ

注釈: intdash Bridge は、トピックによる ROS メッセージのサブスクライブとパブリッシュのみに対応しています。

1.3 動作要件

intdash Bridge の動作には、intdash Edge Agent が必要です。

intdash Bridge が動作する ROS のディストリビューションは以下のとおりです。

- ROS Melodic
- ROS Noetic

02 使用方法

intdash Bridge を使用するには、intdash Edge Agent と intdash Bridge をインストールする必要があります。

2.1 intdash Edge Agent をインストールする

intdash Edge Agent は、アプトポッドのリポジトリからパッケージを取得することによりインストールします。

1. 以下のようにコマンドを実行し、アプトポッドのリポジトリを取得元として追加します。

DISTRIBUTION、ARCHITECTURE には、以下の表から値を設定してください。

DISTRIBUTION	ARCHITECTURE
ubuntu	amd64, armhf, arm64

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

なお、上記の手順は最新ではない可能性があります。intdash Edge Agent のインストール方法の詳細については、[intdash Edge Agent デベロッパーガイド](#) を参照してください。

2. intdash Edge Agent をインストールします。

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

2.2 intdash Bridge をインストールする

ヒント: 以下の説明ではご使用の環境に直接 intdash Bridge をインストールしますが、Docker を使って intdash Bridge を起動することも可能です。[Amazon ECR Public Gallery](#) にて配布されている [Docker イメージ](#) の説明を参考にしてください。

intdash Bridge をインストールするには、アプトポッドのリポジトリからパッケージを取得します。

1. 以下のようにコマンドを実行し、アプトポッドのリポジトリを取得元として追加します。

ARCHITECTURE は [intdash Edge Agent をインストールする](#) (p. 5) で設定したものを使用します。

```
$ curl -s --compressed \  
"https://repository.aptpod.jp/intdash-robotics/linux/ubuntu/gpg" | sudo apt-key add -  
$ echo "deb [arch=${ARCHITECTURE}] \  
https://repository.aptpod.jp/intdash-robotics/linux/ubuntu \  
$(lsb_release -cs) \  
stable" \  
| sudo tee /etc/apt/sources.list.d/intdash-robotics.list  
$ sudo apt-get update
```

重要: intdash Bridge v2.0.0 から、インストール時に指定するアプトポッドのリポジトリが変更されました。

2. 使用する ROS のディストリビューションに合わせて intdash Bridge をインストールします。

```
$ sudo apt-get install ros-[melodic,noetic]-intdash-bridge
```

3. JPEG または H.264 で送信する機能を利用する場合のみ: GStreamer を利用するためのパッケージをインストールします。

```
$ sudo apt-get install gstreamerproxy
```

2.3 intdash Edge Agent の設定を行う

intdash Edge Agent に、intdash Bridge と接続するための設定を追加します。

intdash Edge Agent の設定の詳細については、[intdash Edge Agent デベロッパーガイド](#) を参照してください。

設定ファイル manager.conf 内で、loggers (デバイスコネクター) に、intdash_bridge を追加します。

```
"loggers": [  
  {  
    "devicetype": "intdash_bridge",  
    "path": "",  
    "connections": [  
      {
```

(次のページに続く)

(前のページからの続き)

```
    "fifo_tx": "/opt/vm2m/var/run/intdash/logger_001.tx", # (1)
    "fifo_rx": "/opt/vm2m/var/run/intdash/logger_001.rx", # (2)
    "channel": 1 # (3)
  }, # (X)
  {
    "fifo_tx": "/opt/vm2m/var/run/intdash/logger_002.tx", # (1)
    "channel": 2 # (3)
  } # (Y)
],
"details": {
  "plugin": "fifo"
}
},
...
]
```

上記サンプルでは、intdash Bridge との間で双方向の送受信を行うチャンネル 1 (X) と、intdash Bridge からの受信のみを行うチャンネル 2 (Y) を設定しています。このあとの設定により、チャンネル 1 では intdash Bridge 専用の Raw バイナリ形式で送受信を行い、チャンネル 2 では JSON 形式で送信のみを行います。

番号	フィールド	説明
(1)	loggers[.connections[.fifo_tx	intdash Edge Agent が intdash Bridge からデータを受け取るために使用する FIFO のパスです (アップストリーム用)。任意のパスを指定してください。このパスは intdash Bridge の設定ファイル (p. 8) にも設定します。
(2)	loggers[.connections[.fifo_rx	intdash Edge Agent が intdash Bridge にデータを渡すために使用する FIFO のパスです (ダウンストリーム用)。任意のパスを指定してください。このパスは intdash Bridge の設定ファイル (p. 8) にも設定します。
(3)	loggers[.connections[.channel	データに付与する intdash のチャンネル番号を設定します。

注釈: 「Raw バイナリ形式」 (raw モード) について

ここで使用する「Raw バイナリ形式」は、intdash Bridge 専用のバイナリ形式で、intdash サーバーを介して 2 つの ROS 空間を接続する際に適した形式です (2 つの ROS 空間を接続する方法については、チュートリアル [チュートリアル A: intdash Bridge を使って 2 つの ROS 空間を接続する](#) (p. 14) も参照してください)。Raw バイナリ形式は、データを収集して後から利用するのには向きません。

[intdash Bridge のパラメーター設定](#) (p. 9) で、send_mode: "raw" を設定することにより、Raw バイナリ形式で送信することができます。

なお、Raw バイナリ形式には、topic_tools::ShapeShifter と呼ばれる、ROS の内部実装で使用されているデータ型が使用されています。

2.4 intdash Bridge の設定を行う

intdash Bridge に関する設定は、launch ファイルと、パラメーター設定ファイル params.yaml で行います。

2.4.1 launch ファイルでの設定

intdash Bridge を ROS ノードとして起動する launch ファイルを以下のように作成します。(mylaunch.launch)

```
<launch>
  <arg name="paramsfile" default="/opt/vm2m/etc/params.yaml" />
  <group ns="intdash">
    <node pkg="intdash_bridge" name="intdash_bridge"
      type="intdash_bridge_node" output="screen" clear_params="true">
      <param name="fifo_tx_raw" value="/opt/vm2m/var/run/intdash/logger_001.tx" /> <!-- (1) -->
      <param name="fifo_rx_raw" value="/opt/vm2m/var/run/intdash/logger_001.rx" /> <!-- (2) -->
      <param name="fifo_tx_json" value="/opt/vm2m/var/run/intdash/logger_002.tx" /> <!-- (3) -->
      <param name="fifo_tx_jpeg_1" value="/opt/vm2m/var/run/intdash/logger_003.tx" /> <!-- (4) -->
      <param name="fifo_tx_j264_1" value="/opt/vm2m/var/run/intdash/logger_004.tx" /> <!-- (5) -->
      <rosparam command="load" file="$(arg paramsfile)" />
    </node>
  </group>
</launch>
```


番号	パラメーター名	説明
(1)	fifo_tx_raw	intdash Bridge が intdash Edge Agent に Raw バイナリ形式の ROS メッセージを渡すために使用する FIFO のパスです (アップストリーム用)。intdash Edge Agent の設定を行う (p. 6) で Raw バイナリ形式用の fifo_tx として設定した値を指定します。
(2)	fifo_rx_raw	intdash Bridge が intdash Edge Agent から Raw バイナリ形式の ROS メッセージを受け取るために使用する FIFO のパスです (ダウンストリーム用)。intdash Edge Agent の設定を行う (p. 6) で Raw バイナリ形式用の fifo_rx として設定した値を指定します。
(3)	fifo_tx_json	intdash Bridge が intdash Edge Agent に JSON 文字列形式の ROS メッセージを渡すために使用する FIFO のパスです (アップストリーム用)。intdash Edge Agent の設定を行う (p. 6) で JSON 用の fifo_tx として設定した値を指定します。
(4)	fifo_tx_jpeg_1	intdash Bridge が intdash Edge Agent に JPEG 形式の ROS メッセージを渡すために使用する FIFO のパスです (アップストリーム用)。intdash Edge Agent の設定を行う (p. 6) で JPEG 用の fifo_tx として設定した値を指定します。jpeg_1 から jpeg_4 の値を指定可能です。
(5)	fifo_tx_h264_1	intdash Bridge が intdash Edge Agent に H.264 形式の ROS メッセージを渡すために使用する FIFO のパスです (アップストリーム用)。intdash Edge Agent の設定を行う (p. 6) で H.264 用の fifo_tx として設定した値を指定します。h264_1 から h264_4 の値を指定可能です。

2.4.2 パラメーター設定ファイル params.yaml1 での設定

intdash Bridge の詳しい動作は、launch ファイルから参照されるパラメーター設定ファイル params.yaml1 で設定します。設定例は以下のとおりです。

outgoing (ROS 空間から intdash に向けたデータ転送、アップストリーム用) の設定と、incoming (intdash から ROS 空間に向けたデータ転送、ダウンストリーム用) の設定に分けて記載します。

```
outgoing:
  enabled: true
  buffering: false
  advertise_interval: 5
  max_array_size: 500
```

(次のページに続く)

(前のページからの続き)

```
topics:
- topic_name: "/chatter"
  send_mode: "raw"
  queue_size: 100
- topic_name: "/cmd_vel"
  send_mode: "both"
  queue_size: 100

incoming:
  enabled: true
  queue_size: 100
  suffix: ""
```

outgoing (ROS 空間から intdash に向けたデータ転送)

outgoing は、ROS 空間から intdash に向けたデータ転送に関する設定です。topics に指定されたトピックのメッセージが、intdash Edge Agent を介して intdash へ送信されます (アップストリーム)。topics ではトピックごとに設定を行います。

フィールド名	enum	型	必須／オプション	説明
enabled	true、false	bool	必須	outgoing の処理を有効にするかどうか。
buffering	true、false	bool	オプション (デフォルト false)	FIFO にデータを書き込む際のバッファを有効にするかどうか。なお、intdash Bridge 内部では、バッファは setvbuf 関数により設定されます。
advertise_interval	0: メッセージごとに送信、n: n 秒ごとに送信	number	オプション (デフォルト 0)	トピックの定義情報を送信する間隔を指定します。詳細については表の下の注釈を参照してください。
max_array_size	-	number	オプション (デフォルト 500)	json モードにおいて、配列がここで指定したサイズより大きい場合に、送信せずスキップします。詳細については表の下の注釈を参照してください。

次のページに続く

表 1 – 前のページからの続き

フィールド名	enum	型	必須／オプション	説明
topics[].topic_name	-	string	必須	ROS 空間でサブスクリプトし、intdash に送信するトピックの名前を指定します。
topics[].send_mode	raw: Raw バイナリで送信、json: JSON 文字列で送信、both: Raw バイナリと JSON 文字列双方で送信、jpeg_1: JPEG で送信、h264_1: H.264 で送信	string	必須	このトピックを送信する際の送信モードを指定します。JPEG は jpeg_1 jpeg_2 jpeg_3 jpeg_4、H.264 は h264_1 h264_2 h264_3 h264_4 を指定可能です。
topics[].queue_size	-	string	必須	ROS 空間からメッセージを受け取る際に subscriber が使用するキューの長さです。
topics[].pipeline	-	string	オプション (デフォルト "")	jpeg/h264 モードの場合に実行される GStreamer での処理のパイプラインを設定します。パイプラインの最後は appsink name=appsink emit-signals=true としてください。

注釈: トピックの定義情報の送信間隔 (advertise_interval)

send_mode が raw の場合、データの中にトピックの定義情報が含まれますが、これは固定の情報なのでメッセージごとに送信する必要はありません。advertise_interval を設定することで、トピックの定義情報の送信を間引くことができます。たとえば、advertise_interval: 5 のように設定すると、トピックの定義情報は 5 秒に一度だけ送信されます。

注釈: 配列の最大サイズ (max_array_size)

send_mode が json の場合、map や image など uint8 配列のデータはそのまま数値配列としてエンコードされます。配列サイズが巨大な場合はエンコードに時間を要するため、max_array_size に配列のサイズを指定することにより、それよりも大きな配列をスキップすることができます。

注釈: 「JPEG/H.264 形式」 (jpeg/h264 モード) について

「JPEG/H.264 形式」は、画像 (JPEG) ・動画 (H.264) の送信形式で、sensor_msgs/Image ・sensor_msgs/CompressedImage 画像を送信する際に適した形式です (画像を送信する方法については、チュートリアル [チュートリアル C: 画像の ROS メッセージを JPEG に変換して Data Visualizer で表示する](#) (p. 31)、チュートリアル [D: 画像の ROS メッセージを H.264 に変換して Data Visualizer で表示する](#) (p. 40) も参照してください)。JPEG/H.264 形式は、Visual M2M Data Visualizer での画像・動画閲覧向きです。

incoming (intdash から ROS 空間に向けたデータ転送)

incoming は、intdash から ROS 空間に向けたデータ転送に関する設定です。intdash Bridge は、intdash Edge Agent を介して受け取ったメッセージ (ダウンストリーム) をすべて ROS 空間へパブリッシュするため、トピックごとに設定する項目はありません。

フィールド名	enum	型	必須/オプション	説明
enabled	true 、 false	bool	必須	incoming の処理を有効にするかどうか。
queue_size	-	number	必須	intdash Edge Agent から受け取ったメッセージを ROS にパブリッシュする際に publisher が使用するキューの長さです。
suffix	-	string	必須	デバッグ用の設定です。本設定は空文字列で構いません。

2.5 intdash Edge Agent と intdash Bridge を起動する

[intdash Edge Agent の設定](#) (p. 6) と [intdash Bridge の設定](#) (p. 8) を完了した上で、以下の手順により intdash Edge Agent と intdash Bridge を起動してください。

1. intdash Edge Agent を起動します。

```
$ sudo LD_LIBRARY_PATH=/opt/vm2m/lib /opt/vm2m/sbin/intdash-edge-manager -C manager.conf
```

intdash Edge Agent の起動方法の詳細については、[intdash Edge Agent デベロッパーガイド](#) を参照してください。

2. 以下のコマンドを実行して、roslaunch により intdash Bridge を起動します。

```
$ roslaunch mylaunch.launch
```

03 チュートリアル A: intdash Bridge を使って 2 つの ROS 空間を接続する

このチュートリアルでは、intdash Bridge、intdash Edge Agent、intdash サーバーを使用することにより、インターネットを介して 2 つの ROS 空間を接続します。

publisher と subscriber の例として、[ROS 公式チュートリアル](#)の talker と listener を使用します。

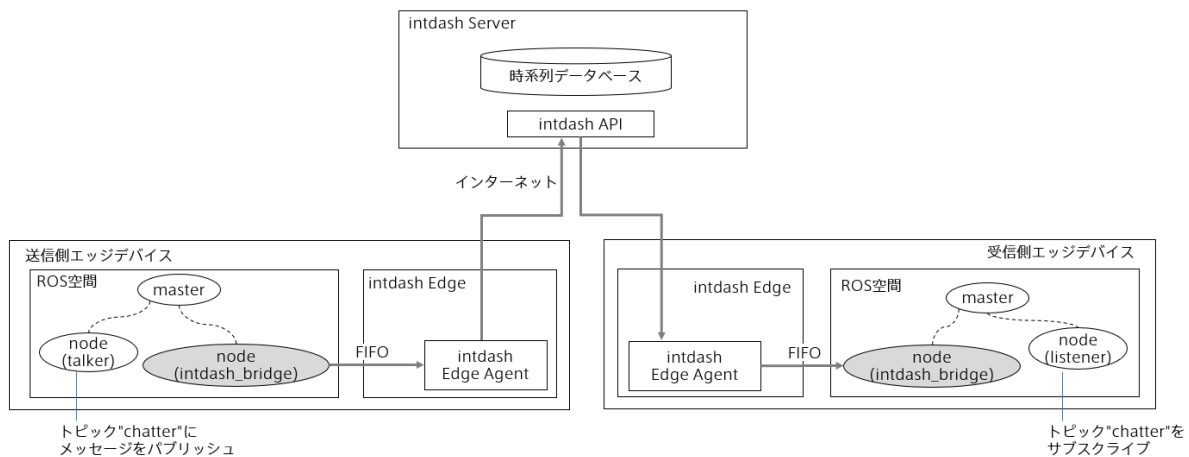


図 2 2 つの ROS 空間を接続する

- 送信側の ROS 空間では、talker が "chatter" というトピックに ROS メッセージをパブリッシュします。
- 受信側の ROS 空間では、listener が、同じく "chatter" というトピックをサブスクライブします。

3.1 送信側エッジを準備する

送信側の環境に、[使用方法](#) (p. 5) の手順に従って intdash Edge Agent および intdash Bridge をインストールしてください。その上で、以下のように設定を行います。

3.1.1 ROS ノードを設定する

1. 以下のコマンドを実行し、ワークスペースとなるディレクトリを作成します

```
$ mkdir -p ~/catkin_ws/src
```

2. ワークスペース内に、beginner_tutorials パッケージを作成します。

```
$ cd ~/catkin_ws  
$ cd src/  
$ catkin_create_pkg beginner_tutorials
```

3. ROS 公式チュートリアルの talker ノードのスクリプトをワークスペースにダウンロードします。

```
$ cd beginner_tutorials/  
$ mkdir scripts  
$ cd scripts/  
$ wget \  
  https://raw.githubusercontent.com/ros/ros_tutorials/indigo-devel/rospy_tutorials/001_talker_listener/talker.py  
$ chmod +x talker.py
```

4. ビルドを実行します。

```
$ cd ~/catkin_ws/  
$ catkin_make
```

5. launch ファイル（~/catkin_ws/sender.launch）を作成し、talker ノードと intdash_bridge ノードの起動を以下のように設定します。

パラメーター `fifo_tx_raw` では、intdash Bridge から intdash Edge Agent にデータを渡すための FIFO（`logger_001.tx`）を設定しています。

sender.launch

```
<launch>  
  <node name="talker" pkg="beginner_tutorials" type="talker.py" />  
  <arg name="paramsfile" default="/opt/vm2m/etc/params.yaml" />  
  <group ns="intdash">  
    <node pkg="intdash_bridge" name="intdash_bridge" type="intdash_bridge_node"  
      output="screen" clear_params="true">  
      <param name="fifo_tx_raw" value="/opt/vm2m/var/run/intdash/logger_001.tx" />  
      <param name="fifo_rx_raw" value="" />  
      <param name="fifo_tx_json" value="" />  
      <rosparam command="load" file="$(arg paramsfile)" />  
    </node>  
  </group>  
</launch>
```

3.1.2 送信側 intdash Edge Agent を設定する

intdash Edge Agent の設定ファイル `manager.conf` で、以下のように設定します。

主な設定内容：

- 送信側エッジの認証情報（クライアントシークレット）と UUID（`my_secret` と `my_id`）
- intdash サーバーのホスト名 + ドメイン名（FQDN）（`host`）
- 受信側エッジの UUID（`dst_id`）
- intdash Bridge からデータを受け取る FIFO（`logger_001.tx` を使用し、チャンネル 1 を付与）

注釈: 旧来のエッジの認証情報（トークン）を使用する場合は、`my_secret` の代わりに `my_token` をご使用ください。

`manager.conf`

```
{
  "manager":{
    "workdirs":[
      "/opt/vm2m/var/lib/intdash/meas",
      "/opt/vm2m/var/run/intdash"
    ],
    "rawdir":"/opt/vm2m/var/lib/intdash/raw",
    "meas_root":"/opt/vm2m/var/lib/intdash/meas",
    "basetime":"/opt/vm2m/var/run/intdash/basetime",
    "stat":"/opt/vm2m/var/run/intdash/manager.stat",
    "system_stat":"/opt/vm2m/var/run/intdash/system.stat",
    "process_stat":"/opt/vm2m/var/run/intdash/process.stat",
    "wwan_stat":"/opt/vm2m/var/run/intdash/wwan.stat",
    "filters":[]
  },
  "clients":[
    {
      "protocol":"mod_websocket.v2",
      "type":"realtime",
      "my_secret":"XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX",
      "my_id":"XXXXXXX-XXX-XXX-XXX-XXXXXXXXXXXX",
      "connection":{
        "host":"xxxxxx.intdash.jp",
        "path":"/api/v1/ws/measurements"
      },
      "dst_id":[
        "YYYYYYYY-YYYY-YYYY-YYYY-YYYYYYYYYYYY"
      ],
      "fifo_rx":"/opt/vm2m/var/run/intdash/client_realtime.rx",
      "fifo_tx":"/opt/vm2m/var/run/intdash/client_realtime.tx",
      "path":"/opt/vm2m/sbin/intdash-edge-client",
      "stat":"/opt/vm2m/var/intdash/client_realtime.stat",
      "fast_net_check_cmd":"/opt/vm2m/bin/intdash-edge-networkd.sh -q -t"
```

(次のページに続く)

(前のページからの続き)

```

    },
    {
      "protocol": "mod_http",
      "type": "resend",
      "my_secret": "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX",
      "my_id": "XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX",
      "connection": {
        "host": "xxxxxxx.intdash.jp",
        "path": "/api/v1/measurements"
      },
      "fifo_rx": "/opt/vm2m/var/run/intdash/client_resend.rx",
      "fifo_tx": "/opt/vm2m/var/run/intdash/client_resend.tx",
      "path": "/opt/vm2m/sbin/intdash-edge-client",
      "stat": "/opt/vm2m/var/run/intdash/client_resend.stat",
      "fast_net_check_cmd": "/opt/vm2m/bin/intdash-edge-networkd.sh -q -t"
    }
  ],
  "loggers": [
    {
      "devicetype": "intdash_bridge",
      "path": "",
      "connections": [
        {
          "fifo_tx": "/opt/vm2m/var/run/intdash/logger_001.tx",
          "channel": 1
        }
      ],
      "details": {
        "plugin": "fifo"
      }
    }
  ]
}

```

3.1.3 送信側 intdash Bridge を設定する

intdash Bridge 用パラメーター設定ファイル (/opt/vm2m/etc/params.yaml) を以下のように作成します。

ここでは、トピック "/chatter" を、raw モードで intdash に送信するように設定しています。

params.yaml

```

outgoing:
  enabled: true
  buffering: false
  advertise_interval: 5
  max_array_size: 500
  topics:

```

(次のページに続く)

(前のページからの続き)

```
- topic_name: "/chatter"
  send_mode: "raw"
  queue_size: 100

incoming:
  enabled: false
  queue_size: 100
  suffix: ""
```

3.2 受信側エッジを準備する

受信側の環境にも、[使用方法](#) (p. 5) の手順に従って intdash Edge Agent および intdash Bridge をインストールしてください。その上で、以下のように設定を行います。

3.2.1 ROS ノードを設定する

1. 以下のコマンドを実行し、ワークスペースとなるディレクトリを作成します

```
$ mkdir -p ~/catkin_ws/src
```

2. ワークスペース内に、beginner_tutorials パッケージを作成します。

```
$ cd ~/catkin_ws
$ cd src/
$ catkin_create_pkg beginner_tutorials
```

3. ROS 公式チュートリアル listener ノードのスクリプトをワークスペースにダウンロードします。

```
$ cd beginner_tutorials/
$ mkdir scripts
$ cd scripts/
$ wget \
  https://raw.githubusercontent.com/ros/ros_tutorials/indigo-devel/rospy_tutorials/001_talker_listener/listener.py
$ chmod +x listener.py
```

4. ビルドを実行します。

```
$ cd ~/catkin_ws/
$ catkin_make
```

5. launch ファイル（~/catkin_ws/receiver.launch）を作成し、listener ノードと intdash_bridge ノードの起動を以下のように設定します。

パラメーター `fifo_rx_raw` では、intdash Bridge が intdash Edge Agent からデータを受け取るための FIFO（`logger_001.rx`）を設定しています。

receiver.launch

```
<launch>
  <node name="listener" pkg="beginner_tutorials" type="listener.py" output="screen" />
  <arg name="paramsfile" default="/opt/vm2m/etc/params.yaml" />

  <node pkg="intdash_bridge" name="intdash_bridge" type="intdash_bridge_node"
    output="screen" clear_params="true">
    <param name="fifo_tx_raw" value="" />
    <param name="fifo_rx_raw" value="/opt/vm2m/var/run/intdash/logger_001.rx" />
    <param name="fifo_tx_json" value="" />
    <rosparm command="load" file="$(arg paramsfile)" />
  </node>
</launch>
```

3.2.2 受信側 intdash Edge Agent を設定する

intdash Edge Agent の設定ファイル manager.conf で、以下のように設定します。

主な設定内容：

- 受信側エッジの認証情報（クライアントシークレット）と UUID（my_secret と my_id）
- 送信側エッジの UUID（ctlr_id）
- intdash サーバーのホスト名 + ドメイン名（FQDN）（host）
- intdash から受け取るデータの設定（ctlr_filts）
- intdash Edge Agent から intdash Bridge にデータを渡すための FIFO（チャンネル 1 のデータを、logger_001.rx に渡す）

注釈: 旧来のエッジの認証情報（トークン）を使用する場合は、my_secret の代わりに my_token をご使用ください。

manager.conf

```
{
  "manager":{
    "meas_root":"/opt/vm2m/var/intdash/meas",
    "rawdir":"/opt/vm2m/var/intdash/raw",
    "basetime":"/opt/vm2m/var/intdash/basetime",
    "stat":"/opt/vm2m/var/intdash/manager.stat",
    "logger_stat":"/opt/vm2m/var/intdash/logger_%03hhu.stat",
    "process_stat":"/opt/vm2m/var/intdash/process.stat",
    "intdash_stat":"/opt/vm2m/var/intdash/intdash.stat",
    "network_stat":"/opt/vm2m/var/intdash/network.stat",
    "system_stat":"/opt/vm2m/var/intdash/system.stat",
    "wwan_stat":"/opt/vm2m/var/intdash/wwan.stat",
    "workdirs":[
      "/opt/vm2m/var/lib/intdash/meas",
      "/opt/vm2m/var/run/intdash"
    ],
  },
}
```

(次のページに続く)

(前のページからの続き)

```
"filters":[],
},
"clients":[
  {
    "path":"/opt/vm2m/sbin/intdash-edge-client",
    "protocol":"mod_websocket.v2",
    "type":"control",
    "fifo_rx":"/opt/vm2m/var/intdash/client_control.rx",
    "fifo_tx":"/opt/vm2m/var/intdash/client_control.tx",
    "stat":"/opt/vm2m/var/intdash/client_control.stat",
    "my_secret":"YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY",
    "my_id":"YYYYYYYY-YYYY-YYYY-YYYY-YYYYYYYYYYYY",
    "connection":{
      "host":"xxxxxxx.intdash.jp",
      "port":443,
      "path":"/api/v1/ws/measurements",
      "ca":"/opt/vm2m/etc/ssl/certs/cacert.pem",
      "ssl":"secure",
      "opts":[]
    },
    "ctrl_id":"XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX",
    "ctrl_flt":[
      {
        "channel":1,
        "dtype":14,
        "ids":[
          "/chatter"
        ]
      }
    ]
  }
],
"loggers":[
  {
    "devicetype":"system",
    "path":"",
    "connections":[
      {
        "channel":255
      }
    ]
  }
],
"details":{
  "plugin":"status",
  "plugin_dir":"/opt/vm2m/lib/plugins",
  "plugin_arg":{
    "mc7430_cli":"/opt/vm2m/sbin/mc7430_cli",
    "stintd":{
      "meas_root":"/opt/vm2m/var/intdash/meas"
```

(次のページに続く)

(前のページからの続き)

```

    },
    "stsys":{
        "storage_dir":"/"
    }
}
},
{
    "connections":[
        {
            "fifo_rx":"/opt/vm2m/var/run/intdash/logger_001.rx",
            "channel":1
        }
    ],
    "details":{
        "plugin":"fifo"
    }
}
]
}

```

注釈: 受信するデータの指定

上記の例のとおり、受信するデータは Control クライアントに関する設定の `ctrl_flts` で指定しています。

- `dtype` では、受信するデータの iSCP データタイプを十進法で指定します。上の設定で指定されている 14 はデータタイプ「Bytes」を指します。
- `ids` では、受信するデータ ID を指定します。

3.2.3 受信側 intdash Bridge を設定する

intdash Bridge 用パラメーター設定ファイル (`/opt/vm2m/etc/params.yaml`) を以下のように作成します。

intdash Bridge は、intdash Edge Agent から受け取ったメッセージをすべて ROS 空間へパブリッシュするため、トピックの指定は必要ありません。

params.yaml

```

outgoing:
  enabled: false

incoming:
  enabled: true
  queue_size: 100
  suffix: ""

```

3.3 メッセージの送信を開始する

送信側エッジと受信側エッジをそれぞれ起動します。

送信側

ワークスペースで `setup.bash` を実行することによりワークスペースの設定を有効にし、`roslaunch` により `talker` と送信側 `intdash Bridge` を起動します。

```
$ cd ~/catkin_ws
$ source devel/setup.bash
$ roslaunch sender.launch
```

別のターミナルで、以下のコマンドを実行して `intdash Edge Agent` を起動します。

```
$ sudo LD_LIBRARY_PATH=/opt/vm2m/lib /opt/vm2m/sbin/intdash-edge-manager -C manager.conf
```

受信側

ワークスペースで `setup.bash` を実行することによりワークスペースの設定を有効にし、`roslaunch` により `listener` と受信側 `intdash Bridge` を起動します。

```
$ cd ~/catkin_ws
$ source devel/setup.bash
$ roslaunch receiver.launch
```

別のターミナルで、以下のコマンドを実行して `intdash Edge Agent` を起動します。

```
$ sudo LD_LIBRARY_PATH=/opt/vm2m/lib /opt/vm2m/sbin/intdash-edge-manager -C manager.conf
```

受信側で以下のような出力が出力されれば成功です。

```
[INFO] [1624435399.116440641]: Upstream disabled.
[INFO] [1624435399.117253533]: Downstream enabled.
[INFO] [1624435399.612276354]: Advertising to /chatter
[INFO] [1624435399.706018]: /listenerI heard hello world 1624435399.67
[INFO] [1624435399.809872]: /listenerI heard hello world 1624435399.77
```

04 チュートリアル B: Data Visualizer で ROS メッセージを表示する

このチュートリアルでは、intdash サーバーに送信される ROS メッセージを Visual M2M Data Visualizer (以下 Data Visualizer) で可視化します。

可視化しやすくするため、このシナリオでは、ROS データの送信時に Raw データではなく JSON 形式を使用します。

注釈: この章は前章のチュートリアルから独立しており、この章のチュートリアルのみを実施することが可能です。

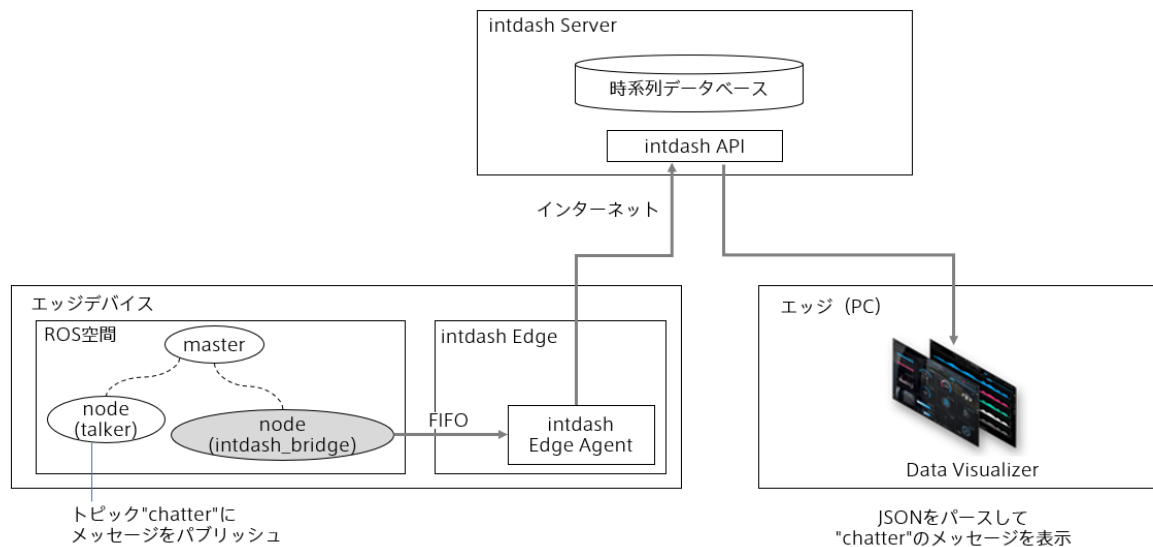


図 3 Data Visualizer で ROS メッセージを表示

4.1 送信側エッジを準備する

送信側の環境に、[使用方法](#) (p. 5) の手順に従って intdash Edge Agent および intdash Bridge をインストールしてください。その上で、以下のように設定を行います。

4.1.1 ROS ノードを設定する

1. 以下のコマンドを実行し、ワークスペースとなるディレクトリを作成します

```
$ mkdir -p ~/catkin_ws/src
```

2. ワークスペース内に、beginner_tutorials パッケージを作成します。

```
$ cd ~/catkin_ws  
$ cd src/  
$ catkin_create_pkg beginner_tutorials
```

3. ROS 公式チュートリアル of talker ノードのスクリプトをワークスペースにダウンロードします。

```
$ cd beginner_tutorials/  
$ mkdir scripts  
$ cd scripts/  
$ wget \  
  https://raw.githubusercontent.com/ros/ros_tutorials/indigo-devel/rospy_tutorials/001_talker_listener/talker.py  
$ chmod +x talker.py
```

4. ビルドを実行します。

```
$ cd ~/catkin_ws/  
$ catkin_make
```

5. launch ファイル（~/catkin_ws/sender.launch）を作成し、talker ノードと intdash_bridge ノードの起動を以下のように設定します。

パラメーター fifo_tx_json では、intdash Bridge から intdash Edge Agent にデータを渡すための FIFO（logger_001.tx）を設定しています。

sender.launch

```
<launch>  
  <node name="talker" pkg="beginner_tutorials" type="talker.py" />  
  <arg name="paramsfile" default="/opt/vm2m/etc/params.yaml" />  
  <group ns="intdash">  
    <node pkg="intdash_bridge" name="intdash_bridge" type="intdash_bridge_node"  
      output="screen" clear_params="true">  
      <param name="fifo_tx_raw" value="" />  
      <param name="fifo_rx_raw" value="" />  
      <param name="fifo_tx_json" value="/opt/vm2m/var/run/intdash/logger_001.tx" />  
      <rosparam command="load" file="$(arg paramsfile)" />  
    </node>  
  </group>  
</launch>
```


4.1.2 送信側 intdash Edge Agent を設定する

intdash Edge Agent の設定ファイル `manager.conf` で、以下のように設定します。

主な設定内容：

- 送信側エッジの認証情報（クライアントシークレット）と UUID（`my_secret` と `my_id`）
- intdash サーバーのホスト名 + ドメイン名（FQDN）（`host`）
- intdash Bridge からデータを受け取る FIFO（`logger_001.tx` を使用し、チャンネル 1 を付与）

注釈: 旧来のエッジの認証情報（トークン）を使用する場合は、`my_secret` の代わりに `my_token` をご使用ください。

`manager.conf`

```
{
  "manager":{
    "workdirs":[
      "/opt/vm2m/var/lib/intdash/meas",
      "/opt/vm2m/var/run/intdash"
    ],
    "rawdir":"/opt/vm2m/var/lib/intdash/raw",
    "meas_root":"/opt/vm2m/var/lib/intdash/meas",
    "basetime":"/opt/vm2m/var/run/intdash/basetime",
    "stat":"/opt/vm2m/var/run/intdash/manager.stat",
    "system_stat":"/opt/vm2m/var/run/intdash/system.stat",
    "process_stat":"/opt/vm2m/var/run/intdash/process.stat",
    "wwan_stat":"/opt/vm2m/var/run/intdash/wwan.stat",
    "filters":[]
  },
  "clients":[
    {
      "protocol":"mod_websocket.v2",
      "type":"realtime",
      "my_secret":"XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX",
      "my_id":"XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX",
      "connection":{
        "host":"xxxxxxx.intdash.jp",
        "path":"/api/v1/ws/measurements"
      },
      "fifo_rx":"/opt/vm2m/var/run/intdash/client_realtime.rx",
      "fifo_tx":"/opt/vm2m/var/run/intdash/client_realtime.tx",
      "path":"/opt/vm2m/sbin/intdash-edge-client",
      "stat":"/opt/vm2m/var/intdash/client_realtime.stat",
      "fast_net_check_cmd":"/opt/vm2m/bin/intdash-edge-networkd.sh -q -t"
    },
    {
      "protocol":"mod_http",
      "type":"resend",
```

(次のページに続く)

(前のページからの続き)

```
"my_secret": "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX",
"my_id": "XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX",
"connection": {
  "host": "xxxxxxx.intdash.jp",
  "path": "/api/v1/measurements"
},
"fifo_rx": "/opt/vm2m/var/run/intdash/client_resend.rx",
"fifo_tx": "/opt/vm2m/var/run/intdash/client_resend.tx",
"path": "/opt/vm2m/sbin/intdash-edge-client",
"stat": "/opt/vm2m/var/run/intdash/client_resend.stat",
"fast_net_check_cmd": "/opt/vm2m/bin/intdash-edge-networkd.sh -q -t"
}
],
"loggers": [
  {
    "devicetype": "intdash_bridge",
    "path": "",
    "connections": [
      {
        "fifo_tx": "/opt/vm2m/var/run/intdash/logger_001.tx",
        "channel": 1
      }
    ],
    "details": {
      "plugin": "fifo"
    }
  }
]
}
```

4.1.3 送信側 intdash Bridge を設定する

intdash Bridge 用パラメーター設定ファイル (/opt/vm2m/etc/params.yaml) を以下のように作成します。

ここでは、トピック "/chatter" を、json モードで intdash に送信するように設定しています。

params.yaml

```
outgoing:
  enabled: true
  buffering: false
  advertise_interval: 5
  max_array_size: 500
  topics:
  - topic_name: "/chatter"
    send_mode: "json"
    queue_size: 100
```

(次のページに続く)

(前のページからの続き)

```
incoming:
  enabled: false
  queue_size: 100
  suffix: ""
```

json モードを指定したことにより、ROS メッセージは、以下のようなデータポイントとして intdash サーバーに送信されます。hello world の後の数値は時刻です。

- データタイプ: String
- データ ID: /chatter
- チャンネル: 1


```
{"msg":{"data":"hello world 1642497396.37"}}
```

4.2 Data Visualizer で JSON 用のデータ設定を準備する

重要:

- intdash Bridge v2.0.0 以降の JSON 形式は、それより前のバージョンとの互換性がありません。

JSON データを Data Visualizer で可視化するためには、JSON データをパースしてメッセージを取り出す設定を作成する必要があります。

1. Data Visualizer 画面左側の [Data Settings] () をクリックします。
2. [Add Group] をクリックします。

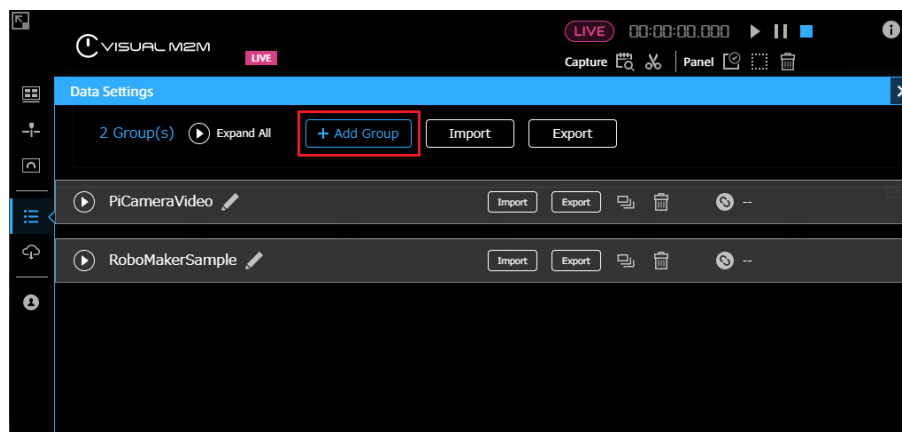


図 4 グループの追加

New Data Group が追加されます。

3. New Data Group の [Add Data] をクリックします。

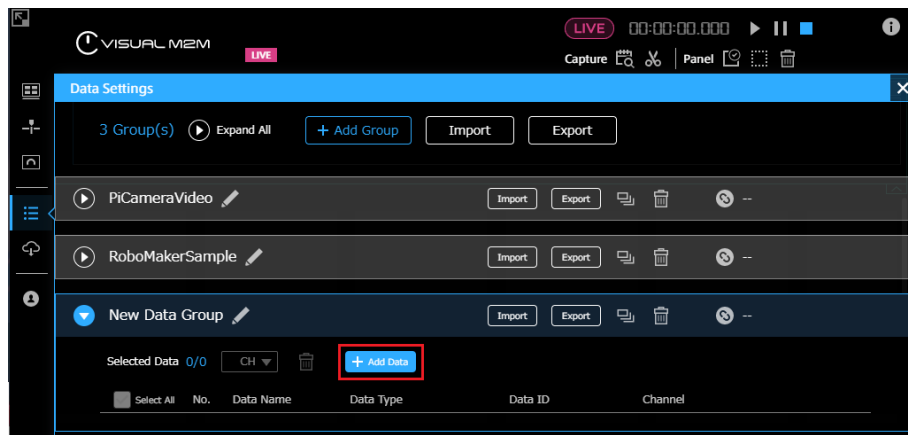


図 5 データの追加

4. String 型データを JSON としてパースし、msg 内の data の値を文字列として取り出す設定をします。

- Data Name: 分かりやすい任意の名前
- Target Data:
 - Data Type: String
 - Data ID: /chatter
 - Channel: 1
- Conversion Settings:
 - Conversion Type: As JSON
 - Field Path: msg.data
 - Value Type: String

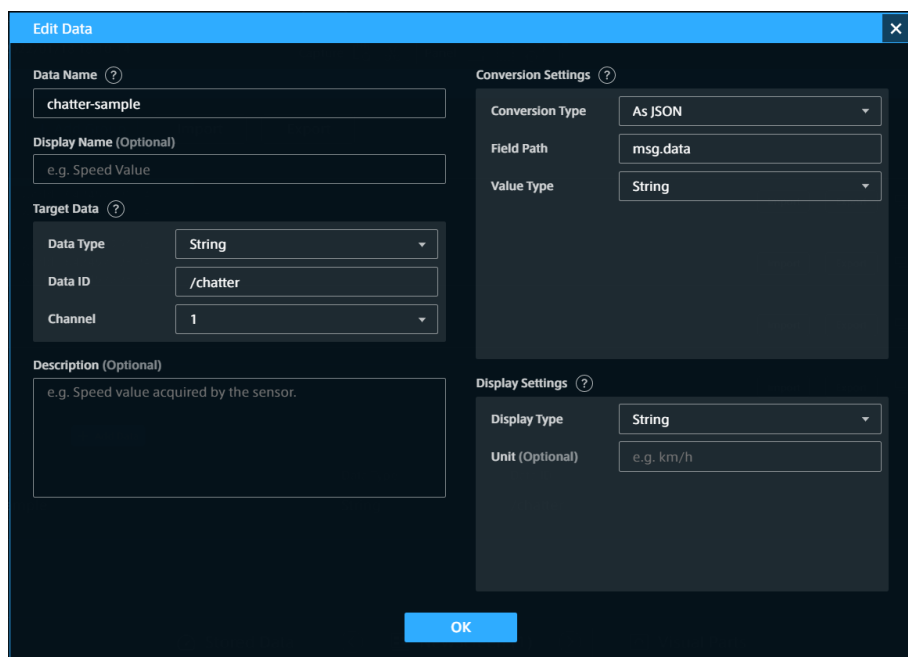


図 6 データ設定

6. [OK] をクリックし、[x] をクリックして、元の画面に戻ります。

以上で、データ設定の準備は完了です。

4.3 Data Visualizer でビジュアルパーツを配置する

1. Data Visualizer 上に、文字列を表示することができるビジュアルパーツを配置します。ここでは例として Text Stream を使用します。

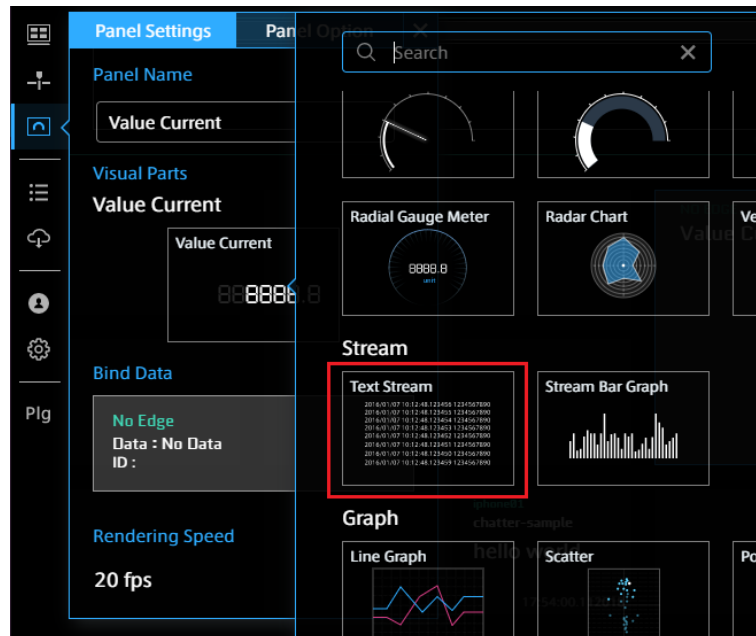


図 7 Text Stream

2. Data Visualizer で JSON 用のデータ設定を準備する (p. 27) で作成した設定を使って、送信側エッジからのデータをビジュアルパーツにバインドします。

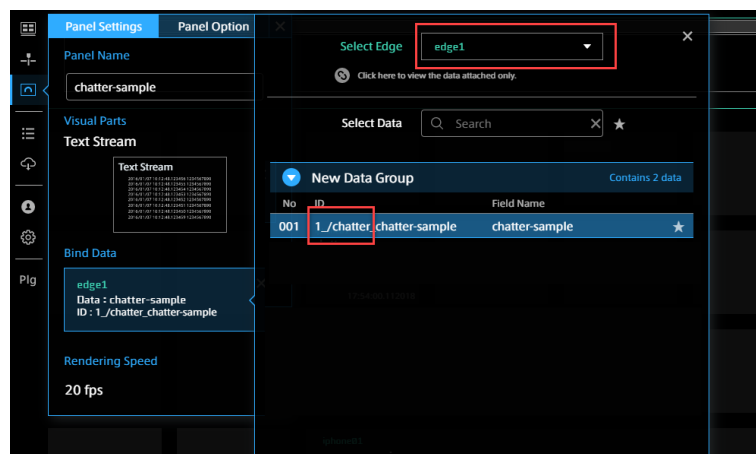


図 8 送信側エッジを選択し、表示したいデータを選択する

3. ライブモードになっていることを確認し (**(LIVE)** アイコンがピンク色)、▶ をクリックして、表示を開始します。

4.4 メッセージの送信を開始する

ワークスペースで `setup.bash` を実行することによりワークスペースの設定を有効にし、`roslaunch` により `talker` と送信側 `intdash Bridge` を起動します。

```
$ cd ~/catkin_ws  
$ source devel/setup.bash  
$ roslaunch sender.launch
```

別のターミナルで、以下のコマンドを実行して `intdash Edge Agent` を起動します。

```
$ sudo LD_LIBRARY_PATH=/opt/vm2m/lib /opt/vm2m/sbin/intdash-edge-manager -C manager.conf
```

Data Visualizer に、送信側エッジからのメッセージが表示されれば成功です。

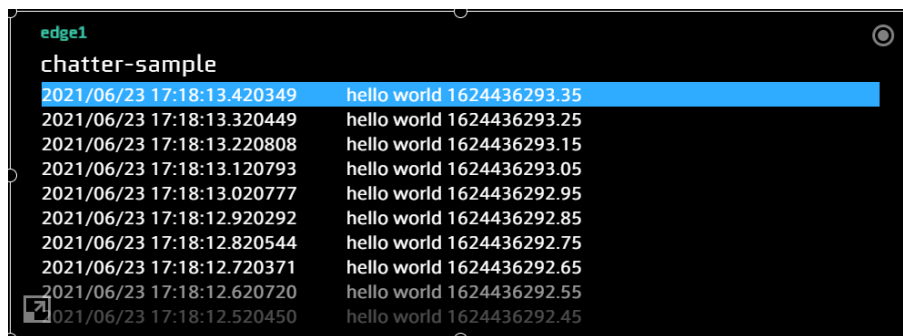


図 9 Text Stream でのメッセージの表示

05 チュートリアル C: 画像の ROS メッセージを JPEG に変換して Data Visualizer で表示する

このチュートリアルでは、画像データの ROS メッセージを JPEG に変換して intdash サーバーに送信し、Visual M2M Data Visualizer（以下 Data Visualizer）で可視化します。

JPEG 画像として可視化可能な ROS メッセージは以下のとおりです：

- sensor_msgs/Image
- sensor_msgs/CompressedImage

注釈: この章は前章のチュートリアルから独立しており。この章のチュートリアルのみを実施することが可能です。

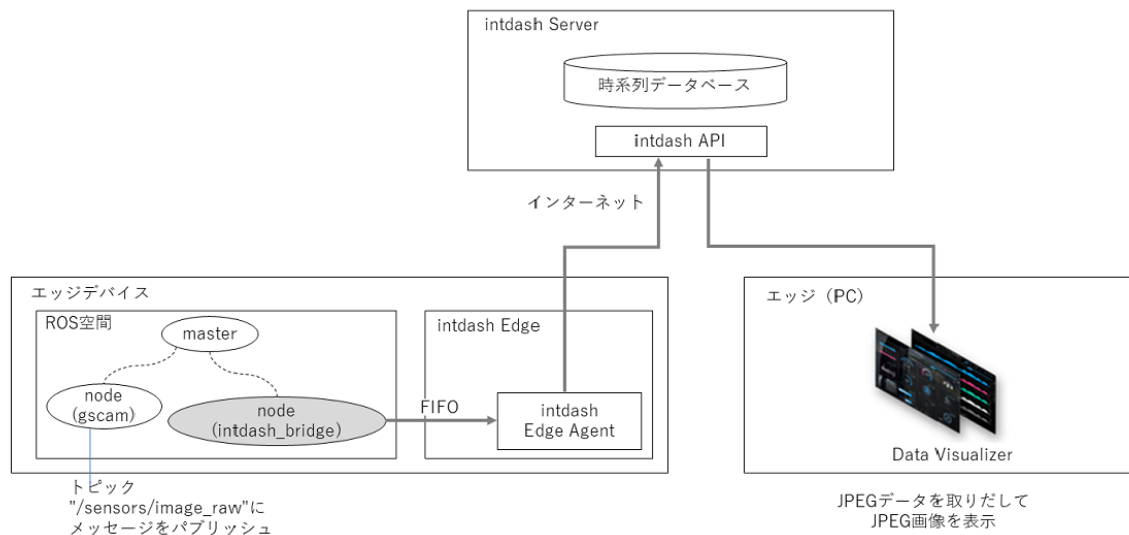


図 10 画像データの ROS メッセージを JPEG に変換して Data Visualizer で表示

5.1 送信側エッジを準備する

送信側の環境に、[使用方法](#) (p. 5) の手順に従って intdash Edge Agent および intdash Bridge をインストールしてください。その上で、以下のように設定を行います。

5.1.1 ROS ノードを設定する

1. 以下のコマンドを実行し、ワークスペースとなるディレクトリを作成します

```
$ mkdir -p ~/catkin_ws/src
```

2. ワークスペース内に、gscam (ROS Camera driver for GStreamer-based video streams) をダウンロードします。

```
$ cd ~/catkin_ws
$ cd src/
$ git clone https://github.com/ros-drivers/gscam
$ cd gscam
$ sed -e "s/EXTRA_CMAKE_FLAGS = -DUSE_ROSBUILD:BOOL=1$/EXTRA_CMAKE_FLAGS = -DUSE_ROSBUILD:BOOL=1 -
↪DGSTREAMER_VERSION_1_x=0n/" -i Makefile
```

3. ビルドを実行します。

```
$ cd ~/catkin_ws/
$ sudo rosdep install --from-paths src --ignore-src -y
$ catkin_make
```

4. 画像変換に必要な GStreamer プラグインをインストールします。

```
$ sudo apt-get install gstreamer1.0-plugins-good
```

5. launch ファイル（~/catkin_ws/sender.launch）を作成し、画像送信ノードと intdash_bridge ノードの起動を以下のように設定します。

パラメーター fifo_tx_jpeg_1 では、intdash Bridge から intdash Edge Agent にデータを渡すための FIFO (logger_001.tx) を設定しています。

sender.launch

```
<launch>
  <arg name="WIDTH" default="640"/>
  <arg name="HEIGHT" default="480"/>
  <arg name="FPS" default="1/1"/>
  <arg name="PUBLISH_FRAME" default="false"/>
  <arg name="aptpod_camera_name" default="/sensors/image_raw" />
  <node ns="v4l" name="gscam_driver_v4l" pkg="gscam" type="gscam" output="screen">
    <param name="camera_name" value="default"/>
    <param name="gscam_config" value="videotestsrc pattern=smtpe75 ! video/x-raw,format=YUY2,width=
↪$(arg WIDTH),height=$(arg HEIGHT),framerate=$(arg FPS) ! videoconvert"/>
    <param name="frame_id" value="/v4l_frame"/>
    <param name="sync_sink" value="true"/>
    <remap from="camera/image_raw" to="$(arg aptpod_camera_name)"/>
  </node>

  <arg name="paramsfile" default="/opt/vm2m/etc/params.yaml" />
```

(次のページに続く)

(前のページからの続き)

```
<group ns="intdash">
  <node pkg="intdash_bridge" name="intdash_bridge" type="intdash_bridge_node"
    output="screen" clear_params="true">
    <param name="fifo_tx_raw" value="$(env INTDASH_EDGE_ROOT)/opt/vm2m/var/run/intdash/logger_200.
↪tx" />
    <param name="fifo_rx_raw" value="$(env INTDASH_EDGE_ROOT)/opt/vm2m/var/run/intdash/logger_200.
↪tx" />
    <param name="fifo_tx_json" value="$(env INTDASH_EDGE_ROOT)/opt/vm2m/var/run/intdash/logger_201.
↪tx" />
    <param name="fifo_tx_jpeg_1" value="$(env INTDASH_EDGE_ROOT)/opt/vm2m/var/run/intdash/logger_
↪001.tx" />
    <rosparam command="load" file="$(arg paramsfile)" />
  </node>
</group>
</launch>
```

5.1.2 送信側 intdash Edge Agent を設定する

intdash Edge Agent の設定ファイル `manager.conf` で、以下のように設定します。

主な設定内容：

- 送信側エッジの認証情報（クライアントシークレット）と UUID（`my_secret` と `my_id`）
- intdash サーバーのホスト名 + ドメイン名（FQDN）（`host`）
- intdash Bridge からデータを受け取る FIFO（`logger_001.tx` を使用し、チャンネル 1 を付与）

注釈: 旧来のエッジの認証情報（トークン）を使用する場合は、`my_secret` の代わりに `my_token` をご使用ください。

`manager.conf`

```
{
  "manager":{
    "workdirs":[
      "/opt/vm2m/var/lib/intdash/meas",
      "/opt/vm2m/var/run/intdash"
    ],
    "rawdir":"/opt/vm2m/var/lib/intdash/raw",
    "meas_root":"/opt/vm2m/var/lib/intdash/meas",
    "basetime":"/opt/vm2m/var/run/intdash/basetime",
    "stat":"/opt/vm2m/var/run/intdash/manager.stat",
    "system_stat":"/opt/vm2m/var/run/intdash/system.stat",
    "process_stat":"/opt/vm2m/var/run/intdash/process.stat",
    "wwan_stat":"/opt/vm2m/var/run/intdash/wwan.stat",
    "filters":[]
  },
  "clients":[
    {
      "protocol":"mod_websocket.v2",
      "type":"realtime",
      "my_secret":"XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX",
      "my_id":"XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX",
      "connection":{
        "host":"xxxxxxx.intdash.jp",
        "path":"/api/v1/ws/measurements"
      },
      "fifo_rx":"/opt/vm2m/var/run/intdash/client_realtime.rx",
      "fifo_tx":"/opt/vm2m/var/run/intdash/client_realtime.tx",
      "path":"/opt/vm2m/sbin/intdash-edge-client",
      "stat":"/opt/vm2m/var/intdash/client_realtime.stat",
      "fast_net_check_cmd":"/opt/vm2m/bin/intdash-edge-networkd.sh -q -t"
    },
    {
      "protocol":"mod_http",
      "type":"resend",
```

(次のページに続く)

(前のページからの続き)

```

    "my_secret": "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX",
    "my_id": "XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX",
    "connection": {
      "host": "xxxxxxx.intdash.jp",
      "path": "/api/v1/measurements"
    },
    "fifo_rx": "/opt/vm2m/var/run/intdash/client_resend.rx",
    "fifo_tx": "/opt/vm2m/var/run/intdash/client_resend.tx",
    "path": "/opt/vm2m/sbin/intdash-edge-client",
    "stat": "/opt/vm2m/var/run/intdash/client_resend.stat",
    "fast_net_check_cmd": "/opt/vm2m/bin/intdash-edge-networkd.sh -q -t"
  }
],
"loggers": [
  {
    "devicetype": "intdash_bridge",
    "path": "",
    "connections": [
      {
        "fifo_tx": "/opt/vm2m/var/run/intdash/logger_001.tx",
        "channel": 1
      }
    ],
    "details": {
      "plugin": "fifo"
    }
  }
]
}

```

5.1.3 送信側 intdash Bridge を設定する

intdash Bridge 用パラメーター設定ファイル (/opt/vm2m/etc/params.yaml) を以下のように作成します。

ここでは、トピック "/sensors/image_raw" を、jpeg モードで intdash に送信するように設定しています。

params.yaml

```

outgoing:
  enabled: true
  buffering: false
  advertise_interval: 5
  max_array_size: 500
  topics:
  - topic_name: "/sensors/image_raw"
    send_mode: "jpeg_1"
    queue_size: 100
    pipeline: "jpegenc ! appsink name=appsink emit-signals=true"

```

(次のページに続く)

(前のページからの続き)


```
incoming:
  enabled: false
  queue_size: 100
  suffix: ""
```

jpeg モードを指定したことにより、ROS メッセージは以下のようなデータポイントとして intdash サーバーに送信されます。

- データタイプ: JPEG
- データ ID: jpeg
- チャンネル: 1

5.2 Data Visualizer で JPEG 用のデータ設定を準備する

JPEG を Data Visualizer で可視化するためには、JPEG 用のデータ設定が必要です。

1. Data Visualizer 画面左側の [Data Settings] () をクリックします。
2. [Add Group] をクリックします。

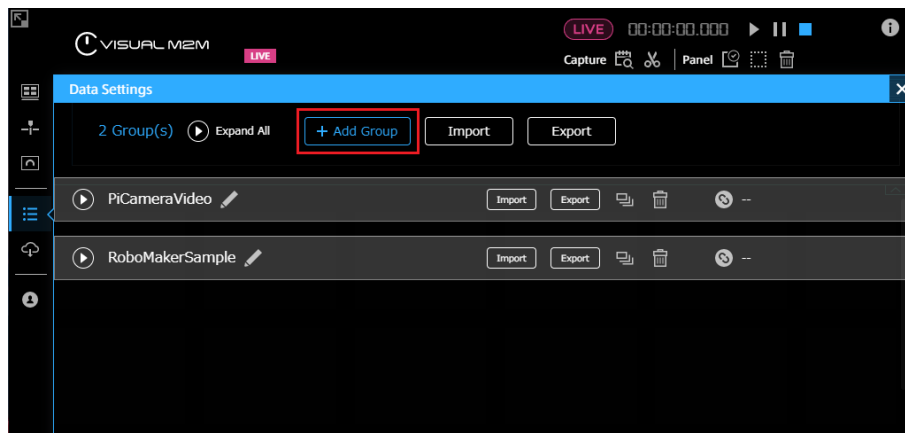


図 11 グループの追加

New Data Group が追加されます。

3. New Data Group の [Add Data] をクリックします。

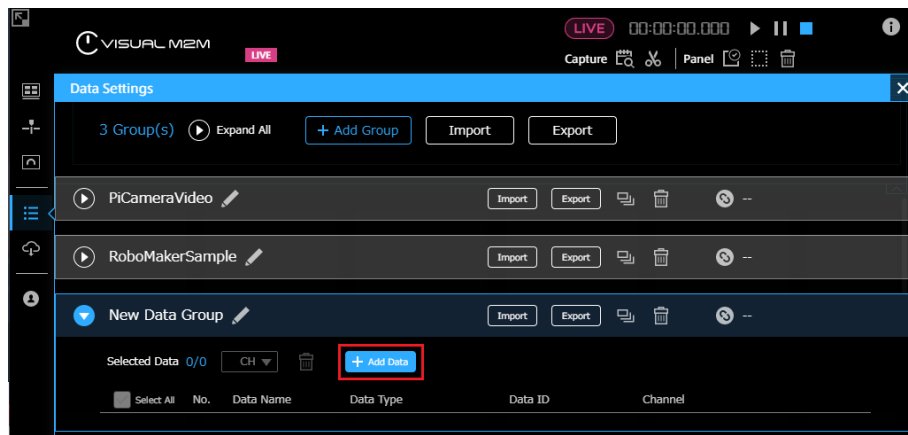


図 12 データの追加

4. データポイントを JPEG 型データとして扱うための設定をします。

- Data Name: 分かりやすい任意の名前
- Target Data:
 - Data Type: VIDEO(H.264, JPEG)
 - Data ID: VIDEO
 - Channel: 1

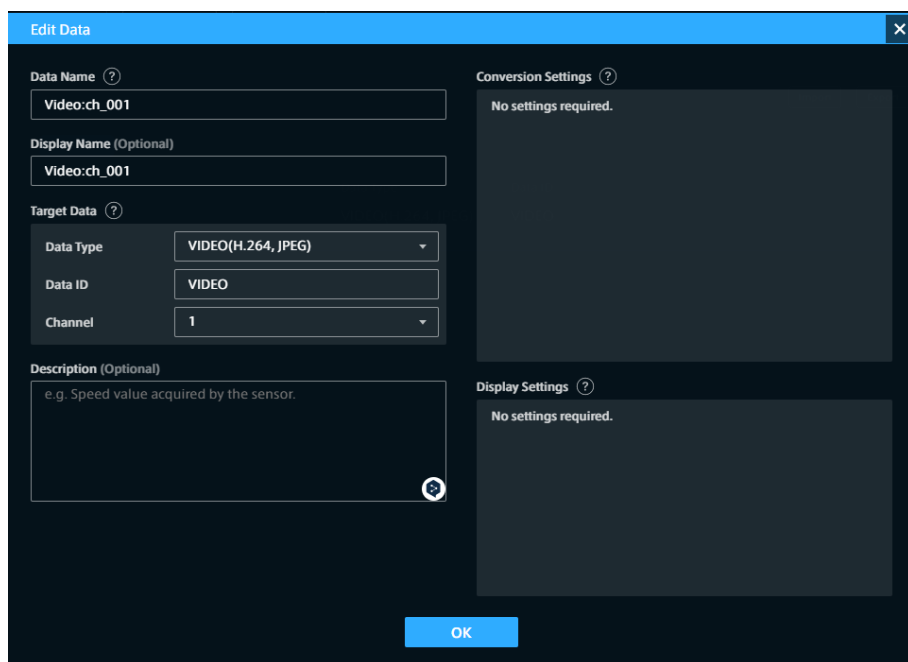


図 13 データ設定

6. [OK] をクリックし、[x] をクリックして、元の画面に戻ります。

以上で、データ設定の準備は完了です。

5.3 Data Visualizer でビジュアルパーツを配置する

1. Data Visualizer 上に Image Viewer を配置します。

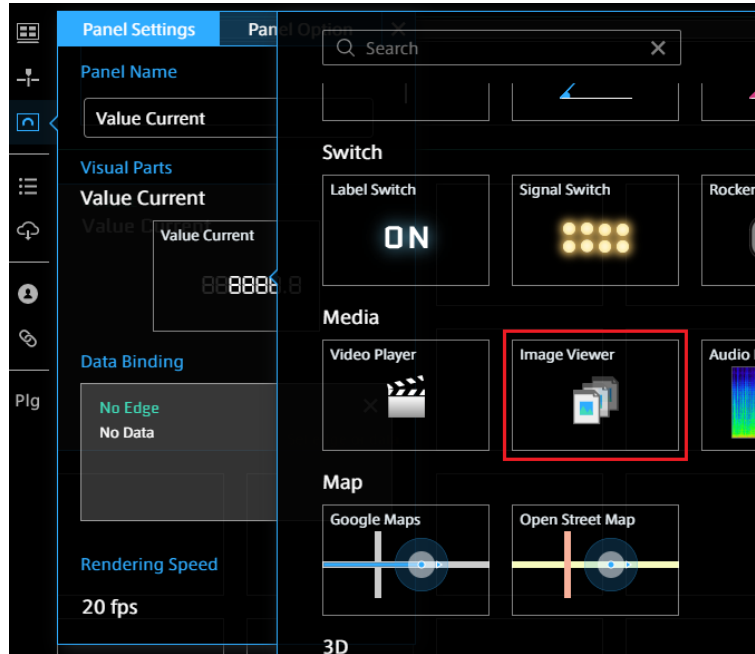


図 14 Image Viewer

2. Data Visualizer で JPEG 用のデータ設定を準備する (p. 36) で作成した設定を使って、送信側エッジからのデータをビジュアルパーツにバインドします。

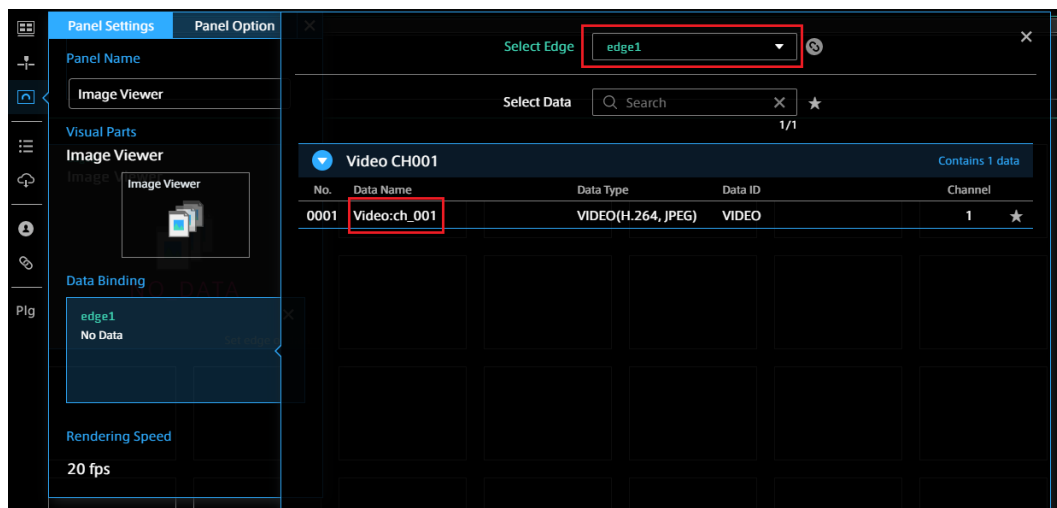


図 15 送信側エッジを選択し、表示したいデータを選択する

3. ライブモードになっていることを確認し (**LIVE** アイコンがピンク色)、▶ をクリックして、表示を開始します。

5.4 メッセージの送信を開始する

ワークスペースで `setup.bash` を実行することによりワークスペースの設定を有効にし、`roslaunch` により画像送信ノードと送信側 intdash Bridge を起動します。

```
$ cd ~/catkin_ws  
$ source devel/setup.bash  
$ roslaunch sender.launch
```

別のターミナルで、以下のコマンドを実行して intdash Edge Agent を起動します。

```
$ sudo LD_LIBRARY_PATH=/opt/vm2m/lib /opt/vm2m/sbin/intdash-edge-manager -C manager.conf
```

Data Visualizer に、送信側エッジからの JPEG 画像が表示されれば成功です。

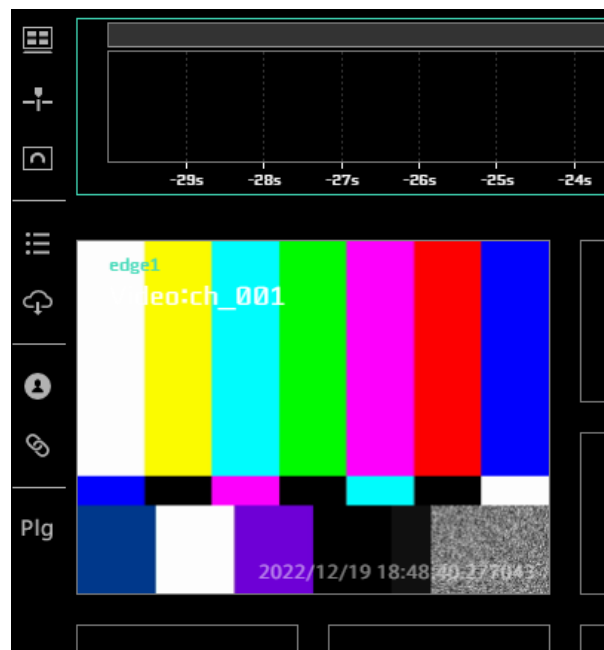


図 16 Image Viewer での画像の表示

06 チュートリアル D: 画像の ROS メッセージを H.264 に変換して Data Visualizer で表示する

このチュートリアルでは、画像データの ROS メッセージを H.264 に変換して intdash サーバーに送信し、Visual M2M Data Visualizer（以下 Data Visualizer）で可視化します。

H.264 動画として可視化可能な ROS メッセージは以下のとおりです：

- sensor_msgs/Image
- sensor_msgs/CompressedImage

注釈: この章は前章のチュートリアルから独立しており。この章のチュートリアルのみを実施することが可能です。

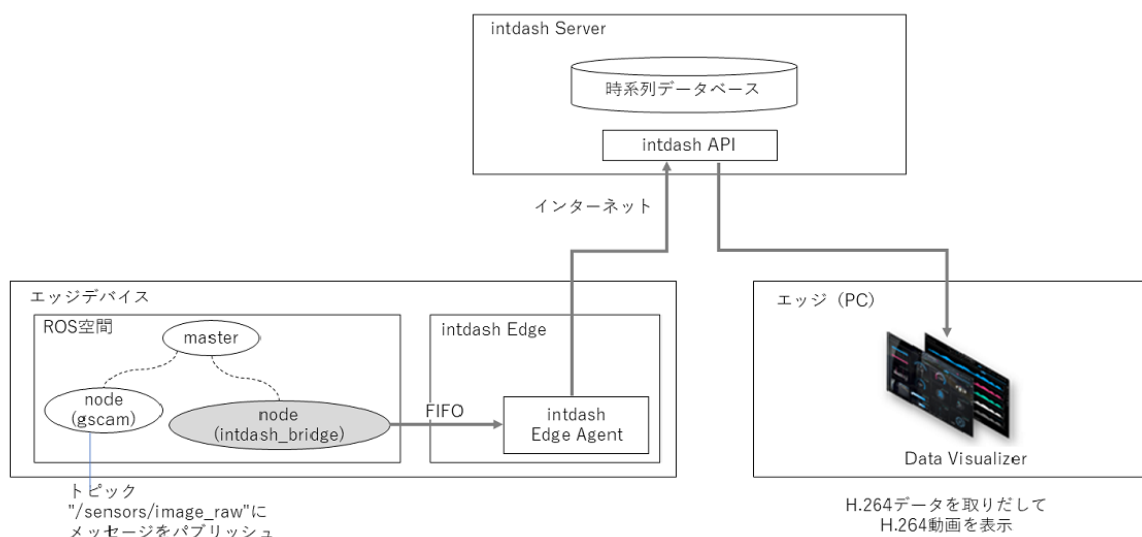


図 17 画像データの ROS メッセージを H.264 に変換して Data Visualizer で表示

6.1 送信側エッジを準備する

送信側の環境に、[使用方法](#) (p. 5) の手順に従って intdash Edge Agent および intdash Bridge をインストールしてください。その上で、以下のように設定を行います。

6.1.1 ROS ノードを設定する

1. 以下のコマンドを実行し、ワークスペースとなるディレクトリを作成します

```
$ mkdir -p ~/catkin_ws/src
```

2. ワークスペース内に、gscam (ROS Camera driver for GStreamer-based video streams) をダウンロードします。

```
$ cd ~/catkin_ws
$ cd src/
$ git clone https://github.com/ros-drivers/gscam
$ cd gscam
$ sed -e "s/EXTRA_CMAKE_FLAGS = -DUSE_ROSBUILD:BOOL=1$/EXTRA_CMAKE_FLAGS = -DUSE_ROSBUILD:BOOL=1 -
↪DGSTREAMER_VERSION_1_x=0n/" -i Makefile
```

3. ビルドを実行します。

```
$ cd ~/catkin_ws/
$ sudo rosdep install --from-paths src --ignore-src -y
$ catkin_make
```

4. 画像変換に必要な GStreamer プラグインをインストールします。

```
$ sudo apt-get install gstreamer1.0-plugins-good gstreamer1.0-plugins-bad gstreamer1.0-plugins-ugly
```

5. launch ファイル（~/catkin_ws/sender.launch）を作成し、画像送信ノードと intdash_bridge ノードの起動を以下のように設定します。

パラメーター fifo_tx_h264_1 では、intdash Bridge から intdash Edge Agent にデータを渡すための FIFO (logger_001.tx) を設定しています。

sender.launch

```
<launch>
  <arg name="WIDTH" default="640"/>
  <arg name="HEIGHT" default="480"/>
  <arg name="FPS" default="1/1"/>
  <arg name="PUBLISH_FRAME" default="false"/>
  <arg name="aptpod_camera_name" default="/sensors/image_raw" />
  <node ns="v4l" name="gscam_driver_v4l" pkg="gscam" type="gscam" output="screen">
    <param name="camera_name" value="default"/>
    <param name="gscam_config" value="videotestsrc pattern=smtpe75 ! video/x-raw,format=YUY2,width=
↪$(arg WIDTH),height=$(arg HEIGHT),framerate=$(arg FPS) ! videoconvert"/>
    <param name="frame_id" value="/v4l_frame"/>
    <param name="sync_sink" value="true"/>
    <remap from="camera/image_raw" to="$(arg aptpod_camera_name)"/>
  </node>

  <arg name="paramsfile" default="/opt/vm2m/etc/params.yaml" />
```

(次のページに続く)

(前のページからの続き)

```
<group ns="intdash">
  <node pkg="intdash_bridge" name="intdash_bridge" type="intdash_bridge_node"
    output="screen" clear_params="true">
    <param name="fifo_tx_raw" value="$(env INTDASH_EDGE_ROOT)/opt/vm2m/var/run/intdash/logger_200.
↪tx" />
    <param name="fifo_rx_raw" value="$(env INTDASH_EDGE_ROOT)/opt/vm2m/var/run/intdash/logger_200.
↪tx" />
    <param name="fifo_tx_json" value="$(env INTDASH_EDGE_ROOT)/opt/vm2m/var/run/intdash/logger_201.
↪tx" />
    <param name="fifo_tx_h264_1" value="$(env INTDASH_EDGE_ROOT)/opt/vm2m/var/run/intdash/logger_
↪001.tx" />
    <roscparam command="load" file="$(arg paramsfile)" />
  </node>
</group>
</launch>
```

6.1.2 送信側 intdash Edge Agent を設定する

intdash Edge Agent の設定ファイル `manager.conf` で、以下のように設定します。

主な設定内容：

- 送信側エッジの認証情報（クライアントシークレット）と UUID（`my_secret` と `my_id`）
- intdash サーバーのホスト名 + ドメイン名（FQDN）（`host`）
- intdash Bridge からデータを受け取る FIFO（`logger_001.tx` を使用し、チャンネル 1 を付与）

注釈: 旧来のエッジの認証情報（トークン）を使用する場合は、`my_secret` の代わりに `my_token` をご使用ください。

`manager.conf`

```
{
  "manager":{
    "workdirs":[
      "/opt/vm2m/var/lib/intdash/meas",
      "/opt/vm2m/var/run/intdash"
    ],
    "rawdir":"/opt/vm2m/var/lib/intdash/raw",
    "meas_root":"/opt/vm2m/var/lib/intdash/meas",
    "basetime":"/opt/vm2m/var/run/intdash/basetime",
    "stat":"/opt/vm2m/var/run/intdash/manager.stat",
    "system_stat":"/opt/vm2m/var/run/intdash/system.stat",
    "process_stat":"/opt/vm2m/var/run/intdash/process.stat",
    "wwan_stat":"/opt/vm2m/var/run/intdash/wwan.stat",
    "filters":[]
  },
  "clients":[
    {
      "protocol":"mod_websocket.v2",
      "type":"realtime",
      "my_secret":"XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX",
      "my_id":"XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX",
      "connection":{
        "host":"xxxxxxx.intdash.jp",
        "path":"/api/v1/ws/measurements"
      },
      "fifo_rx":"/opt/vm2m/var/run/intdash/client_realtime.rx",
      "fifo_tx":"/opt/vm2m/var/run/intdash/client_realtime.tx",
      "path":"/opt/vm2m/sbin/intdash-edge-client",
      "stat":"/opt/vm2m/var/intdash/client_realtime.stat",
      "fast_net_check_cmd":"/opt/vm2m/bin/intdash-edge-networkd.sh -q -t"
    },
    {
      "protocol":"mod_http",
      "type":"resend",
```

(次のページに続く)

(前のページからの続き)

```

    "my_secret": "XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX",
    "my_id": "XXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX",
    "connection": {
      "host": "xxxxxxx.intdash.jp",
      "path": "/api/v1/measurements"
    },
    "fifo_rx": "/opt/vm2m/var/run/intdash/client_resend.rx",
    "fifo_tx": "/opt/vm2m/var/run/intdash/client_resend.tx",
    "path": "/opt/vm2m/sbin/intdash-edge-client",
    "stat": "/opt/vm2m/var/run/intdash/client_resend.stat",
    "fast_net_check_cmd": "/opt/vm2m/bin/intdash-edge-networkd.sh -q -t"
  }
],
"loggers": [
  {
    "devicetype": "intdash_bridge",
    "path": "",
    "connections": [
      {
        "fifo_tx": "/opt/vm2m/var/run/intdash/logger_001.tx",
        "channel": 1
      }
    ],
    "details": {
      "plugin": "fifo"
    }
  }
]
}

```

6.1.3 送信側 intdash Bridge を設定する

intdash Bridge 用パラメーター設定ファイル (/opt/vm2m/etc/params.yaml) を以下のように作成します。

ここでは、トピック "/sensors/image_raw" を、h264 モードで intdash に送信するように設定しています。

params.yaml

```

outgoing:
  enabled: true
  buffering: false
  advertise_interval: 5
  max_array_size: 500
  topics:
  - topic_name: "/sensors/image_raw"
    send_mode: "h264_1"
    queue_size: 100
    pipeline : "\

```

(次のページに続く)

(前のページからの続き)

```

videoconvert ! \
videorate ! \
video/x-raw,framerate=5/1 ! \
x264enc ! \
video/x-h264, stream-format=byte-stream ! \
queue ! \
h264parse ! \
queue ! \
appsink name=appsink emit-signals=true"

```

incoming:

```

enabled: false
queue_size: 100
suffix: ""


```

h264 モードを指定したことにより、ROS メッセージは以下のようなデータポイントとして intdash サーバーに送信されます。

- データタイプ: H.264
- データ ID: 0x0 0x1 0x8
- チャンネル: 1

6.2 Data Visualizer で H.264 用のデータ設定を準備する

H.264 を Data Visualizer で可視化するためには、H.264 用のデータ設定が必要です。

1. Data Visualizer 画面左側の [Data Settings] () をクリックします。
2. [Add Group] をクリックします。

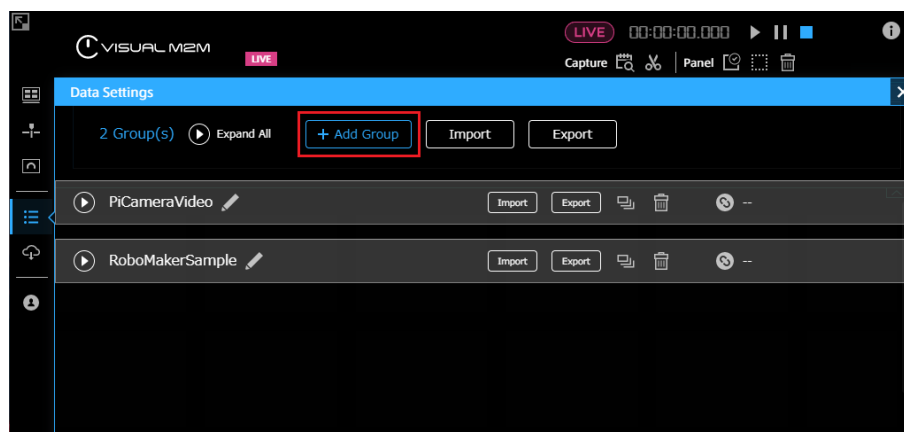


図 18 グループの追加

New Data Group が追加されます。

3. New Data Group の [Add Data] をクリックします。

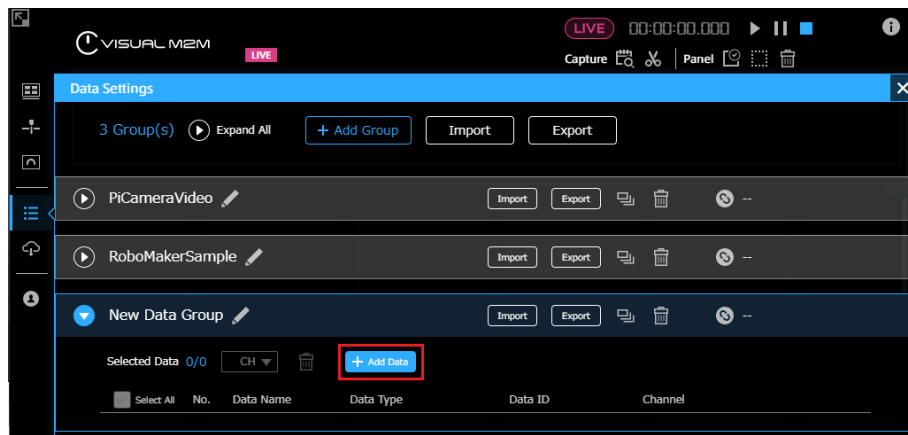


図 19 データの追加

4. データポイントを H.264 型データとして扱うための設定をします。

- Data Name: 分かりやすい任意の名前
- Target Data:
 - Data Type: VIDEO(H.264, JPEG)
 - Data ID: VIDEO
 - Channel: 1

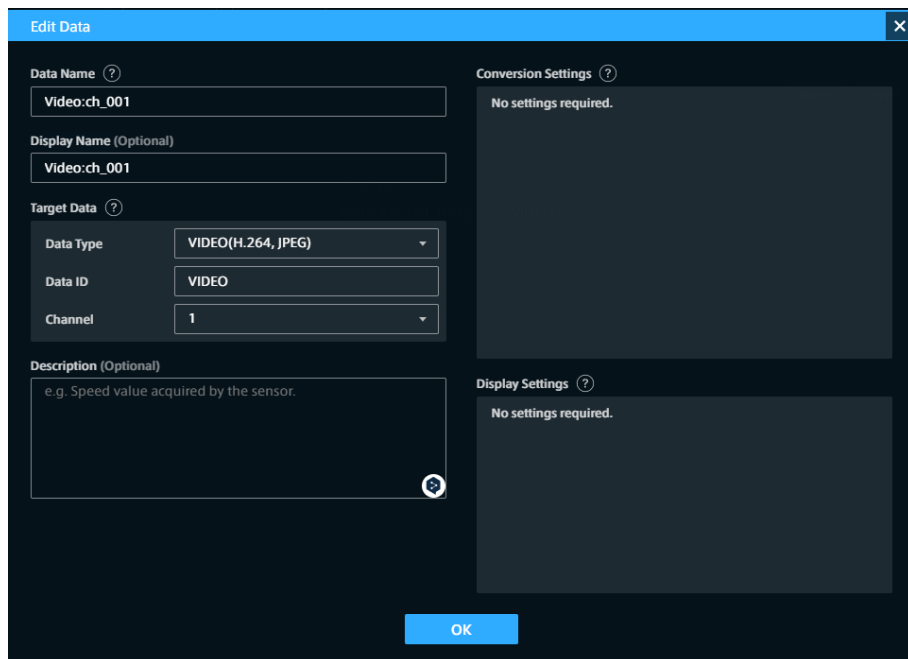


図 20 データ設定

6. [OK] をクリックし、[x] をクリックして、元の画面に戻ります。

以上で、データ設定の準備は完了です。

6.3 Data Visualizer でビジュアルパーツを配置する

1. Data Visualizer 上に Video Player を配置します。

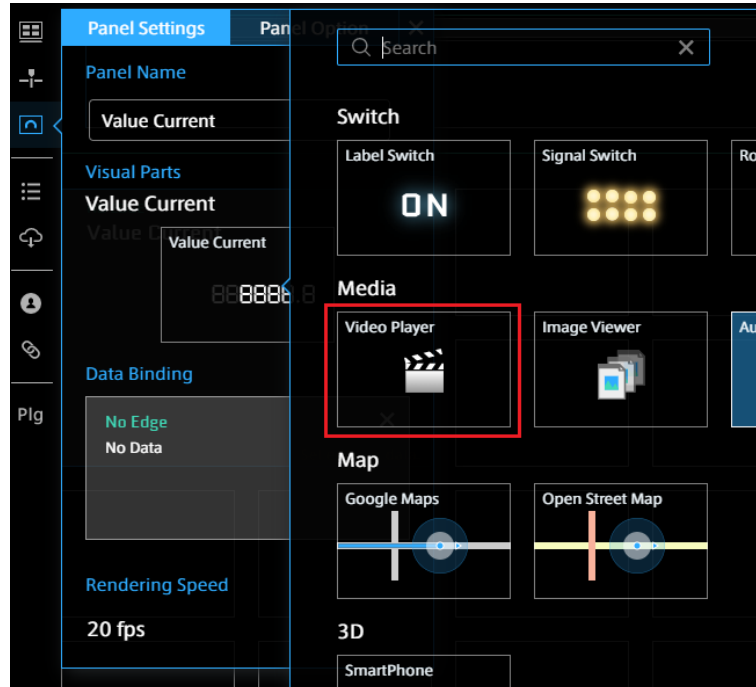


図 21 Video Player

2. Data Visualizer で H.264 用のデータ設定を準備する (p. 45) で作成した設定を使って、送信側エッジからのデータをビジュアルパーツにバインドします。

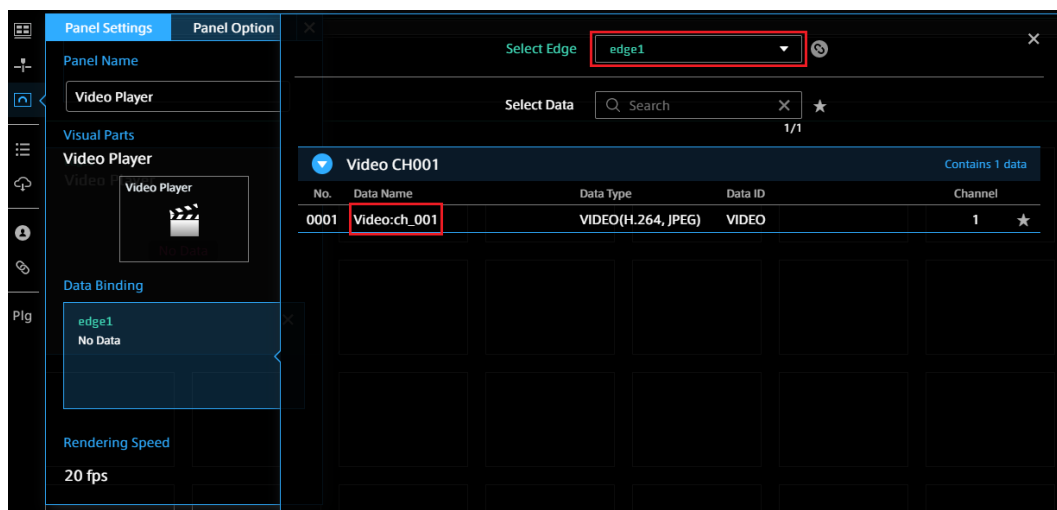


図 22 送信側エッジを選択し、表示したいデータを選択する

3. ライブモードになっていることを確認し (**LIVE** アイコンがピンク色)、▶ をクリックして、表示を開始します。

6.4 メッセージの送信を開始する

ワークスペースで `setup.bash` を実行することによりワークスペースの設定を有効にし、`roslaunch` により画像送信ノードと送信側 `intdash Bridge` を起動します。

```
$ cd ~/catkin_ws  
$ source devel/setup.bash  
$ roslaunch sender.launch
```

別のターミナルで、以下のコマンドを実行して `intdash Edge Agent` を起動します。

```
$ sudo LD_LIBRARY_PATH=/opt/vm2m/lib /opt/vm2m/sbin/intdash-edge-manager -C manager.conf
```

Data Visualizer に、送信側エッジからの H.264 動画が表示できれば成功です。

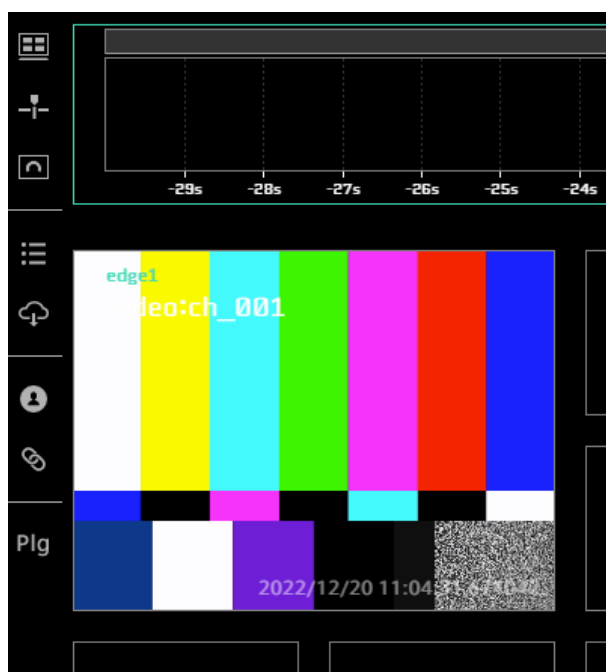


図 23 Video Player での画像の表示

07 制限事項

intdash Bridge には以下の制限事項があります。

7.1 JSON の出力形式について

intdash Bridge v2.0.0 以降の JSON 形式は、それより前のバージョンの JSON 形式との互換性はありません。

7.2 JSON での uint8 型の出力について

uint8 型の出力は、値の範囲が $[-128, 127]$ ではなく、 $[0, 255]$ に変換されます。例えば、 -1 は 255 に変換されます。