intdash ROS2Bridge デベロッパーガイド

intdash ROS2Bridge Version 1.3.0

第 4 版（2024 年 1 月）

aptpod

Rev. 93d654e

目次

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場合があります。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 ROS2Bridge とは

Intdash ROS2Bridge は、エッジデバイスにおいて、ROS2 空間と intdash Edge Agent との間を仲介するソフトウェアです。intdash ROS2Bridge を使用することにより、ROS2 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 ROS2Bridge 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、ROS2 空間では 1 つの ROS2 のノードとして振る舞います。これにより、ROS2 空間から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 ROS2 空間に送信することができます（ダウンストリーム）。

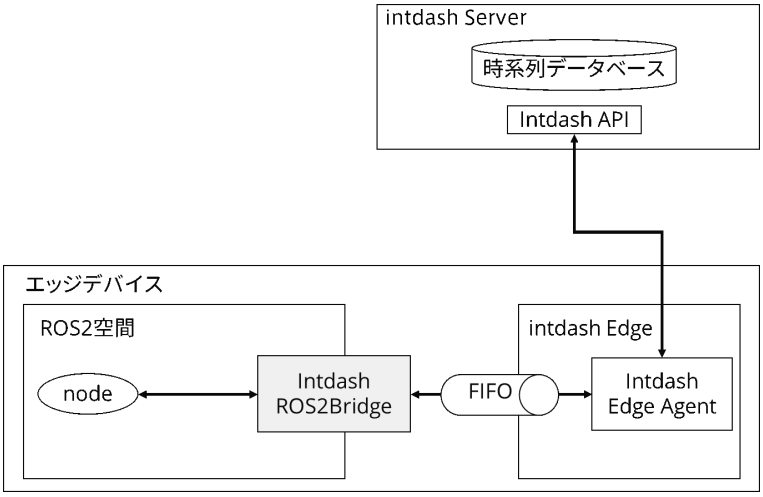


図 1　ROS2 と intdash Edge Agent を仲介する intdash ROS2Bridge

1.2 主要機能

intdash ROS2Bridge は、以下の機能を提供します。

- ROS2 空間内の ROS メッセージを intdash Edge Agent に渡す（ROS2 → intdash）
- intdash Edge Agent から取得したメッセージを ROS2 空間内に流す（intdash → ROS2）

intdash ROS2Bridge が扱うことができる ROS2 メッセージの種類は以下のとおりです。

- トピック
- サービス
  - サービスリクエスト
  - サービスレスポンス
- パラメータ
  - パラメータリクエスト
  - パラメータレスポンス
- アクション
  - アクションゴールリクエスト
  - アクションゴールレスポンス
  - アクションフィードバック
  - アクションリザルト
  - アクションキャンセルリクエスト
  - アクションキャンセルレスポンス

ROS2 空間と intdash の間のデータ交換では以下のフォーマットを使用します。

- CDR(Common Data Representation)
  - ROS2 空間と intdash との間で双方向の送受信が可能
  - 遠隔の ROS2 空間同士を接続する際に使用
- JSON
  - ROS2 空間から intdash への送信のみに対応
  - Data Visualizer でデータを可視化する際に使用
- JPEG
  - ROS2 空間から intdash への送信のみに対応
  - Data Visualizer で画像データを可視化する際に使用
- H264
  - ROS2 空間から intdash への送信のみに対応
  - Data Visualizer で動画データを可視化する際に使用

1.3 動作要件

intdash ROS2Bridge が対応するプラットフォームは以下のとおりです。

- AMD64 上の Ubuntu 20.04 および 22.04
- Arm64 上の Ubuntu 20.04 および 22.04

intdash ROS2Bridge が動作する ROS2 のディストリビューションは以下のとおりです。

- foxy（Ubuntu 20.04 向け）
- humble（Ubuntu 22.04 向け）

intdash ROS2Bridge は以下の DDS(Data Distribution Service) 上で動作することを確認しています。

- Fast-RTPS（rmw_fastrtps_cpp のみに対応しています。rmw_fastrtps_dynamic_cpp はサポートしていません。）

1.4 システム構成

intdash ROS2Bridge を使って ROS2 空間を intdash Edge Agent と接続する場合のシステム構成を以下に挙げます。

> **注釈:** 本書では、intdash サーバーを介して接続される 2 つのデバイスをそれぞれ「エッジデバイス 1」「エッジデバイス 2」と呼びます。

1.4.1 トピックをブリッジする構成

エッジデバイス 1 の ROS2 ノードがパブリッシュしたトピックを、エッジデバイス 2 の ROS2 ノードがサブスクライブする場合の構成は以下のようになります。

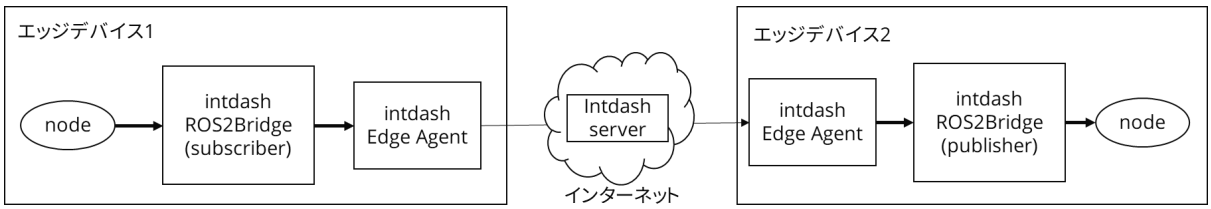


図 2　トピックをブリッジする構成

エッジデバイス 1 において:

1. intdash ROS2Bridge(subscriber) は、ROS2 空間内のトピックをサブスクライブします。
2. intdash ROS2Bridge(subscriber) は、取得したトピックメッセージを intdash データポイントに変換して intdash Edge Agent に渡します。
3. intdash Edge Agent は、データポイントを intdash Server に送信します（アップストリーム）。

エッジデバイス 2 において:

4. intdash Edge Agent は、intdash Server からデータポイントを受信し（ダウンストリーム）、intdash Edge Agent に渡します。
5. intdash ROS2Bridge(publisher) は、データポイントをトピックメッセージに変換して ROS2 空間にパブリッシュします。

1.4.2 サービスをブリッジする構成

エッジデバイス 1 のサービスクライアント（ROS2 ノード）がサービスリクエストを発行し、エッジデバイス 2 のサービスサーバー（ROS2 ノード）がサービスレスポンスを返す場合の構成は以下のようになります。

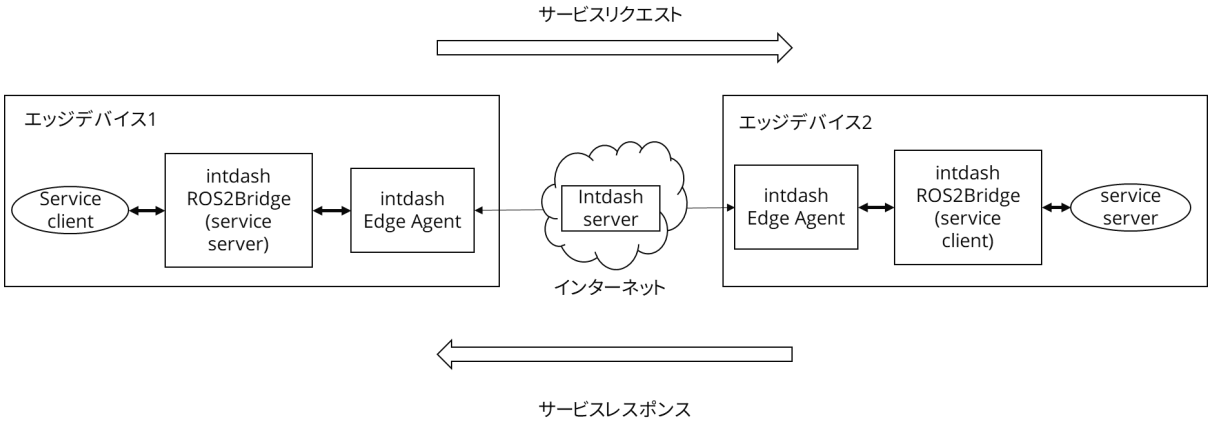


図 3　サービスをブリッジする構成

エッジデバイス 1 において:

1. サービスクライアント（ROS2 ノード）がサービスリクエストを発行します。
2. intdash ROS2Bridge(service server) は、サービスリクエストを受信し、intdash データポイントに変換して intdash Edge Agent に渡します。
3. intdash Edge Agent はデータポイントを intdash Server に送信します（アップストリーム）。

エッジデバイス 2 において:

4. intdash Edge Agent は、intdash Server からデータポイントを受信し（ダウンストリーム）、intdash ROS2Bridge(service client) に渡します。
5. intdash ROS2Bridge(service client) はデータポイントをサービスリクエストに変換してサービスサーバーに送信します。
6. service server はサービスリクエストを処理してサービスレスポンスを発行します。
7. intdash ROS2Bridge(service client) はサービスレスポンスを受信し、intdash のデータポイントに変換して intdash Edge Agent に渡します。
8. intdash Edge Agent はデータポイントを intdash Server に送信します（アップストリーム）。

エッジデバイス 1 において:

9. intdash Edge Agent は、intdash Server からデータポイントを受信し（ダウンストリーム）、intdash

ROS2Bridge(service server) に渡します。

10. intdash ROS2Bridge(service server) はデータポイントをサービスレスポンスに変換して元のサービスクライアントに送信します。

1.4.3 パラメータをブリッジする構成

エッジデバイス 1 のパラメータクライアント（ROS2 ノード）がパラメータリクエストを発行し、エッジデバイス 2 のパラメータサービスサーバー（ROS2 ノード）がパラメータレスポンスを返す場合の構成は以下のようになります。

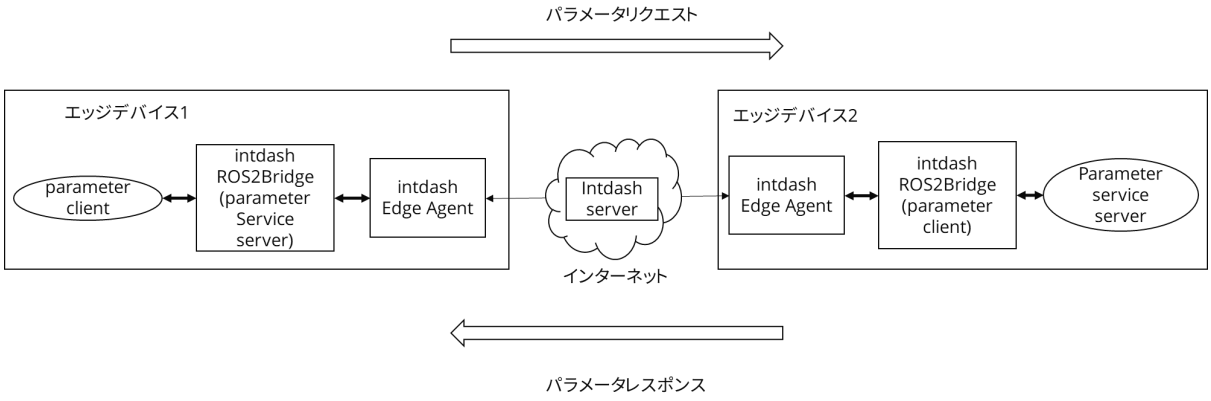


図 4　パラメーターをブリッジする構成

ROS2 では、パラメータリクエストとパラメータレスポンスはサービスリクエストとサービスレスポンスと同じ機構を使用しているので、データの流れは サービスをブリッジする構成 (p. 7) と同じです。

1.4.4 アクションをブリッジする構成

エッジデバイス 1 のアクションクライアント（ROS2 ノード）がアクションリクエストを発行し、エッジデバイス 2 のアクションサーバー（ROS2 ノード）がアクションを実行しレスポンスを返す場合の構成は以下のようになります。

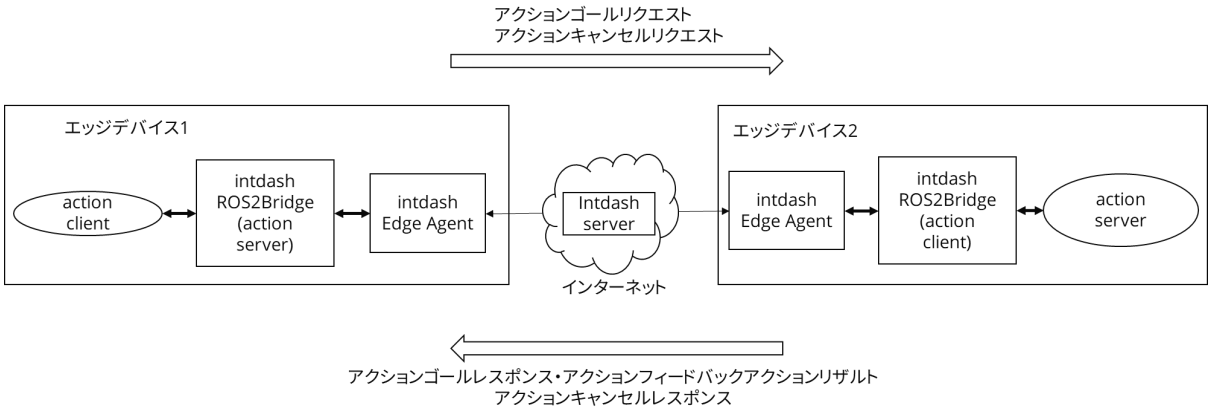


図 5　アクションをブリッジする構成

エッジデバイス 1 において:

1. アクションクライアント（ROS2 ノード）がアクションゴールリクエストを発行します。
2. intdash ROS2Bridge(action server) は、アクションリクエストを受信し、intdash データポイントに変換して intdash Edge Agent に渡します。
3. intdash Edge Agent はデータポイントを intdash Server に送信します（アップストリーム）。

エッジデバイス 2 において:

4. intdash Edge Agent は、intdash Server からデータポイントを受信し（ダウンストリーム）、intdash ROS2Bridge(action client) に渡します。
5. intdash ROS2Bridge(action client) はデータポイントをアクションゴールリクエストに変換してアクションサーバーに送信します。
6. action server はアクションゴールリクエストを処理してアクションゴールレスポンスを発行します。
7. intdash ROS2Bridge(action client) はアクションゴールレスポンスを受信し、intdash のデータポイントに変換して intdash Edge Agent に渡します。
8. intdash Edge Agent はデータポイントを intdash Server に送信します（アップストリーム）。

エッジデバイス 1 において:

9. intdash Edge Agent は、intdash Server からデータポイントを受信し（ダウンストリーム）、intdash ROS2Bridge(action server) に渡します。
10. intdash ROS2Bridge(action server) はデータポイントをアクションゴールレスポンスに変換して、元のアクションクライアントに送信します。

アクションキャンセルリクエストは上記のアクションゴールリクエストと同じ流れでブリッジされます。

また、アクションフィードバック、アクションゴールリザルト、アクションキャンセルレスポンスは、アクションゴールレスポンスと同じ流れでブリッジされます。

02 インストールと起動

2.1 intdash Edge Agent と intdash ROS2Bridge をインストールする

intdash ROS2Bridge を使用するには、intdash Edge Agent と intdash ROS2Bridge をエッジデバイスインストールする必要があります。

2.1.1 intdash Edge Agent をインストールする

intdash Edge Agent 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。

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

DISTRIBUTION	ARCHITECTURE
ubuntu	amd64, 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. 以下のコマンドを実行して intdash Edge Agent をインストールします。

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

2.1.2 intdash ROS2Bridge 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 ROS2Bridge をインストールしますが、Docker を使って intdash ROS2Bridge 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 ROS2Bridge 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. 10) 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
   ```

2. 以下のコマンドを実行して intdash ROS2Bridge をインストールします。

   ```
   $ sudo apt-get install ros-[foxy,humble]-intdash-ros2bridge
   ```

   intdash ROS2Bridge は以下のパスにインストールされます。

   ```
   /opt/ros/[foxy,humble]/share/intdash_ros2bridge
   ```

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
   ```

2.2 intdash Edge Agent の設定を行う（manager.conf）

2.2.1 デバイスコネクターの設定

intdash Edge Agent に、intdash ROS2Bridge と接続するための設定を追加します。

注釈:

- 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。
- manager.conf 全体の例は、付録：intdash Edge Agent 用 manager.conf のサンプル (p. 73) を参照してください。

intdash Edge Agent の設定ファイル manager.conf 内で、`loggers`（デバイスコネクター）として、`int-`

dash_bridge を追加します。

```
"loggers": [
    {
        "devicetype": "intdash_ros2bridge",
        "path": "",
        "connections": [
            {
                "fifo_tx": "/var/run/intdash/logger_001.tx", # (1)
                "fifo_rx": "/var/run/intdash/logger_001.rx", # (2)
                "channel": 1 # (3)
            }, # (X)
            {
                "fifo_tx": "/var/run/intdash/logger_002.tx", # (1)
                "channel": 2 # (3)
            } # (Y)
        ],
        "details": {
            "plugin": "fifo"
        }
    },
...
]
```

上記サンプルでは、intdash ROS2Bridge との間で双方向の送受信を行うチャンネル 1（X）と、intdash ROS2Bridge 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 CDR 形式で intdash サーバーとの送受信を行い、チャンネル 2 では JSON 形式で intdash サーバーへの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 intdashROS2Bridge 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 ROS2Bridge の設定を行う (p. 16) にも設定します。
(2)	loggers[].connections[].fifo_rx	intdash Edge Agent が intdashROS2Bridge 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 ROS2Bridge の設定を行う (p. 16) 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。

2.2.2 ダウンストリームの設定

以下に該当する場合は、manager.conf でダウンストリームの設定も行う必要があります。

- トピックメッセージのブリッジをする場合の、受信（サブスクライブ）側エッジデバイス
- サービス、アクション、またはパラメータのブリッジをする場合の、送信側と受信側エッジデバイス（レスポンスの処理が発生するため、双方向の伝送が発生します）

ここでは、intdash ROS2Bridge に関わる部分のみを説明します。

intdash Edge Agent におけるダウンストリーム用のモジュールである「control クライアント」の詳細は、intdash Edge Agent デベロッパーガイド を参照してください。

> **注釈:** 以下の例では、受信するデータのデータ ID を `ctlr_flt_ids` で指定します。データ ID については、ROS2 メッセージの種類と intdash データ ID の対応関係 (p. 66) を参照してください。

トピックをブリッジする場合のエッジデバイス 2（受信側）

エッジデバイス 1 でパブリッシュされたトピックをエッジデバイス 2 でサブスクライブする場合は、以下のようにします。

エッジデバイス 2 の manager.conf の設定:

- `ctlr_id` (受信するデータの送信元の指定）にエッジデバイス 1 の UUID を設定します。
- `ctlr_flt_ids` (受信するデータの ID の指定）に、`"/topic:/<topic_name>"` を追加します。

以下は設定例です。

```
"ctlr_id": "エッジデバイス 1 の UUID",
"ctlr_flt_ids": [
  "/topic:/chatter",
]
```

サービスをブリッジする場合のエッジデバイス 1 とエッジデバイス 2

エッジデバイス 1 のサービスクライアントが発行したサービスリクエストをエッジデバイス 2 のサービスサーバーが処理する場合は、以下のようにします。

エッジデバイス 1 の manager.conf の設定:

- `ctlr_id` (受信するデータの送信元の指定）にエッジデバイス 2 の UUID を設定します
- `ctlr_flt_ids` (受信するデータの ID の指定）に `"/srv/resp:<service_name>"` を追加します。

以下は設定例です。

```
"ctlr_id": "エッジデバイス 2 の UUID",
"ctlr_flt_ids": [
  "/srv/resp:/add_two_ints",
]
```

エッジデバイス 2 の設定:

- ctlr_id（受信するデータの送信元の指定）にエッジデバイス 1 の UUID を設定します
- ctlr_flt_ids（受信するデータの ID の指定）に "/srv/req:<service_name>" を追加します。

以下は設定例です。

```
"ctlr_id": "エッジデバイス 1 の UUID",
"ctlr_flt_ids": [
  "/srv/req:/add_two_ints",
]
```

アクションをブリッジする場合のエッジデバイス 1 とエッジデバイス 2

エッジデバイス 1 のアクションクライアント（ROS2 ノード）のアクションリクエストをエッジデバイス 2 のアクションサーバー（ROS2 ノード）が処理しアクションを実行する場合は、以下のようにします。

エッジデバイス 1 の設定:

- ctlr_id（受信するデータの送信元の指定）にエッジデバイス 2 の UUID を設定します
- ctlr_flt_ids（受信するデータの ID の指定）に以下を追加します。
  - アクションゴールレスポンス "/act/goal_resp:<action_name>"
  - アクションフィードバック "/act/fb:<action_name>"
  - アクションゴールリザルト "/act/result:<action_name>"
  - アクションキャンセルレスポンス "/act/cancel_resp:<action_name>"

以下は設定例です。

```
"ctlr_id": "エッジデバイス 2 の UUID",
"ctlr_flt_ids": [
  "/act/goal_resp:/fibonacci",
  "/act/fb:/fibonacci",
  "/act/result:/fibonacci",
  "/act/cancel_resp:/fibonacci",
]
```

エッジデバイス 2 の設定:

- ctlr_id（受信するデータの送信元の指定）にエッジデバイス 1 の UUID を設定します。
- ctlr_flt_ids（受信するデータの ID の指定）に以下を追加します。
  - アクションゴールリクエスト "/act/goal_req:<action_name>"
  - アクションキャンセルリクエスト "/act/cancel_req:<action_name>"

以下は設定例です。

```
"ctlr_id": "エッジデバイス 1 の UUID",
"ctlr_flt_ids": [
  "/act/goal_req:/fibonacci",
  "/act/cancel_req:/fibonacci",
]
```

パラメータをブリッジする場合のエッジデバイス 1 とエッジデバイス 2

エッジデバイス 1 のパラメータクライアント（ROS2 ノード）のパラメータリクエストをエッジデバイス 2 のパラメータサービスサーバ（ROS2 ノード）が処理しパラメータレスポンスを返す場合は、以下のようにします。

エッジデバイス 1 の設定:

- `ctlr_id`（受信するデータの送信元の指定）にエッジデバイス 2 の UUID を設定します
- `ctlr_flt_ids`（受信するデータの ID の指定）に、パラメータレスポンスを受信するための設定を追加します。
  - `"/param/resp:<node_name>/get_parameters"`
  - `"/param/resp:<node_name>/describe_parameters"`
  - `"/param/resp:<node_name>/get_parameter_types"`
  - `"/param/resp:<node_name>/get_parameters"`
  - `"/param/resp:<node_name>/list_parameters"`
  - `"/param/resp:<node_name>/set_parameters"`
  - `"/param/resp:<node_name>/set_parameters_atomically"`

以下は設定例です。

```
"ctlr_id": "エッジデバイス 2 の UUID",
"ctlr_flt_ids": [
  "/param/resp:/minimal_action_server/get_parameters",
  "/param/resp:/minimal_action_server/describe_parameters",
  "/param/resp:/minimal_action_server/get_parameter_types",
  "/param/resp:/minimal_action_server/get_parameters",
  "/param/resp:/minimal_action_server/list_parameters",
  "/param/resp:/minimal_action_server/set_parameters",
  "/param/resp:/minimal_action_server/set_parameters_atomically"
]
```

エッジデバイス 2 の設定:

- `ctlr_id`（受信するデータの送信元の指定）にエッジデバイス 2 の UUID を設定します。
- `ctlr_flt_ids`（受信するデータの ID の指定）に、パラメータリクエストを受信するための設定を追加します。
  - `"/param/req:<node_name>/get_parameters"`
  - `"/param/req:<node_name>/get_parameters"`
  - `"/param/req:<node_name>/describe_parameters"`
  - `"/param/req:<node_name>/get_parameter_types"`
  - `"/param/req:<node_name>/get_parameters"`
  - `"/param/req:<node_name>/list_parameters"`
  - `"/param/req:<node_name>/set_parameters"`
  - `"/param/req:<node_name>/set_parameters_atomically`

以下は設定例です。

```
"ctlr_id": "エッジデバイス 1 の UUID",
"ctlr_flt_ids": [
  "/param/req:/minimal_action_server/get_parameters",
  "/param/req:/minimal_action_server/get_parameters",
  "/param/req:/minimal_action_server/describe_parameters",
  "/param/req:/minimal_action_server/get_parameter_types",
  "/param/req:/minimal_action_server/get_parameters",
  "/param/req:/minimal_action_server/list_parameters",
  "/param/req:/minimal_action_server/set_parameters",
  "/param/req:/minimal_action_server/set_parameters_atomically"
]
```

注意: topic_name 、service_name 、action_name 、node_name の先頭に / が含まれている場合は、manager.conf の ctlr_flt_ids にも / を入力してください。

2.3 intdash ROS2Bridge の設定を行う

intdash ROS2Bridge の設定は、yaml 形式の設定ファイルで行います。設定項目の詳細については、intdash ROS2Bridge の設定 (p. 37) を参照してください。

2.3.1 エッジデバイス 1 の設定

エッジデバイス 1 では以下のような設定ファイルを用意します。以下の設定ファイルは後述するサンプルで使用します。

```yaml
num_callback_threads: 15

upstream:
  enabled: true
  formats:
  - format: "cdr"
    writer:
      path: "/var/run/intdash/logger_001.tx"
      buffering: true
  - format: "json"
    writer:
      path: "/var/run/intdash/logger_002.tx"
      max_array_size: 100
downstream:
  enabled: true
  format: "cdr"
  reader:
    path: "/var/run/intdash/logger_001.rx"

action_servers:
```

(次のページに続く)

(前のページからの続き)

```
  enabled: true
  actions:
  - action_name: "/fibonacci"
    action_type: "example_interfaces/action/Fibonacci"
    format:
      - "cdr"
      - "json"

parameter_service_servers:
  enabled: true
  nodes:
  - node_name: "/minimal_action_server"
    format:
    - "cdr"
    - "json"

service_servers:
  enabled: true
  services:
  - service_name: "/add_two_ints"
    service_type: "example_interfaces/srv/AddTwoInts"
    format:
      - "cdr"
      - "json"

subscribers:
  enabled: true
  topics:
  - topic_name: "/chatter"
    format:
    - "cdr"
    - "json"
```

2.3.2 エッジデバイス 2 の設定

エッジデバイス 2 では以下のような設定ファイルを用意します。以下の設定ファイルは後述するサンプルで使用します。

> **注釈:** Data Visualizer を使った可視化のみを行う場合は、エッジデバイス 2 の準備は不要です。

```
num_callback_threads: 15

upstream:
  enabled: true
  formats:
  - format: "cdr"
```

(次のページに続く)

(前のページからの続き)

```
    writer:
      path: "/var/run/intdash/logger_001.tx"
      buffering: true
  - format: "json"
    writer:
      path: "/var/run/intdash/logger_002.tx"
      max_array_size: 100
downstream:
  enabled: true
  format: "cdr"
  reader:
    path: "/var/run/intdash/logger_001.rx"

action_clients:
  enabled: true
  actions:
  - action_name: "/fibonacci"
    action_type: "example_interfaces/action/Fibonacci"
    format:
      - "cdr"

parameter_clients:
  enabled: true
  nodes:
  - node_name: "/minimal_action_server"
    format:
    - "cdr"
    - "json"

service_clients:
  enabled: true
  response:
    resend_duration: "10sec"
    resend_interval: "1sec"
  services:
  - service_name: "/add_two_ints"
    service_type: "example_interfaces/srv/AddTwoInts"
    format:
      - "cdr"
      - "json"

publishers:
  enabled: true
```

2.4 intdash ROS2Bridge と intdash Edge Agent を起動する

エッジデバイス 1（および必要な場合はエッジデバイス 2）で、intdash ROS2Bridge と intdash Edge Agent を起動します。

1. intdash Edge Agent を起動します。

```
$ sudo \
LD_LIBRARY_PATH=/opt/vm2m/lib \
INTDASH_EDGE_UUID=エッジデバイス 1 の UUID \
INTDASH_EDGE_SECRET=エッジデバイス 1 の シークレット \
INTDASH_EDGE_SERVER=dummy.intdash.jp \
INTDASH_EDGE_APPDIR=/var/lib \
INTDASH_EDGE_RUNDIR=/var/run \
INTDASH_EDGE_BINDIR=/opt/vm2m/bin \
INTDASH_EDGE_SBINDIR=/opt/vm2m/sbin \
INTDASH_EDGE_LIBDIR=/opt/vm2m/lib \
INTDASH_EDGE_CONFDIR=/etc/opt/intdash \
/opt/vm2m/sbin/intdash-edge-manager -C manager.conf
```

2. 以下のいずれかの方法で intdash ROS2Bridge を起動します。起動時にネームスペースを指定することもできます。

   - ros2 run で起動する場合、以下のコマンドを実行します。
     - ネームスペースを指定しない場合

```
$ ros2 run intdash_ros2bridge intdash_ros2bridge --ros-args -p config_path:=path_to_config_file
```

     - ネームスペースを指定する場合

```
$ ros2 run intdash_ros2bridge intdash_ros2bridge --ros-args -p config_path:=path_to_config_
↪file -r  __ns:=/namespace
```

   - 事前に用意されている launch ファイルを使う場合は以下のコマンドを実行します。
     - ネームスペースを指定しない場合

```
$ ros2 launch intdash_ros2bridge intdash_ros2bridge.launch.py config_path:=path_to_config_file
```

     - ネームスペースを指定する場合

```
$  ros2 launch intdash_ros2bridge intdash_ros2bridge.launch.py config_path:=path_to_config_
↪file namespace:=/namespace
```

   - コンポーネントを使用する場合、まずコンポーネントを起動します。

```
$ ros2 run rclcpp_components component_container
```

     次に、intdash ROS2Bridge をロードします。
     - ネームスペースを指定しない場合

```
$ ros2 component load /ComponentManager intdash_ros2bridge -p config_path:=/ws/src/intdash_
↪ros2bridge/test/system_test/action/config/cdr/action_server.yaml intdash_
↪ros2bridge::IntdashRos2Bridge
```

- ネームスペースを指定する場合

```
$ ros2 component load /ComponentManager intdash_ros2bridge -p config_path:=path_to_config_file␣
↪-r __ns:=/namespace intdash_ros2bridge::IntdashRos2Bridge
```

- ユーザー自身が作成した launch ファイルから intdash_ros2bridge の launch ファイルを呼び出す場合は場合は以下のように記述します。
  - ネームスペースを指定しない場合

```
from launch import LaunchDescription
from launch.actions import IncludeLaunchDescription
from launch.actions import LogInfo
from launch.launch_description_sources import PythonLaunchDescriptionSource
from launch.substitutions import ThisLaunchFileDir

def generate_launch_description():
    return LaunchDescription([
        IncludeLaunchDescription(
            PythonLaunchDescriptionSource(
                [ThisLaunchFileDir(), '/intdash_ros2bridge.launch.py']
            ),
            launch_arguments=
                {'config_path': '/ws/src/intdash_ros2bridge/config/sample1.yaml'}.items()
        ),
    ])
```

  - ネームスペースを指定する場合

```
from launch import LaunchDescription
from launch.actions import IncludeLaunchDescription
from launch.actions import LogInfo
from launch.launch_description_sources import PythonLaunchDescriptionSource
from launch.substitutions import ThisLaunchFileDir

def generate_launch_description():
    return LaunchDescription([
        IncludeLaunchDescription(
            PythonLaunchDescriptionSource(
                [ThisLaunchFileDir(), '/intdash_ros2bridge.launch.py']
            ),
            launch_arguments=
                {'config_path': '/ws/src/intdash_ros2bridge/config/sample1.yaml',
                 'namespace':'/namespace'}.items()
        ),
    ])
```

intdash サーバーを仲介したデータ送受信が開始されます。

03 サンプルの実行

本章では、ROS2 公式リポジトリから入手したサンプルプログラムを使って、intdash ROS2Bridge の機能を実際に使用します。画像と動画のサンプルについては、入力データの生成に利用しているソフトウェアの都合で、Foxy 環境でのみ実行可能となっています。

3.1 サンプルを実行するためにエッジデバイスをセットアップする

1. エッジデバイス 1 とエッジデバイス 2 で以下のコマンドを実行し、公式の ROS2 Foxy または Humble 用のサンプルプログラムをインストールします。

   注釈: Data Visualizer を使った可視化のみを行う場合は、エッジデバイス 2 の準備は不要です。

   ```
   $ sudo apt install ros-[foxy,humble]-examples-*
   ```

2. エッジデバイス 1 とエッジデバイス 2 で、intdash ROS2Bridge を起動するターミナルで以下のコマンドを実行して環境をセットアップします。

   ```
   $ source /opt/ros/[foxy,humble]/setup.bash
   ```

3.2 トピックを伝送する

1. エッジデバイス 1 上とエッジデバイス 2 上で intdash ROS2Bridge と intdash Edge Agent を起動します。
2. エッジデバイス 1 で以下のコマンドを実行します。

   ```
   $ ros2 topic pub /chatter std_msgs/String "data: Hello world"
   ```

3. エッジデバイス 2 で以下のコマンドを実行します。

   ```
   $ ros2 topic echo /chatter
   ```

   エッジデバイス 1 でパブリッシュされたトピックが intdash 経由でメッセージが伝送され、エッジデバイス 2 に出力されます。

   ```
   # ros2 topic echo /chatter
   data: Hello world
   ---
   data: Hello world
   ---
   （省略）
   ```

3.3 サービスを伝送する

1. エッジデバイス 1 上とエッジデバイス 2 上で intdash ROS2Bridge と intdash Edge Agent を起動します。
2. エッジデバイス 2 で以下のコマンドを実行しサービスを起動します。

```
$ ros2 run examples_rclcpp_minimal_service service_main
```

3. エッジデバイス 1 で以下のコマンドを実行しサービスクライアントを起動します。

```
$ ros2 run examples_rclcpp_minimal_client client_main
```

エッジデバイス 1 では以下のような出力が表示されます。

```
$ ros2 run examples_rclcpp_minimal_service service_main
[INFO] [1634879875.987806930] [minimal_service]: request: 41 + 1
```

intdash 経由でメッセージが伝送され、エッジデバイス 2 に以下のような出力が表示されます。

```
$ ros2 run examples_rclcpp_minimal_client client_main
[INFO] [1634879881.148063426] [minimal_client]: result of 41 + 1 = 42
```

3.4 アクションを伝送する

1. エッジデバイス 1 上とエッジデバイス 2 上で intdash ROS2Bridge と intdash Edge Agent を起動します。
2. エッジデバイス 2 で以下のコマンドを実行します

```
$ ros2 run examples_rclcpp_minimal_action_server action_server_member_functions
```

3. エッジデバイス 1 で以下のコマンドを実行します

```
$ ros2 run examples_rclcpp_minimal_action_client action_client_member_functions
```

intdash 経由でメッセージが伝送され、エッジデバイス 2 に以下のような出力が表示されます。

```
$ ros2 run examples_rclcpp_minimal_action_client action_client_member_functions
[INFO] [1634881800.018154229] [minimal_action_client]: Sending goal
[INFO] [1634881800.020415652] [minimal_action_client]: Goal accepted by server, waiting for result
[INFO] [1634881801.019846782] [minimal_action_client]: Next number in sequence received: 2
[INFO] [1634881802.020532553] [minimal_action_client]: Next number in sequence received: 3
[INFO] [1634881803.019839647] [minimal_action_client]: Next number in sequence received: 5
[INFO] [1634881804.019882402] [minimal_action_client]: Next number in sequence received: 8
[INFO] [1634881805.019849450] [minimal_action_client]: Next number in sequence received: 13
[INFO] [1634881806.020189671] [minimal_action_client]: Next number in sequence received: 21
[INFO] [1634881807.020374203] [minimal_action_client]: Next number in sequence received: 34
[INFO] [1634881808.020415252] [minimal_action_client]: Next number in sequence received: 55
[INFO] [1634881809.020678136] [minimal_action_client]: Result received
```

(次のページに続く)

(前のページからの続き)

```
[INFO] [1634881809.020779122] [minimal_action_client]: 0
[INFO] [1634881809.021047253] [minimal_action_client]: 1
[INFO] [1634881809.021416536] [minimal_action_client]: 1
[INFO] [1634881809.021530100] [minimal_action_client]: 2
[INFO] [1634881809.021594926] [minimal_action_client]: 3
[INFO] [1634881809.021844648] [minimal_action_client]: 5
[INFO] [1634881809.021910571] [minimal_action_client]: 8
[INFO] [1634881809.022269954] [minimal_action_client]: 13
[INFO] [1634881809.022328569] [minimal_action_client]: 21
[INFO] [1634881809.022367620] [minimal_action_client]: 34
[INFO] [1634881809.022578242] [minimal_action_client]: 55
```

3.5 パラメータを伝送する

1. エッジデバイス 1 上とエッジデバイス 2 上で intdash ROS2Bridge と intdash Edge Agent を起動します。
2. エッジデバイス 2 で以下のコマンドを実行します

```
$ ros2 run examples_rclcpp_minimal_action_server action_server_member_functions
```

3. エッジデバイス 1 で以下のコマンドを実行します。

```
$ ros2 service call /minimal_action_server/get_parameters rcl_interfaces/srv/GetParameters
```

intdash 経由でメッセージが伝送され、エッジデバイス 1 に以下のような出力が表示されます。

```
waiting for service to become available...
requester: making request: rcl_interfaces.srv.GetParameters_Request(names=[])

response:
rcl_interfaces.srv.GetParameters_Response(values=[])
```

3.6 Data Visualizer でメッセージを可視化する

ROS2 メッセージを Data Visualizer で可視化するためには、以下を行います。

- Data Visualizer で、JSON データをパースしてメッセージを取り出す設定を行う
- intdash ROS2Bridge でメッセージを JSON 表現に変換したうえで、intdash Edge Agent から intdash サーバーに送信する

まず、Data Visualizer で JSON データをパースする設定を行います。

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。

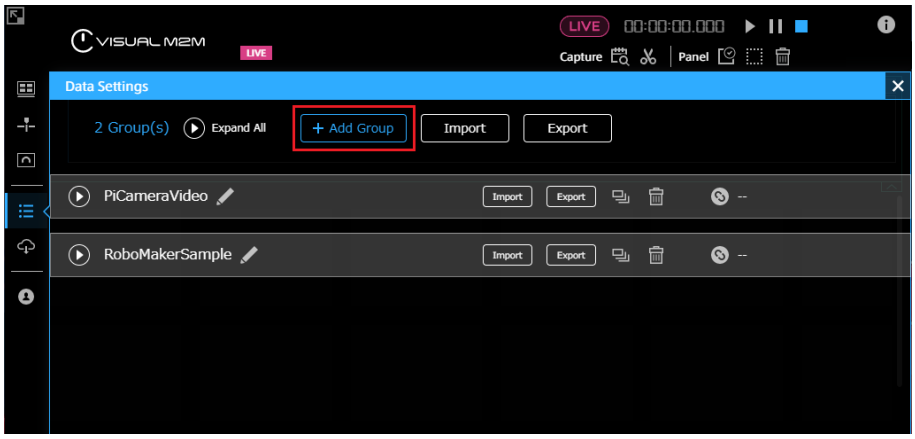


図 6　Group を追加

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。

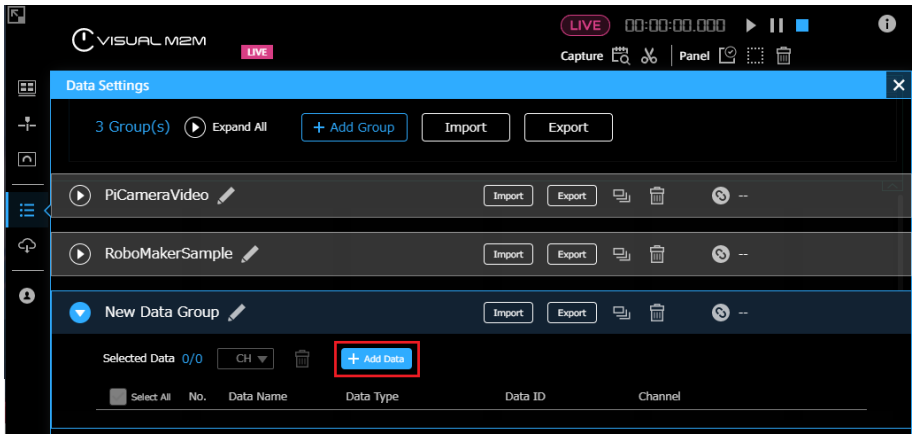


図 7　Data を追加

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

> **注釈:** intdash における、ROS2 メッセージの JSON 表現については ROS2 メッセージの JSON 表現 (p. 67) を確認ください。

- Data Name: 分かりやすい任意の名前
- Target Data:
  - Data Type: String
  - Data ID: `/topic:/chatter`
  - Channel: 2
- Conversion Settings:
  - Conversion Type: As JSON
  - Field Path: `msg.data`
  - Value Type: String

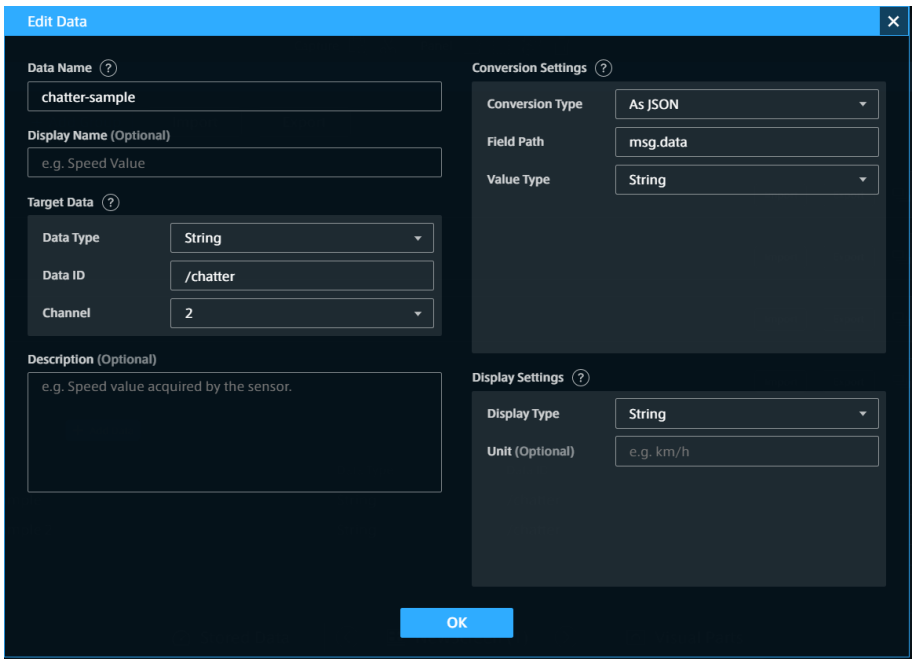


図 8　JSON データ {"msg":{"data":"Hello world"}} をパースして値を取り出す設定

[OK] をクリックし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。

次に、Data Visualizer でビジュアルパーツを配置します。

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。

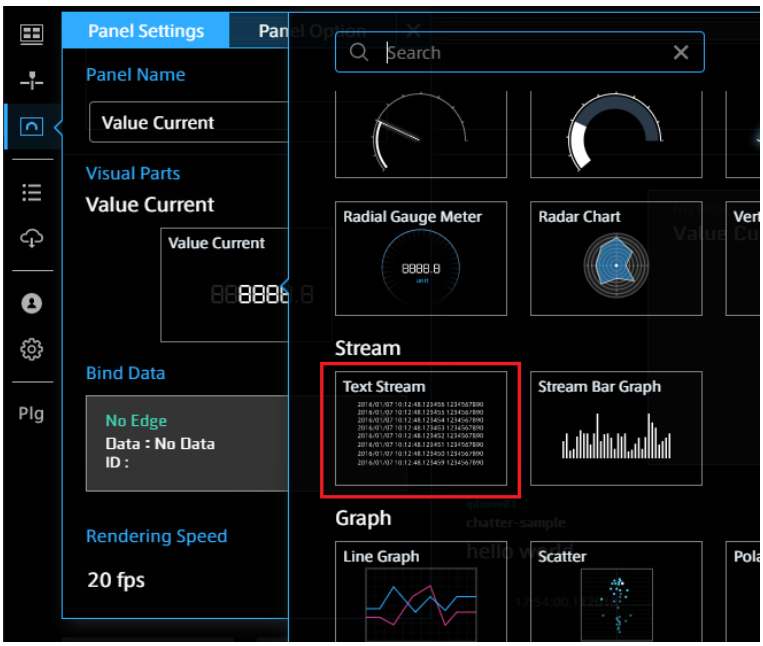


図 9　Text Stream を選択

2. 先ほど作成したデータ設定を使って、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。

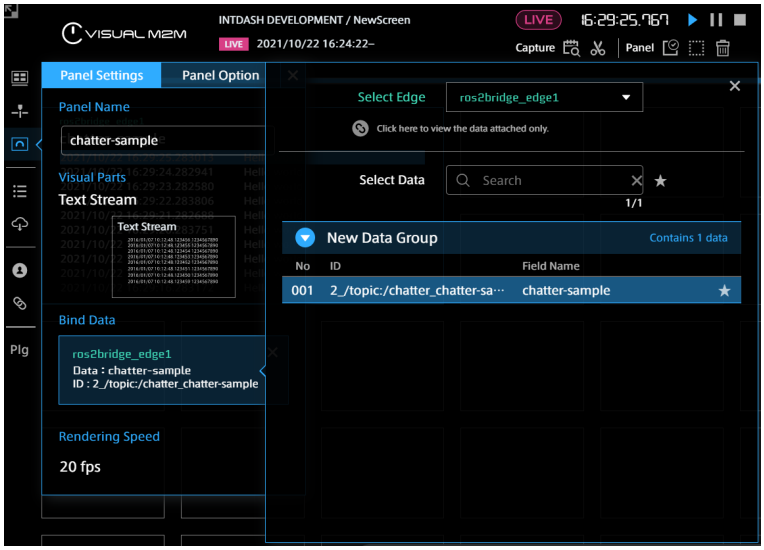


図 10　データをバインド

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. エッジデバイス 1 で intdash ROS2Bridge と intdash Edge Agent を起動します。

3. エッジデバイス 1 で以下のコマンドを実行します。

```
$ ros2 topic pub /chatter std_msgs/String "data: Hello world"
```

Data Visualizer に、エッジデバイス 1 からのメッセージが表示されれば成功です。

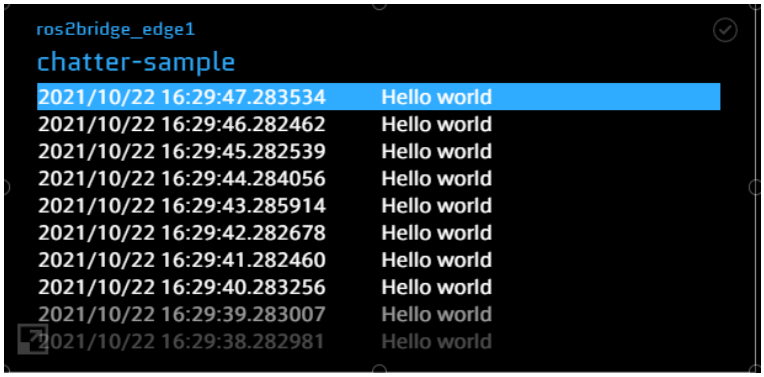


図 11　メッセージの表示

3.7 Data Visualizer で画像を可視化する（Foxy 向け）

画像の ROS2 メッセージを Data Visualizer で JPEG として可視化するためには、以下を行います。

- Data Visualizer で、JPEG データを表示するための設定を行う
- intdash ROS2Bridge で画像を JPEG に変換したうえで、intdash Edge Agent から intdash サーバーに送信する

準備として、ROS2 空間で画像を送信するノードを準備します。

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 ~/ws/src
```

2. ワークスペース内に、gscam2 (ROS2 camera driver for GStreamer-based video streams) をダウンロードします。

```
$ cd ~/ws
$ cd src/
$ git clone https://github.com/clydemcqueen/gscam2.git -b foxy
$ git clone https://github.com/ptrmu/ros2_shared.git
```

3. ビルドを実行します。

```
$ cd ~/ws/
$ sudo rosdep install --from-paths src --ignore-src -y
$ colcon build
```

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. 画像送信ノード用パラメーター設定ファイル (`/opt/vm2m/etc/params_image.yaml`) を以下のように作成します。

   params_image.yaml

```yaml
gscam_publisher:
   ros__parameters:
      gscam_config: 'videotestsrc pattern=smtpe75 ! video/x-raw,format=YUY2,width=320,height=240,
↪framerate=1/1 ! videoconvert'
      preroll: False
      use_gst_timestamps: False
      camera_name: 'my_camera'
      frame_id: 'my_camera_frame'
```

Data Visualizer で JPEG を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。

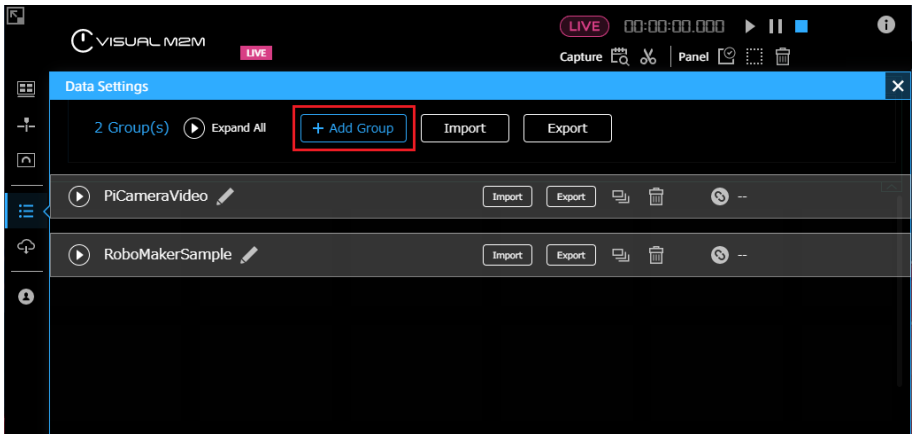


図 12　Group を追加

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。

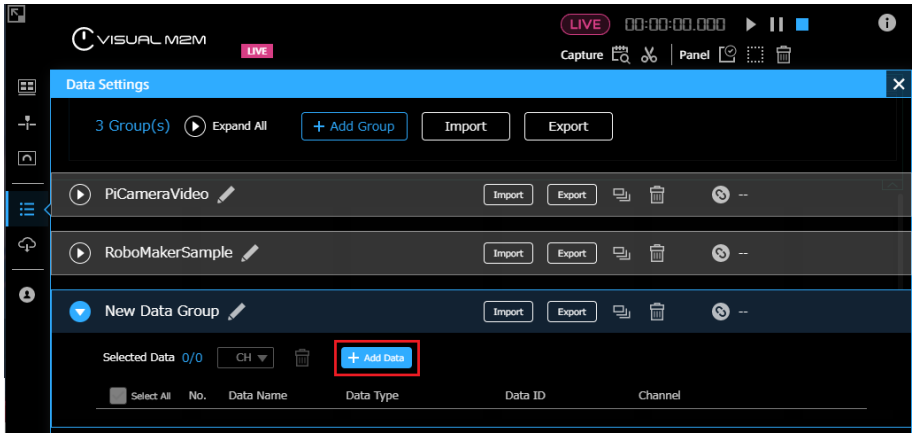
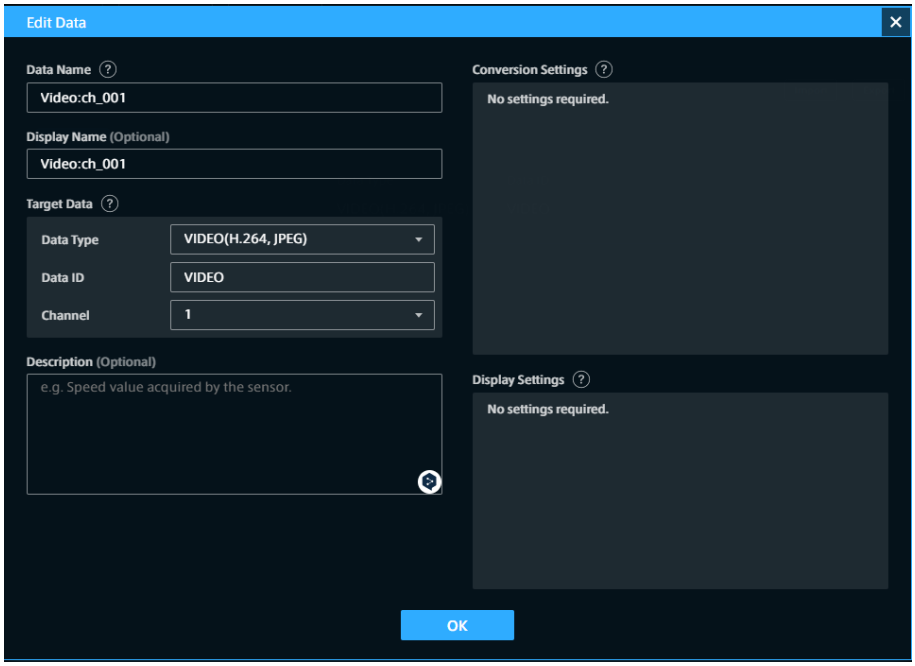


図 13　Data を追加

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



[OK] をクリックし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。

次に、DataVisualizer でビジュアルパーツを配置します。

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

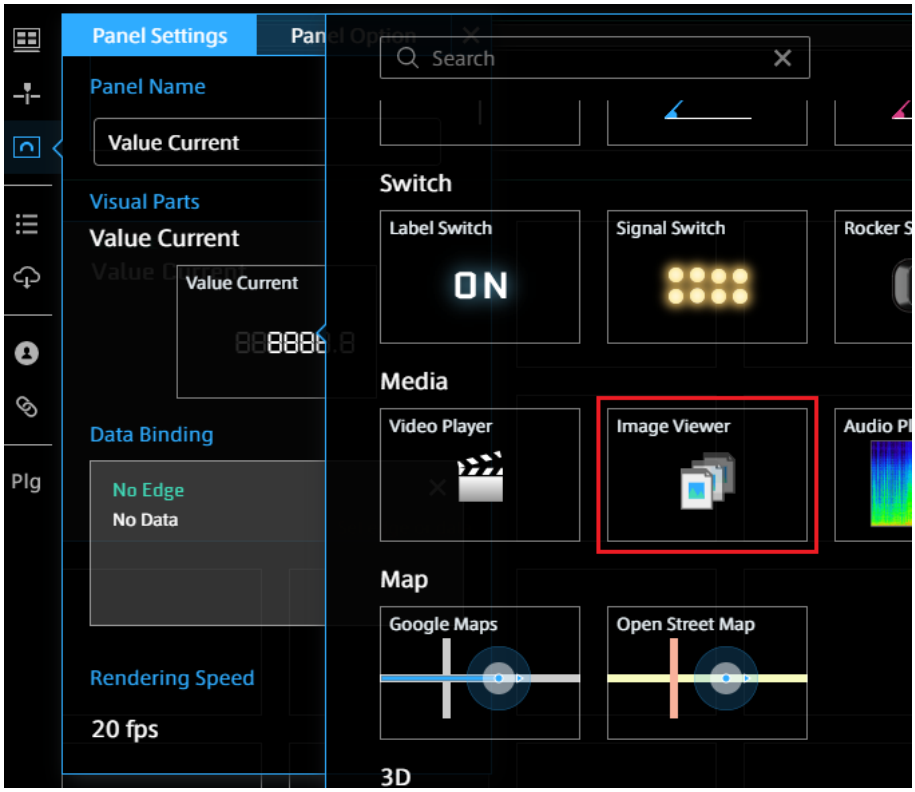


図 14　Image Viewer を選択

2. 先ほど作成したデータ設定を使って、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。

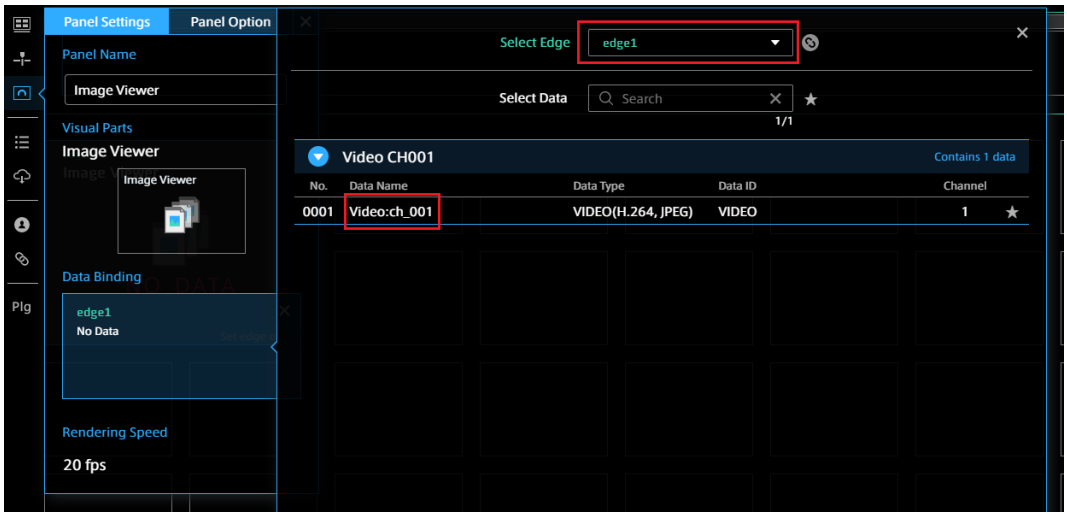


図 15　データをバインド

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. エッジデバイス 1 の intdash ROS2Bridge 設定ファイルを以下のように作成します。

   params.yaml

```
upstream:
   enabled: true
   formats:
   - format: "jpeg_1"
      writer:
         path: "/opt/vm2m/var/run/intdash/logger_001.tx"
         buffering: true

subscribers:
   enabled: true
   topics:
   - topic_name: "/sensor_msgs/image_raw"
      format:
      - "jpeg_1"
      pipeline: "jpegenc ! appsink name=appsink emit-signals=true"
```

5. エッジデバイス 1 で intdash ROS2Bridge と intdash Edge Agent を起動します。

6. エッジデバイス 1 で以下のコマンドを実行します。

```
$ source ~/ws/install/setup.bash
$ ros2 run gscam2 gscam_main --ros-args --remap /image_raw:=/sensor_msgs/image_raw --params-file /opt/
↪vm2m/etc/params_image.yaml
```

   Data Visualizer に、エッジデバイス 1 からの画像が表示されれば成功です。

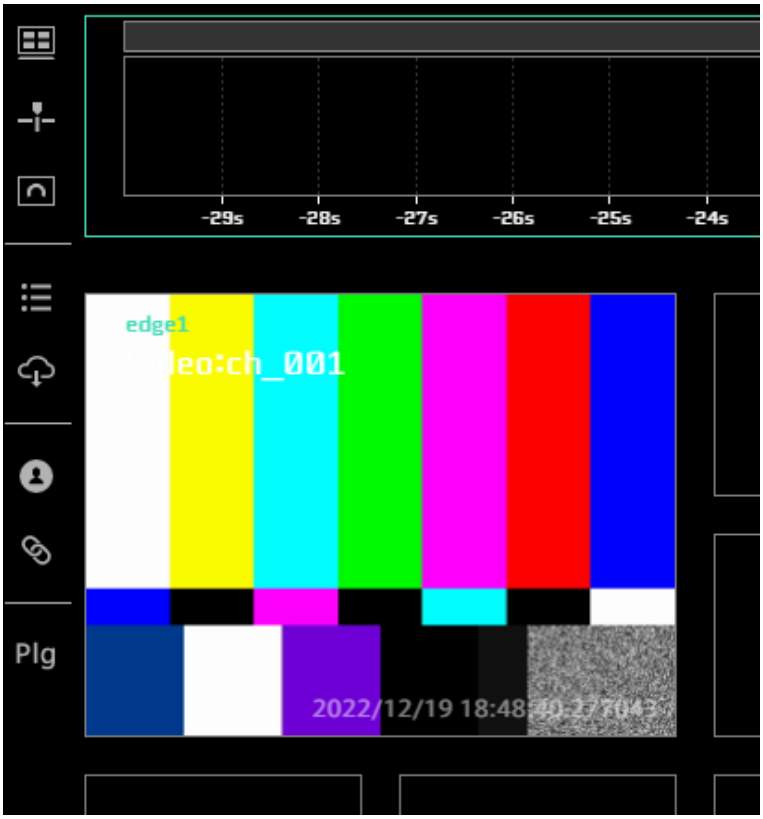


図 16　Image Viewer での画像の表示

3.8 Data Visualizer で動画を可視化する（Foxy 向け）

動画の ROS2 メッセージを Data Visualizer で H.264 として可視化するためには、以下を行います。

- Data Visualizer で、H.264 データを表示するための設定を行う
- intdash ROS2Bridge で動画を H.264 に変換したうえで、intdash Edge Agent から intdash サーバーに送信する

準備として、ROS2 空間で画像を送信するノードを準備します。

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 ~/ws/src
```

2. ワークスペース内に、gscam2 (ROS2 camera driver for GStreamer-based video streams) をダウンロードします。

```
$ cd ~/ws
$ cd src/
$ git clone https://github.com/clydemcqueen/gscam2.git -b foxy
$ git clone https://github.com/ptrmu/ros2_shared.git
```

3. ビルドを実行します。

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

(次のページに続く)

(前のページからの続き)

```
$ colcon build
```

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. 画像送信ノード用パラメーター設定ファイル (`/opt/vm2m/etc/params_image.yaml`) を以下のように作成します。

   params_image.yaml

```yaml
gscam_publisher:
   ros__parameters:
      gscam_config: 'videotestsrc pattern=smtpe75 ! video/x-raw,format=YUY2,width=320,height=240,
↪framerate=1/1 ! videoconvert'
      preroll: False
      use_gst_timestamps: False
      camera_name: 'my_camera'
      frame_id: 'my_camera_frame'
```

Data Visualizer で H.264 を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。

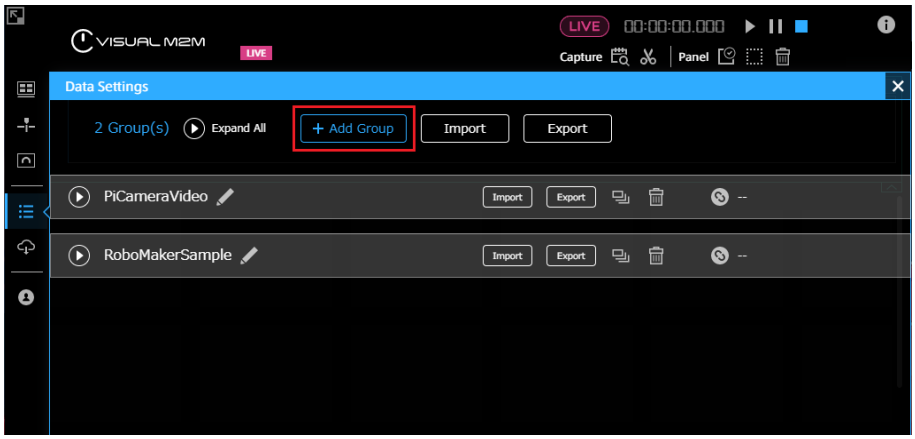


図 17　Group を追加

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。

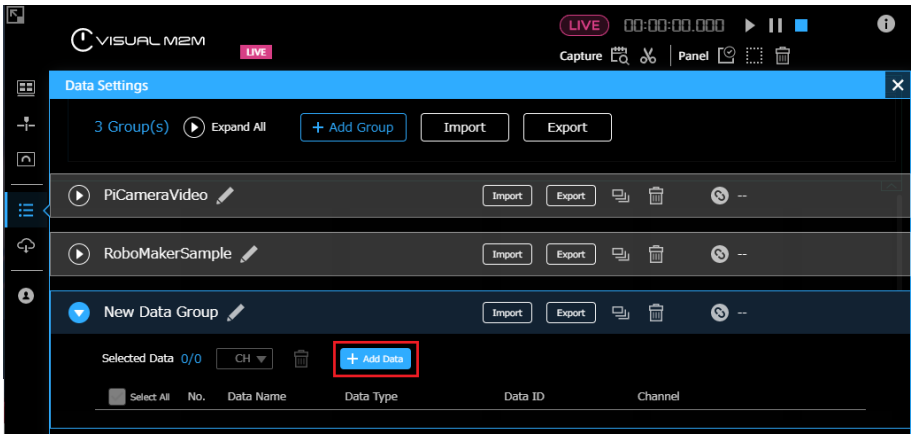
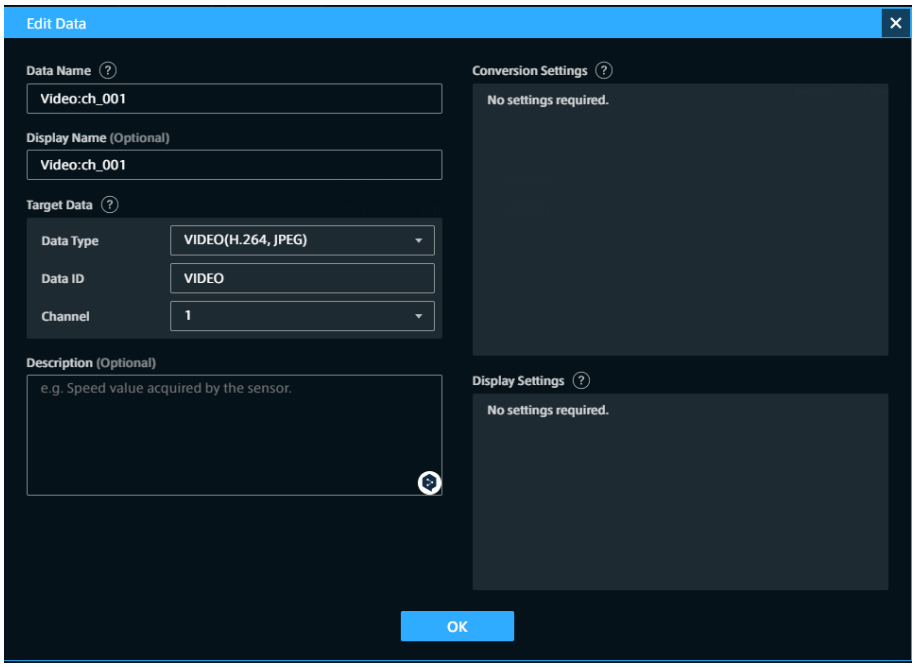


図 18　Data を追加

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



[OK] をクリックし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。

次に、DataVisualizer でビジュアルパーツを配置します。

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

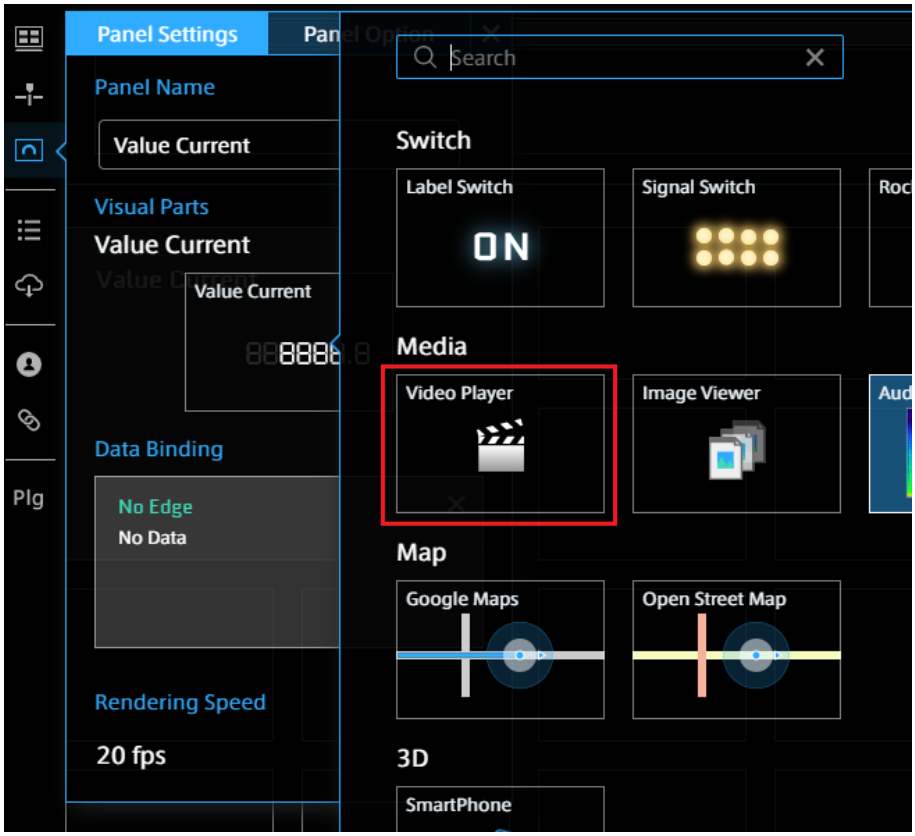


図 19　Video Player を選択

2. 先ほど作成したデータ設定を使って、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。

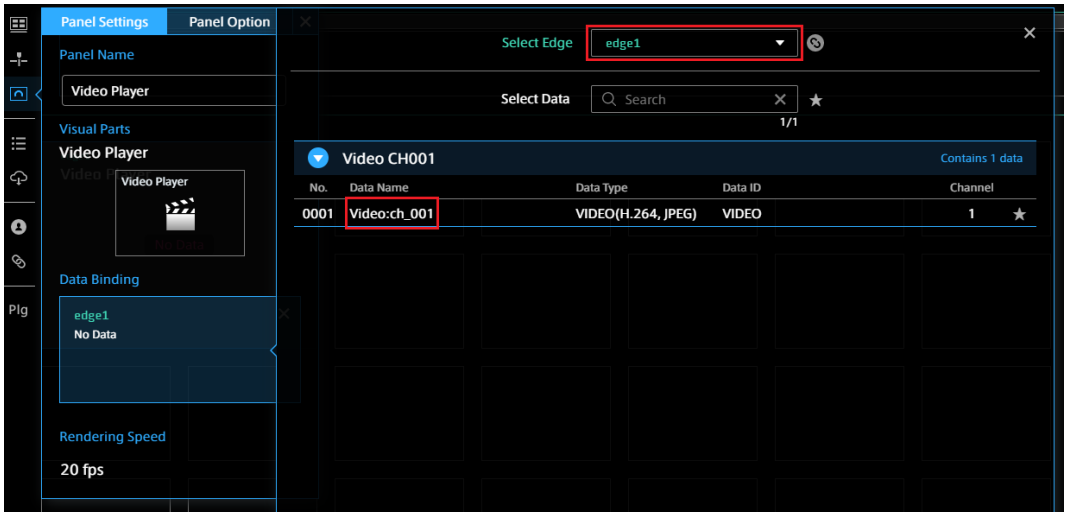


図 20　データをバインド

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. エッジデバイス 1 の intdash ROS2Bridge 設定ファイルを以下のように作成します。

params.yaml

```yaml
upstream:
   enabled: true
   formats:
   - format: "h264_1"
      writer:
         path: "/opt/vm2m/var/run/intdash/logger_001.tx"
         buffering: true

subscribers:
   enabled: true
   topics:
   - topic_name: "/sensor_msgs/image_raw"
      format:
      - "h264_1"
      pipeline : "\
         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"
```

5. エッジデバイス 1 で intdash ROS2Bridge と intdash Edge Agent を起動します。
6. エッジデバイス 1 で以下のコマンドを実行します。

```
$ source ~/ws/install/setup.bash
$ ros2 run gscam2 gscam_main --ros-args --remap /image_raw:=/sensor_msgs/image_raw --params-file /opt/
↪vm2m/etc/params_image.yaml
```

Data Visualizer に、エッジデバイス 1 からの動画が表示されれば成功です。

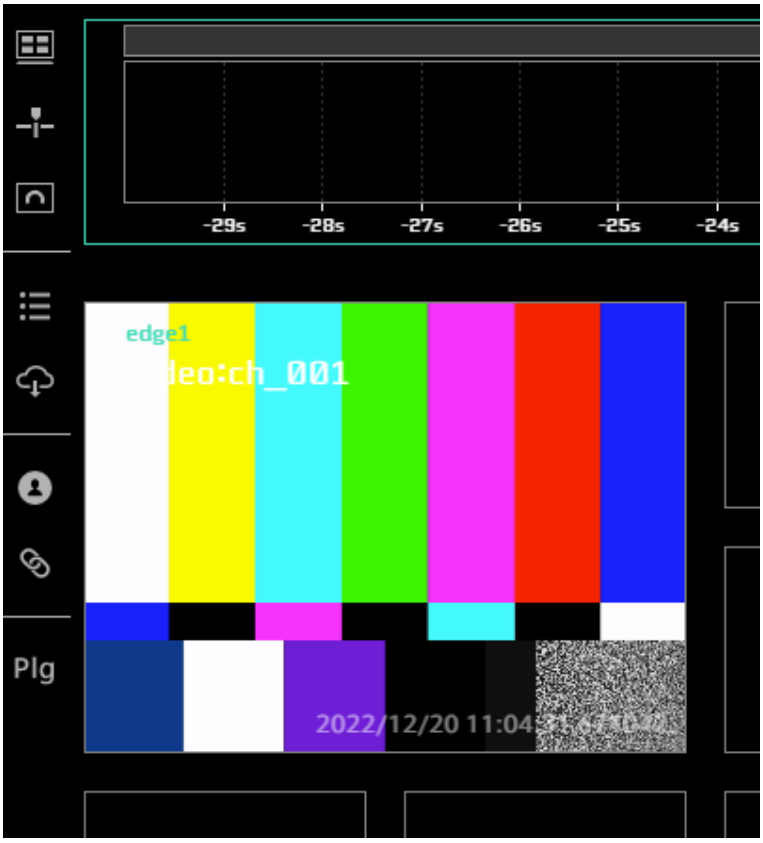


図 21　Video Player での動画の表示

04 intdash ROS2Bridge の設定

intdash ROS2Bridge の設定は、yaml 形式の設定ファイルで行います。ここでは設定可能な項目について説明します。

4.1 コールバックのスレッド数

コールバックのスレッド数は、トピックのサブスクリプションや、サービスリクエスト、アクションゴールリクエストなどを並列に実行することができる最大数です。

設定例:

```
num_callback_threads: 15
```

フィールド名	enum	型	必須/オプショナル	説明
num_callback_threads	-	unsigned int	オプション (デフォルトは 10)	並列にコールバックを実行するために、スレッド数を指定します。実装では MultiThreadedExecutor() のコンストラクタの number_of_threads として使われます。参考 (foxy): rclcpp::executors::MultiThreadedExecutor Class Reference

4.2 FIFO の設定

intdash ROS2Bridge と intdash Edge Agent の間でデータのやり取りをする FIFO の設定を行います。

フィールド名	enum	型	必須/オプショナル	説明
upstream	-	map	オプション	アップストリーム（ROS2 → intdash）の設定を記述します。upstream の設定（ROS2 → intdash）(p. 39) を参照してください。
downstream	-	map	オプション	ダウンストリーム（intdash → ROS2）の設定を記述します。downstream の設定（intdash → ROS2）(p. 41) を参照してください。

4.2.1 upstream の設定（ROS2 → intdash）

設定例:

```yaml
upstream:
  enabled: true
  formats:
  - format: "cdr"
    writer:
      path: "/var/run/intdash/logger_001.tx"
      buffering: false
  - format: "json"
    writer:
      path: "/var/run/intdash/logger_002.tx
      buffering: false
      max_array_size: 100
```

upstream

upstream では以下の項目を設定可能です。

フィールド名	enum	型	必須/オプショナル	説明
enabled	true false	bool	オプション（デフォルトは false）	アップストリームを有効にします。
formats	-	map	必須	アップストリームで使用するフォーマットを指定します。upstream.formats (p. 40) を参照してください。

upstream.formats

formats では以下の項目を設定可能です。

フィールド名	enum	型	必須/オプショナル	説明
`format`	-	string	必須	フォーマットを指定します。以下のいずれかを指定してください。 • `"cdr"` • `"json"` • JPEG の場合は `"jpeg_1"`、`"jpeg_2"`、`"jpeg_3"`、`"jpeg_4"` のいずれか • H.264 の場合は `"h264_1"`、`"h264_2"`、`"h264_3"`、`"h264_4"` のいずれか
`writer`	-	map	必須	アップストリームで使用するフォーマットの情報を指定します。upstream.formats[].writer (p. 40) を参照してください。

注釈: jpeg/h264 モードが送信可能な ROS メッセージは以下のタイプです。
- sensor_msgs/Image
- sensor_msgs/CompressedImage

upstream.formats[].writer

format が `"cdr"` の場合は以下の項目を設定可能です。

フィールド名	enum	型	必須/オプショナル	説明
`path`	-	string	必須	FIFO のパスを設定します。指定がない場合は"cdr"による FIFO は有効になりません。
`buffering`	`true` `false`	bool	オプション (デフォルトは true)	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を設定します。なお、内部では、バッファを有効にするかを setvbuf 関数により設定しています。

format が"json"の場合は以下の項目を設定可能です。

フィールド名	enum	型	必須/オプショナル	説明
path	-	string	必須	FIFO のパスを設定します。指定がない場合は"json"による FIFO は有効になりません。
max_array_size	-	int	オプション（デフォルトは無制限）	JSON の配列に書き込まれる要素数を制限します。この設定は ROS メッセージにのみ適用されます。サービスやアクションを識別する UUID この制限を受けません。
buffering	true false	bool	オプション（デフォルトは 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を設定します。なお、内部では、バッファを有効にするかを setvbuf 関数により設定しています。

4.2.2 downstream の設定（intdash → ROS2）

設定例:

```
downstream:
  enabled: true
  format: "cdr"
  reader:
    path: "/var/run/intdash/logger_001.rx"
```

downstream

downstream は以下の項目を設定可能です。

フィールド名	enum	型	必須/オプショナル	説明
enabled	true false	bool	オプション（デフォルトは false ）	ダウンストリームを有効にします。
format	-	string	必須	ダウンストリームで使用するフォーマットの情報を指定します。"cdr"を設定可能です。
reader	-	map	必須	format の設定を記述します。downstream.reader（p. 42）を参照してください。

downstream.reader

format が"cdr"の時は以下の項目を設定可能です。

フィールド名	enum	型	必須/オプショナル	説明
path	-	string	必須	FIFO のパスを設定します。指定がない場合は"cdr"による FIFO は有効になりません。

4.3 QoS（Quality of Service）の設定

QoS は、subscribers (p. 46) 、publishers (p. 49) 、service_clients (p. 53) 、parameter_clients (p. 58) 、parameter_service_servers (p. 56) のそれぞれの qos フィールドで設定することが可能です。

> **注意:** QoS を明示的に設定しない場合、rclcpp で設定されるデフォルトと同じものが使用されます。そのため、明示的に設定したい項目以外は記載する必要はありません。詳細は qos.policy (p. 43) を参照してください。

ROS2 の QoS の詳細は以下のドキュメントを参照してください。

foxy: About Quality of Service settings

humble: About Quality of Service settings

設定例:

```
qos:
  profile: "default"
  policy:
    history:       "keep_last"
    depth:         10
    reliability:   "best_effort"
    durability:    "transient_local"
    deadline:      "100msec"
    lifespan:      "100msec"
    liveliness:    "automatic"
    liveliness_lease_duration: "100msec"
```

4.3.1 qos

qos では以下の項目を設定可能です。

フィールド名	enum	型	必須/オプショナル	説明

`profile`	`"default"`
	`"services"`
	`"sensor_data"`
	`"parameters"`
	`"system_default"`
		string	オプション	QoS プロファイルを指定します。enum に該当しないものを設定した場合は、"system_default" が使用されます。詳細は、公式ドキュメントを参照してください。
				• Foxy: About Quality of Service settings - ROS 2 Documentation: Foxy documentation
				• Humble: About Quality of Service settings - ROS 2 Documentation: Humble documentation

`policy`	-	map	オプション	QoS ポリシーを個別に設定します。設定しなかった項目はデフォルトのものが使われます。qos.policy (p. 43) を参照してください。

4.3.2 qos.policy

policy では以下の項目を設定可能です。

それぞれの policy の詳細については ROS2 Documentation の QoS policies を参照してください。

設定をしなかった policy の項目は、rclcpp で設定されるデフォルトと同じものが使用されます。そのため、明示的に設定したい項目以外は記載する必要はありません。

> **注意:**　トピックを publish するノードの `durability` が `transient_local`、subscribe するノードの `durability` が `volatile` の組み合わせで、両者の `reliability` を `reliable` としたい場合は、明示的に QoS を設定する必要があります。

フィールド名	enum	型	必須/オプショナル	説明
history	"keep_all" "keep_last"	string	オプション	history ポリシーを設定します。誤ったものを記述した場合に設定されるものは不定です。
depth	-	int	オプション(デフォルトは 10)	history に keep_last を設定した場合に depth の設定に使用されます。
reliability	"reliable" "best_effort"	string	オプション	reliability の設定をします。誤った値を記述した場合の設定は不定です。
durability	"transient_local" "volatile"	string	オプション	durability の設定をします。誤った値を記述した場合の設定は不定です。
deadline	-	string	オプション	deadline を設定します。"Xmin"、"Xsec"、"Xmsec"、"Xnsec" のフォーマットで指定が可能です。複数の単位組み合わせることはできません。誤った単位を指定した場合は、0 が使用されます。
lifespan	-	string	オプション	ifespan を設定します。"Xmin"、"Xsec"、"Xmsec"、"Xnsec" のフォーマットで指定が可能です。複数の単位を組み合わせることはできません。誤った単位を指定した場合は、0 が使用されます。
liveliness	"automatic" "manual_by_topic" "system_default"	-	オプション	liveliness を設定します。誤った値を記述した場合の設定は不定です。
liveliness_lease_duration	-	string	オプション	liveliness で使用する lease_duration を設定します。"Xmin"、"Xsec"、"Xmsec"、"Xnsec" のフォーマットで指定が可能です。複数の単位を組み合わせることはできません。誤った単位を指定した場合は、0 が使用されます。

4.4 ROS2 トピックの伝送に関する設定

ROS2 空間のトピックをサブスクライブする subscribers と、トピックを ROS2 空間にパブリッシュする publishers を設定可能です。

> **注意:** トピックを適切にブリッジするには、互換性のある QoS を設定する必要があります。
>
> intdash ROS2Bridge(subscriber) の QoS 設定は、エッジデバイス 2 でトピックをサブスクライブするノードの QoS と互換性のある設定にしてください（同じ QoS 設定にすることを推奨します）。
>
> また、intdash ROS2Bridge(publisher) は、エッジデバイス 1 でトピックをパブリッシュしているノードの QoS と互換性のある設定にしてください（同じ QoS 設定にすることを推奨します）。

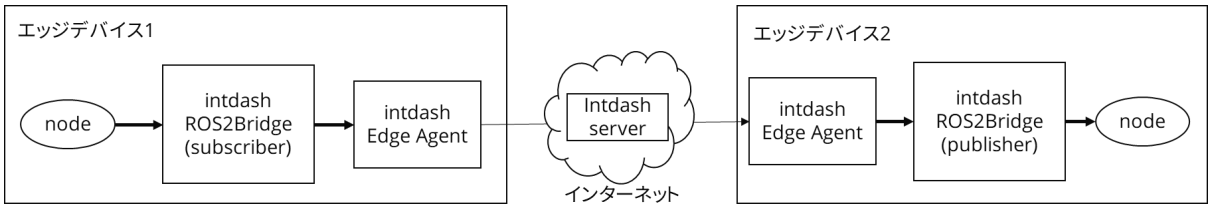


図 22　トピックをブリッジする構成

4.4.1 Subscribers の設定（エッジデバイス 1）

設定例:

```
subscribers:
  enabled: true
  advertise_interval: "5sec"
  topics:
  - topic_name: "aaa"
    format:
    - "cdr"
    - "json"
  - topic_name: "string_msg"
    format:
    - "cdr"
    - "json"
    qos:
 (中略)
  - topic_name: "/tf"
    tf: true # tf
    format:
    - "cdr"
  - topic_name: "/tf_static"
    tf_static: true # tf static
```

(次のページに続く)

(前のページからの続き)

```
    format:
    - "cdr"
  - topic_name: "/oneshot_topic"
    format:
    - "cdr"
    resend_interval: "1sec"
  - topic_name: "/sensor_msgs/image_raw"
    format:
    - "jpeg_1"
    pipeline: "jpegenc ! appsink name=appsink emit-signals=true"
```

subscribers

subscribers では以下の項目を設定可能です。

フィールド名	enum	型	必須/オプショナル	説明
enabled	true false	bool	オプション (デフォルトは false)	true にすると、トピックのサブスクライブと FIFO への書き込みを行います。
advertise_interval	-	string	オプション (デフォルトは 0。毎回書き込まれる)	アップストリームにトピックを書き込むときにトピックの型がメッセージに含まれる間隔を設定します。"XXmin" 、"XXsec" 、"XXmsec" のフォーマットで指定が可能です。複数の単位を組み合わせることはできません。誤った単位を指定した場合は、0 が使用されます。
topics	-	array	オプション	サブスクライブするトピックの個別の設定を記述します。subscribers.topics (p. 46) を参照してください。

subscribers.topics

topics には複数のトピックを設定可能です。それぞれのトピックには以下の設定項目があります。

フィールド名	enum	型	必須/オプショナル	説明
`topic_name`	-	string	必須	トピック名を指定します。トピック名は明示的にネームスペースを付与することが可能です。ネームスペースを付与しなかった場合、つまりトピック名の最初に `/` が含まれない場合は intdash ROS2Bridge と同じネームスペースが付与されます。
`tf`	`true` `false`	bool	オプション（デフォルトは `false`）	`topic_name` が `"tf"` の時は、`true` を指定します。指定しなかった場合、トピック名が `"tf"` であっても通常のトピックと同じ扱いになります。複数トピックに対して設定することはできません
`tf_static`	`true` `false`	bool	オプション	`topic_name` が `"tf_static"` の時は、`true` を指定します。指定しなかった場合、トピック名が `"tf_static"` であっても通常のトピックと同じ扱いになります。複数のトピックに対して設定することはできません
`qos`	-	map	オプション	詳細は qos (p. 43) の設定を参照してください。設定しなかった場合、rclcpp::QoS(10) が使用されます。`tf` が設定されている場合は、デフォルトでは tf2_ros::DynamicListenerQoS() が使用されます。`tf_static` 設定されている場合は、デフォルトでは tf2_ros::StaticListenerQoS() が使用されます。
`format`	-	array of string	必須	upstream で設定したもののなかから 1 つ以上のフォーマットを指定できます。upstream で定義されていないものは無視されます。

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/オプショナル	説明
resend_interval	-	string	オプション (デフォルトは 0。0 の場合、再送は行われません)	トピックを再送する間隔を設定します。"Xmin" 、"Xsec" のフォーマットで指定が可能です（X には数字が入ります)。複数の単位を組み合せることはできません。誤った単位を指定した場合は、0 が設定されます。
pipeline	-	string	オプション (デフォルトは "")	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。

注意: サブスクライブしたいトピックに対する QoS の設定は、パブリッシュされるトピックの QoS と互換性を持つ必要があります。

例えば、点群データなどは一般的には reliability が best_effort で設定されることが多いです。そのようなトピックをサブスクライブする場合、reliability を best_effort に設定しなければいけません。

もし互換性のある QoS を設定しなかった場合、intdash_ros2bridge はトピックをサブスクライブすることができません。その結果、intdash server にデータは流れません。

トピックの QoS は ros2 topic info トピック名 -v コマンドによって確認できます。なお、このコマンドを実行したときに、Lifespan、Deadline、Liveliness、Liveliness の設定も表示されますが、これらの値がシステムのデフォルト値と同じ場合は明示的に設定する必要はありません。

4.4.2 publishers の設定（エッジデバイス 2）

設定例:

```
publishers:
  enabled: true
  suffix: "_suffix"
  topics:
  - topic_name: "string_msg"
    qos:
(中略)
  - topic_name: "/tf"
    tf: true
  - topic_name: "/tf_static"
```

(次のページに続く)

(前のページからの続き)

```
    tf_static: true
  - topic_name: "/oneshot_topic"
    resend: true
```

publishers

publishers では以下の項目を設定可能です。

フィールド名	enum	型	必須/オプショナル	説明
enabled	true false	bool	オプション (デフォルトは false)	true にすると、FIFO から読み込まれたトピックのパブリッシュを行います。
suffix	-	string	オプション (デフォルトは "")	パブリッシュするトピックに suffix を設定します。デバッグ用の設定のため、通常は使用しません。
topics	-	map	オプション	パブリッシュするトピックの個別の設定を記述します。publishers.topics (p. 49) を参照してください。

publishers.topics

ダウンストリームで受信したトピックはすべてパブリッシュされますが、QoS の設定が必要な場合や、tf/tf_static を扱う場合のみ、topics の設定が必要です。

topics には複数のトピックを設定可能です。それぞれのトピックには以下の設定項目があります。

フィールド名	enum	型	必須/オプショナル	説明
topic_name	-	string	必須	トピック名を指定します。トピック名は明示的にネームスペースを付与することが可能です。ネームスペースを付与しなかった場合、つまりトピック名の最初に / が含まれない場合は intdash ROS2Bridge と同じネームスペースが付与されます。
tf	true false	bool	オプション (デフォルトは false)	topic_name が"tf"の時は、true を指定します。指定しなかった場合、トピック名が "tf" であっても通常のトピックと同じ扱いになります。

次のページに続く

表 3 – 前のページからの続き

フィールド名	enum	型	必須/オプショナル	説明
tf_static	true false	bool	オプション (デフォルトは false)	topic_name が "tf_static" の時は、true を指定します。指定しなかった場合、トピック名が "tf_static" であっても通常のトピックと同じ扱いになります。
tf_header_timestamp	default ros_time system_time	string	オプション (デフォルトは、default)	tf または tf_static が true のトピックにおいて、メッセージをパブリッシュする前にヘッダーに含まれるタイムスタンプを更新するかを設定します。 default を設定した場合は更新は行いません。ros_time を設定した場合は intdash_ros2bridge が参照する ROS 時間で更新されます。system_time を設定した場合は、システム時間で更新されます。
qos	-	map	オプション	qos (p. 43) の設定を参照してください。何も設定しない場合は rclcpp::QoS(10) が適用されます。tf を設定した場合は tf2_ros::DynamicBroadcasterQoS() が使用されます。tf_static を設定した場合は tf2_ros::StaticBroadcasterQoS() が使用されます。
resend	true false	bool	オプション (デフォルトは false)	true を設定すると、再送されたトピックの内容を前回送信した内容と比較して、前回と一致した場合パブリッシュしません。差異がある場合はパブリッシュします。

なお、FIFO から読み込まれたトピックをパブリッシュしないように設定することはできません。トピックをパブリッシュしたくない場合は、intdash Edge Agent でそのトピックをダウンストリームの対象外にしてください。

4.5 ROS2 サービスの伝送に関する設定

intdash ROS2Bridge がサービスリクエストをブリッジする `service_servers` と、ブリッジしたサービスリクエストを実際のサービスに送信する `service_clients` を設定が可能です。

2 つの intdash ROS2Bridge が互いにアップストリームを行うため、service_servers 側と service_clients 側の双方に upstream/downstream の設定が必要になります。

また、`format` には `"cdr"` を含む必要があります。

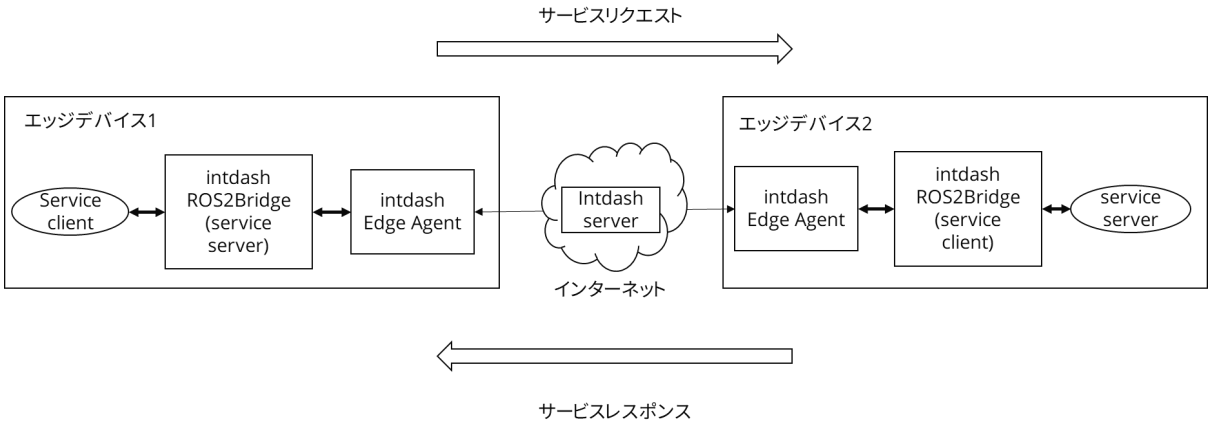


図 23　サービスをブリッジする構成

4.5.1 service_servers の設定（エッジデバイス 1）

設定例:

```
service_servers:
  enabled: true
  services:
  - service_name: "service_name"
    service_type: "package_name/srv/ServiceName"
    format:
      - "cdr"
      - "json"
    qos:
(省略)
```

service_servers

service_servers では以下の項目を設定可能です。

フィールド名	enum	型	必須/オプショナル	説明
`enabled`	`true` `false`	bool	オプション (デフォルトは `false`)	`true` にすると、サービスサーバーを利用可能になります。
`services`	-	array	オプション	サービスサーバーの設定を記述します。service_servers.services (p. 52) を参照してください。

service_servers.services

services では以下の項目を設定可能です。

フィールド名	enum	型	必須/オプショナル	説明
`service_name`	-	string	必須	エッジデバイス 2 の ROS2 空間にある、アクセスしたいサービスサーバー名を指定します。サービスサーバー名は明示的にネームスペースを付与することが可能です。ネームスペースを付与しなかった場合、つまりサービスサーバー名の最初に `/` が含まれない場合は intdash ROS2Bridge と同じネームスペースが付与されます。
`service_type`	-	string	必須	`package_name/srv/ServiceName` の形式で、サービスタイプを指定します。
`qos`	-	map	オプション	qos (p. 43) の設定を参照してください。設定しなかった場合は rclcpp::ServicesQoS() が使用されます。
`format`	-	list of string	必須	フォーマットを指定します。`"cdr"` が含まれていないと、service_clients はサービスリクエストを受け取ることができません。

4.5.2 service_clients の設定（エッジデバイス 2）

設定例:

```
service_clients:
  enabled: true
  response:
    resend_duration: "10min"
    resend_interval: "15sec"
  request:
    timeout: "10sec"
  services:
  - service_name: "service_name"
    service_type: "package_name/srv/ServiceName"
    format:
      - "cdr"
      - "json"
    qos:
 (中略)
```

service_clients

service_clients では以下の項目を設定可能です。

フィールド名	enum	型	必須/オプショナル	説明
enabled	true false	bool	オプション（デフォルトは false）	true にすると、サービスクライアントが利用可能になります。
request	-	map	オプション	サービスリクエストに関する設定を記述します。service_clients.request (p. 54) を参照してください。
response	-	map	オプション	サービスレスポンスの再送に関する設定を記述します。service_clients.response (p. 54) を参照してください。
services	-	map	オプション	サービスクライアントの設定を記述します。service_clients.services (p. 55) を参照してください。

service_clients.response

response では、サービスレスポンスが到達しなかった場合のために、再送間隔と再送を試み続ける時間を設定可能です。

フィールド名	enum	型	必須/オプショナル	説明
`resend_duration`	-	string	オプション デフォルトは 0（0 の場合再送は行われない)	アップストリームにサービスレスポンスの書き込みを繰り返す時間を指定します。`"XXmin"`、`"XXsec"` のフォーマットで指定が可能です。複数の単位を組み合わせることはできません。誤った単位を指定した場合 ("XXmsec"など）は、0 が使用されます。
`resend_interval`	-	string	オプション デフォルトは 0（0 の場合再送は行われない)	アップストリームにサービスレスポンスの書き込みを繰り返す間隔を指定します。`"XXmin"`、`"XXsec"` のフォーマットで指定が可能です。複数の単位を組み合わせることはできません。誤った単位を指定した場合 ("XXmsec"など）は、0 が使用されます。

service_clients.request

request には以下の項目を設定可能です。

フィールド名	enum	型	必須/オプショナル	説明
`timeout`	-	string	オプション (デフォルトは 10 秒)	サービスリクエストをタイムアウトにする時間を設定します。`"XXmin"`、`"XXsec"` のフォーマットで指定が可能です。複数の単位を組み合わせることはできません。誤った単位を指定した場合 ("XXmsec"など) は、`"10sec"` が使用されます。

サービスリクエストがタイムアウトになった場合、service_servers 側にレスポンスは返ってきません。そのため、ユーザーが実装する service_servers にアクセスするサービスクライアントには適切なタイムアウトが設定されている必要があります。

service_clients.services

services では以下の項目を設定可能です。

フィールド名	enum	型	必須/オプショナル	説明
`service_name`	-	string	必須	サービスクライアントがアクセスするサービス名を指定します。サービス名は明示的にネームスペースを付与することが可能です。ネームスペースを付与しなかった場合、つまりサービス名の最初に / が含まれない場合は intdash ROS2Bridge と同じネームスペースが付与されます。
`service_type`	-	string	必須	`package_name/srv/ServiceName` の形式で、サービスタイプを指定します。
`qos`	-	map	オプション	qos (p. 43) の設定を参照してください。何も設定しなかった場合 rclcpp::ServicesQoS() が使用されます。
`format`	-	array of string	必須	フォーマットを指定します。`"cdr"` が含まれていないと、service_servers はサービスレスポンスを受け取ることができません。

4.6 ROS2 パラメータの伝送に関する設定

intdash ROS2Bridge がパラメータリクエストをブリッジする `parameter_service_servers` と、ブリッジしたサービスリクエストを実際のサービスに送信する `parameter_clients` の設定が可能です。

2 地点の intdash ROS2Bridge が相互にデータを送り合うため、parameter_service_servers 側と parameter_clients 側の双方に upstream/downstream の設定が必要になります。

また、`format` は `"cdr"` を含める必要があります。

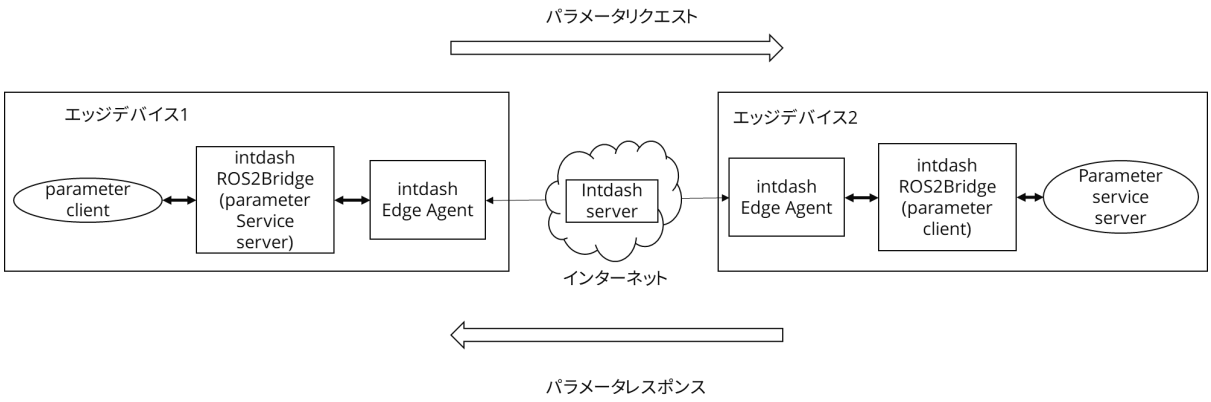


図 24　パラメーターをブリッジする構成

4.6.1 parameter_service_server の設定（エッジデバイス 1）

設定例:

```
parameter_service_servers:
  enabled: true
  nodes:
  - node_name: "bbb"
    format:
    - "cdr"
    - "json"
    qos:
(省略)
```

parameter_service_servers

parameter_service_servers では以下の項目を設定可能です。

フィールド名	enum	型	必須/オプショナル	説明
enabled	true false	bool	オプション (デフォルトは false)	true にすると、パラメータサービスサーバーが利用可能になります。
nodes	-	array	オプション	パラメータサービスサーバーの設定を記述します。parameter_service_servers.nodes (p. 57) を参照してください。

parameter_service_servers.nodes

nodes では以下の項目を設定可能です。

フィールド名	enum	型	必須/オプショナル	説明
node_name	-	string	必須	パラメータサーバーを提供するノード名を指定します。ノード名は明示的にネームスペースを付与することが可能です。ネームスペースを付与しなかった場合、つまりノード名の最初に / が含まれない場合は intdash ROS2Bridge と同じネームスペースが付与されます。
qos	-	map	オプション	qos (p. 43) の設定を参照してください
format	-	array of string	必須	フォーマットを指定します。"cdr" が含まれていないと、parameter_service_clients はパラメータリクエストを受け取ることができません。

4.6.2 parameter_clients の設定（エッジデバイス 2）

設定例:

```
parameter_clients:
  enabled: true
  response:
    resend_duration: "10min"
    resend_interval: "15sec"
  request:
    timeout: "10sec"
  nodes:
  - node_name: "bbb"
    format:
    - "cdr"
    - "json"
    qos:
(省略)
```

parameter_clients

parameter_clients では以下の項目を設定可能です。

フィールド名	enum	型	必須/オプショナル	説明
`enabled`	`true` `false`	bool	オプション (デフォルトは `false`)	`true` にすると、パラメータクライアントが利用可能になります。
`response`	-	map	オプション	パラメータのサービスレスポンスの再送に関する設定を記述します。parameter_clients.response (p. 58) を参照してください。
`request`	-	map	オプション	パラメータのサービスリクエストに関する設定を記述します。parameter_clients.request (p. 59) を参照してください。
`nodes`	-	array	オプション	パラメータに関する設定を記述します。parameter_clients.nodes (p. 59) を参照してください。

parameter_clients.response

response では、パラメータレスポンスが到達しなかった場合のために、再送間隔と再送を試み続ける時間を設定可能です。

フィールド名	enum	型	必須/オプショナル	説明
`resend_duration`	-	string	オプション (デフォルトは 0。0 の場合再送は行われない)	アップストリームにパラメータレスポンスの書き込みを繰り返す時間を指定します。`"XXmin"` 、`"XXsec"` のフォーマットで指定が可能です。複数の単位を組み合わせることはできません。誤った単位を指定した場合 ("XXmsec"など) は、0 が使用されます。
`resend_interval`	-	string	オプション (デフォルトは 0。0 の場合再送は行われない)	アップストリームにパラメータレスポンスの書き込みを繰り返す間隔を指定します。`"XXmin"` 、`"XXsec"` のフォーマットで指定が可能です。複数の単位を組み合わせることはできません。誤った単位を指定した場合 ("XXmsec"など) は、0 が使用されます。

parameter_clients.request

request では以下の項目を設定可能です。

フィールド名	enum	型	必須/オプショナル	説明
`timeout`	-	string	オプション (デフォルトは 10 秒)	パラメータリクエストをタイムアウトにする時間を設定します。`"XXmin"`、`"XXsec"` のフォーマットで指定が可能です。複数の単位を組み合わせることはできません。誤った単位を指定した場合は、"10sec"が使用されます。"msec"を指定した場合は、"10sec"が使用されます。

パラメータリクエストがタイムアウトになった場合、parameter_servers 側にレスポンスは返ってきません。そのため、ユーザーが実装する parameter_servers にアクセスするパラメータクライアントには適切なタイムアウトが設定されている必要があります。

parameter_clients.nodes

nodes では以下の項目を設定可能です。

フィールド名	enum	型	必須/オプショナル	説明
`node_name`	-	string	必須	パラメータライアントがアクセスするノード名を指定します。ノード名は明示的にネームスペースを付与することが可能です。ネームスペースを付与しなかった場合、つまりノード名の最初に `/` が含まれない場合は intdash ROS2Bridge と同じネームスペースが付与されます。
`qos`	-	map	オプション	qos (p. 43) の設定を参照してください。設定しなかった場合は rclcpp::ParametersQoS() が使用されます。
`format`	-	array of string	必須	フォーマットを指定します。`"cdr"` が含まれていないと、parameter_service_servers はパラメータレスポンスを受け取ることができません。

4.7 ROS2 アクションの伝送に関する設定

intdash ROS2Bridge がアクションリクエストをブリッジする `action_servers` と、ブリッジしたアクションリクエストを実際のアクションサーバーに送信する `action_clients` の設定が可能です。

2 地点の intdash ROS2Bridge が相互にデータを送り合うため、action_servers 側と action_clients 側の双方に upstream/downstream の設定が必要になります。

また、format は `"cdr"` を含む必要があります。

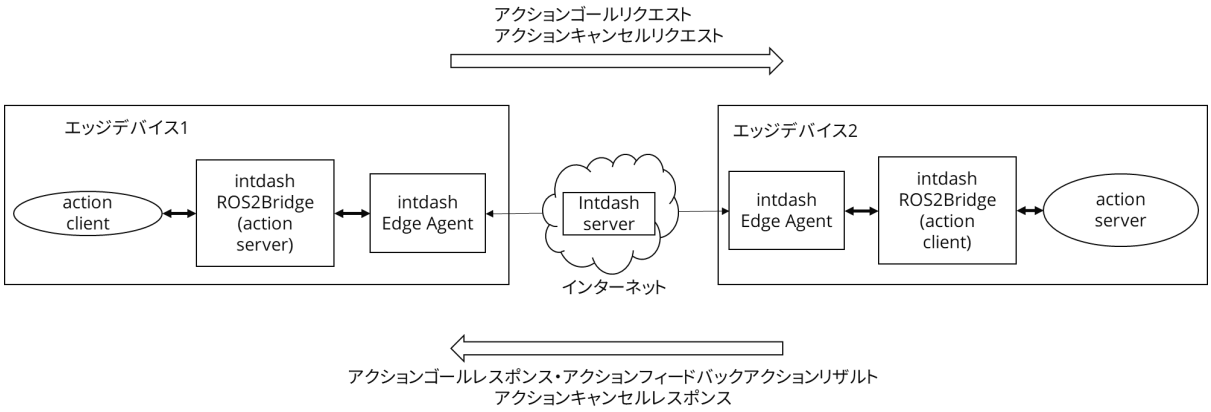


図 25　アクションをブリッジする構成

4.7.1 action_servers の設定（エッジデバイス 1）

設定例:

```
action_servers:
  enabled: true
  request:
    timeout: "10min"
  result:
    timeout: "10min"
  actions:
  - action_name: "aaa"
    action_type: "action_tutorials_interfaces/action/Fibonacci"
    format:
      - "cdr"
      - "json"
```

action_servers

action_servers では以下の項目を設定可能です。

フィールド名	enum	型	必須/オプショナル	説明
`enabled`	`true` `false`	bool	オプション (デフォルトは `false`)	`true` にすると、アクションサーバーが利用可能になります。
`request`	-	map	オプション	ゴールリクエストとキャンセルリクエストのタイムアウトを設定します。action_servers.request (p. 61) を参照してください。
`result`	-	map	オプション (デフォルトは 20 分)	リザルトのタイムアウトを設定します。アクションが完了するのに必要な十分な時間を設定することが推奨されます。action_servers.result (p. 62) を参照してください。
`actions`	-	map	オプション	アクションサーバーの設定を記述します。action_servers.actions (p. 63) を参照してください。

action_servers.request

request では、action_clients にリクエストが届かなかった場合のタイムアウトを設定することが可能です。なお、設定したタイムアウトまでにリクエストへのレスポンスが返ってこなかった場合は、`rclcpp_action::GoalResponse::REJECT` がリクエストの結果として返ってきます。

なお、タイムアウトになり `rclcpp_action::GoalResponse::REJECT` の後に正しいレスポンスが返ってきた場合であってもユーザーが実装したアクションクライアントは処理を継続しないため、設定する場合は十分大きい時間を設定することを推奨します。

フィールド名	enum	型	必須/オプショナル	説明
`timeout`	-	string	オプション（デフォルトは `"15min"`）	ゴールリクエストとキャンセルリクエストのタイムアウトを設定します。`"XXmin"`、`"XXsec"` のフォーマットで指定が可能です。複数の単位を組み合わせることはできません。誤った単位を指定した場合（"XXmsec"など）は、`"15min"` が使用されます。タイムアウトまでにレスポンスを受信できなかった場合は、REJECT が返ってきます。使用しない場合は"0min"または"0sec"を設定します。

action_servers.result

result では、action_servers にゴールリザルトが返ってこなかった場合のタイムアウトを設定することが可能です。

なお、設定したタイムアウトまでに action_servers がリザルトを受け取れなかった場合、アクションの結果を rclcpp_action::ResultCode::UNKNOWN としてクライアントに送信します。

この場合、rclcpp_action::ResultCode::UNKNOWN の送信後に正しいリザルトが返ってきたとしても、ユーザーが実装したアクションクライアントの処理が持つアクションのリザルトは UNKNOWN から変わりません。したがって、使用する場合はアクションを終了するのに十分な時間を設定することを推奨します。

フィールド名	enum	型	必須/オプショナル	説明
`timeout`	-	string	オプション（デフォルトは `"20min"`）	ゴールリザルトのタイムアウトを設定します。`"XXmin"`、`"XXsec"` のフォーマットで指定が可能です。複数の単位を組み合わせることはできません。誤った単位を指定した場合（"XXmsec"など）は、"20min"が使用されます。タイムアウトまでにリザルトを受信できなかった場合は、UNKNOWN を返します。使用しない場合は"0min"または"0sec"を設定します。

action_servers.actions

actions では以下の項目を設定可能です。

フィールド名	enum	型	必須/オプショナル	説明
`action_name`	-	string	必須	アクションサーバー名を指定します。アクションサーバー名は明示的にネームスペースを付与することが可能です。ネームスペースを付与しなかった場合、つまりアクションサーバー名の最初に / が含まれない場合は intdash ROS2Bridge と同じネームスペースが付与されます。
`action_type`	-	string	必須	package_name/srv/ServiceName の形式で、サービスタイプを指定します。
`format`	-	list of string	必須	フォーマットを指定します。"cdr" が含まれていないと action_clients はゴールリクエストやキャンセルリクエストをを受け取ることができません。

4.7.2 action_clients の設定（エッジデバイス 2）

設定例:

```yaml
action_clients:
  enabled: true
  response:
    resend_duration: "10min"
    resend_interval: "15sec"
  actions:
  - action_name: "aaa"
    action_type: "action_tutorials_interfaces/action/Fibonacci"
    format:
      - "cdr"
      - "json"
```

action_clients

action_clients では以下の項目を設定可能です。

フィールド名	enum	型	必須/オプショナル	説明
`enabled`	`true` `false`	bool	オプション（デフォルトは `false`）	`true` にすると、アクションクライアントが利用可能になります。
`response`	-	map	オプション	アクションゴールレスポンス、アクションキャンセルレスポンス、アクションリザルトの再送に関する設定を記述します。action_clients.response (p. 64) を参照してください。
`actions`	-	map	オプション	アクションクライアントの設定を記述します。action_clients.actions (p. 65) を参照してください。

action_clients.response

response では、アクションレスポンス、キャンセルレスポンス、アクションリザルトが action_servers に到達しなかった場合のために、再送間隔と再送を試み続ける時間を設定可能です。

なお、フィードバックの再送機能はありません。

フィールド名	enum	型	必須/オプショナル	説明
`resend_duration`	-	string	オプション（デフォルトは 0。0 の場合再送は行われない）	レスポンスやリザルトの書き込みを繰り返す時間を指定します。`"XXmin"`、`"XXsec"` のフォーマットで指定が可能です。複数の単位を組み合わせることはできません。誤った単位を指定した場合（"XXmsec"など）は、0 が使用されます。
`resend_interval`	-	string	オプション（デフォルトは 0。0 の場合再送は行われない）	レスポンスやリザルトの書き込みを繰り返す間隔を指定します。`"XXmin"`、`"XXsec"` のフォーマットで指定が可能です。複数の単位を組み合わせることはできません。誤った単位を指定した場合（"XXmsec"など）は、0 が使用されます。

action_clients.actions

actions では以下の項目を設定可能です。

フィールド名	enum	型	必須/オプショナル	説明
`action_name`	-	string	必須	アクションクライアントがアクセスするアクション名を指定します。アクション名は明示的にネームスペースを付与することが可能です。ネームスペースを付与しなかった場合、つまりアクション名の最初に `/` が含まれない場合は intdash ROS2Bridge と同じネームスペースが付与されます。
`action_type`	-	string	必須	`package_name/action/ActionName` の形式で、アクションタイプを指定します。
`format`	-	array of string	必須	フォーマットを指定します。`"cdr"` が含まれていないと、action_servers はレスポンスやフィードバックを受け取ることができません。

05 ROS2 メッセージの種類と intdash データ ID の対応関係

intdash ROS2Bridge を使って ROS2 から intdash に渡されるメッセージは、以下のルールに従って ID が付与されます。

メッセージの分類	メッセージの種類	メッセージ ID
Topic	Topic	/topic:<topic_name>
Service	Request	/srv/req:<service_name>
Service	Response	/srv/resp:<service_name>
Action	Goal Request	/act/goal_req:<action_name>
Action	Goal Response	/act/goal_resp:<action_name>
Action	Feedback	/act/fb:<action_name>
Action	Result	/act/result:<action_name>
Action	Cancel Request	/act/cancel_req:<action_name>
Action	Cancel Response	/act/cancel_resp:<action_name>
Parameter	Request	/param/req:<node_name>
Parameter	Response	/param/resp:<node_name>

なお、<topic_name>、<service_name>、<action_name>、<node_name>の先頭に / が含まれる場合は / も ID に含まれます。

06 ROS2 メッセージの JSON 表現

ROS2 メッセージを intdash で伝送する際の形式として JSON を選択した場合、JSON データには以下のようにメタデータと ROS メッセージが格納されます。

6.1 トピックに関するメッセージ

```
{
  "msg": {
    # ROS トピックのメンバが含まれます
  }
}
```

6.2 サービスに関するメッセージ

ROS2 では 1 つのサービスリクエスト、サービスレスポンスは、16 個の 8 ビット整数 "writer_guid" とシーケンス番号 "sequence_number" で識別されます。

6.2.1 サービスリクエスト

```
{
  "writer_guid": [
    # 16 個の 8bit 整数が含まれます
  ],
  "sequence_number": # リクエストを識別するシーケンス番号が含まれます
  "msg": {
    # サービスリクエストが含まれます
  }
}
```

6.2.2 サービスレスポンス

```
{
  "writer_guid": [
  # 16 個の 8bit 整数が含まれます
  ],
  "sequence_number": # リクエストを識別するシーケンス番号が含まれます
  "msg": {
    # サービスレスポンスが含まれます
  }
}
```

6.3 パラメータに関するメッセージ

ROS2 ではパラメータはサービスとして実装されています。そのため 1 つのパラメータリクエスト、パラメータレスポンスは、16 個の 8 ビット整数 "writer_guid" とシーケンス番号 "sequence_number" で識別されます。

6.3.1 パラメータリクエスト

```
{
  "writer_guid": [
  # 16 個の 8bit 整数が含まれます
  ],
  "sequence_number": # リクエストを識別するシーケンス番号が含まれます
  "msg": {
    # パラメータリクエストが含まれます
  }
}
```

6.3.2 パラメータレスポンス

```
{
  "writer_guid": [
  # 16 個の 8bit 整数が含まれます
  ],
  "sequence_number": # リクエストを識別するシーケンス番号が含まれます
  "msg": {
    # パラメータレスポンスが含まれます
  }
}
```

6.4 アクションに関するメッセージ

ROS2 では 1 つのアクションに紐づくリクエスト、レスポンス、フィードバック、リザルトは、16 個の 8 ビット整数"uuid"で識別されます。そのため、アクションに関わるメッセージには UUID が含まれます。

6.4.1 アクションゴールリクエスト

```
{
  "uuid": [
  # 16 個の 8bit 整数が含まれます
  ],
  "msg": {
    "goal_id": {
      "uuid": [
        # 16 個の 8bit 整数が含まれます
```

(次のページに続く)

(前のページからの続き)

```
      ]
    },
    "goal": {
      # アクションのゴールが含まれます
    }
  }
}
```

6.4.2 アクションゴールレスポンス

```
{
  "uuid": [
            # 16 個の 8bit 整数が含まれます
  ],
  "goal_response": # "REJECT"、"ACCEPT_AND_EXECUTE"、"ACCEPT_AND_DEFER" のいずれかが入ります
}
```

6.4.3 アクションフィードバック

```
{
  "uuid": [
            # 16 個の 8bit 整数が含まれます
  ],
  "msg": {
    # ゴールのフィードバックが含まれます
  }
}
```

6.4.4 アクションリザルト

```
{
  "uuid": [
            # 16 個の 8bit 整数が含まれます
  ],
  "result_code": # "SUCCEEDED", "UNKNOWN"、"CANCELED"、"ABORTED"のいずれかが含まれます
  "msg": {
    # アクションリザルトが含まれます
  }
}
```

6.4.5 アクションキャンセルリクエスト

```
{
  "uuid": [
            # 16 個の 8bit 整数が含まれます
  ]
}
```

6.4.6 アクションキャンセルレスポンス

```
{
  "uuid": [
    # 16 個の 8bit 整数が含まれます
  ],
  "result_code": # "SUCCEEDED", "UNKNOWN"、"CANCELED"、"ABORTED"のいずれかが含まれます
}
```

07 制限事項

intdash ROS2Bridge には以下の制限事項があります。

7.1 QoS について

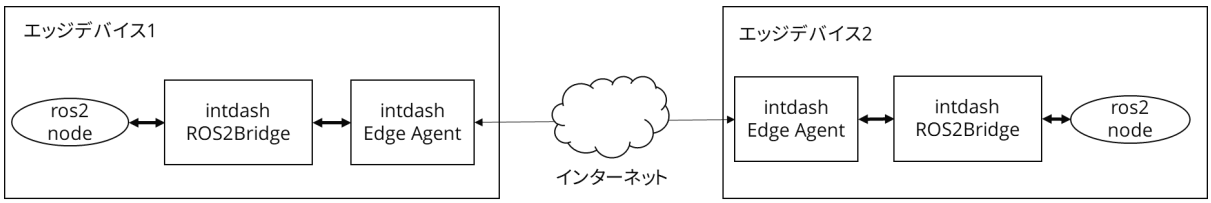


図 26　構成

エッジデバイス 1 内とエッジデバイス 2 内の ROS2 ノードのトピック・サービスに対して互換性がある QoS を設定する必要があります。

また、intdash ROS2Bridge に対して設定する QoS は、ローカルネットワーク内の ROS2 ノードとの通信のみに有効です。そのため、以下のような点を考慮する必要があります。

ポリシー	考慮すべき点
history + depth	エッジデバイス 2 に設定した history と depth はエッジデバイス 2 内の intdashROS2Bridge との間で有効です。
reliability	2 つの intdash Edge Agent 間の通信は基本的には信頼性のある通信ですが、到達保証はされません。エッジデバイス 1 あるいはエッジデバイス 2 と intdashEdgeAgent とサーバー間の通信が切断された場合やサーバーのリソースに問題がある場合にデータが到達しない可能性があります。
durability	エッジデバイス 2 の intdashROS2Bridge で transient_local を設定した場合、エッジデバイス 2 の保持しているトピックを取得することになります。intdash には Transient Local に相当する QoS の仕組みはないため、エッジデバイス 1 から送られたトピックがエッジ 2 に届いていない場合は publish できるトピックはありません。
deadline	deadline を設定する場合、インターネットを経由する通信遅延時間を考慮して設定する必要があります。
lifespan	ROS2 ノード間の lifespan ではなく、ROS2 ノードと intdashROS2Bridge 間の lifespan になります。そのため、publisher 側では削除されているのに ROS2Bridge が保持している可能性があります。
liveliness	ROS2Bridge 間で、liveliness_assert の通知はブリッジされません。そのため、manual_by_topic の使用は推奨されません。

7.2 tf/tf_static の制限

設定ファイル上で tf、tf_static に対応するトピック名を変更することはできません。

7.3 bool 型の可変長配列

bool 型の可変長配列を含むメッセージは扱うことができません。

7.4 サービスについて

intdash Edge Agent は完全なデータの到達保証がないため、サービスリクエストやサービスレスポンスが返ってこない場合があります。そのため、intdash ROS2Bridge の service_clients の設定には、リクエストを再送するための設定があります。

また、サービスクライアントは適切にタイムアウトを設定することが推奨されます。

7.5 時間を入力する場合の上限

設定ファイルで時間を入力する場合、符号なし 64 ビット整数で表すことができる最大の値が上限になります。

ただし、単位が "min" の場合は、入力値に 60 をかけた結果が、符号なし 64 ビット整数で表される範囲に収まる必要があります。

08 付録：intdash Edge Agent 用 manager.conf のサンプル

本書においてエッジデバイス 1 とエッジデバイス 2 で使用した intdash Edge Agent 用設定ファイル manager.conf は以下のとおりです。

8.1 エッジデバイス 1 用 manager.conf のサンプル

dst_id と ctlr_id で、エッジデバイス 2 の UUID を設定する必要があります。

```
{
  "manager": {
    "meas_root": "$APPDIR/intdash/meas",
    "rawdir": "$APPDIR/intdash/raw",
    "basetime": "$RUNDIR/intdash/basetime",
    "stat": "$RUNDIR/intdash/manager.stat",
    "logger_stat": "$RUNDIR/intdash/logger_%03hhu.stat",
    "process_stat": "$RUNDIR/intdash/process.stat",
    "intdash_stat": "$RUNDIR/intdash/intdash.stat",
    "network_stat": "$RUNDIR/intdash/network.stat",
    "system_stat": "$RUNDIR/intdash/system.stat",
    "wwan_stat": "$RUNDIR/intdash/wwan.stat",
    "workdirs": [
      "$APPDIR/intdash/meas",
      "$RUNDIR/intdash"
    ],
    "filters": []
  },
  "clients": [
    {
      "protocol": "mod_websocket.v2",
      "type": "realtime",
      "my_token": "$TOKEN",
      "my_id": "$UUID",
      "my_secret": "$SECRET",
      "auth_path": "$APPDIR/intdash/.auth",
      "connection": {
        "host": "$SERVER",
        "path": "/api/v1/ws/measurements"
      },
      "fifo_rx": "$RUNDIR/intdash/client_%s.rx",
      "fifo_tx": "$RUNDIR/intdash/client_%s.tx",
      "path": "$SBINDIR/intdash-edge-client",
      "stat": "$RUNDIR/intdash/client_%s.stat",
      "fast_net_check_cmd":"$BINDIR/intdash-edge-networkd.sh -q -t",
      "dst_id": ["エッジデバイス 2 の UUID"]
    },
    {
```

(次のページに続く)

(前のページからの続き)

```
    "protocol": "mod_http",
    "type": "resend",
    "my_token": "$TOKEN",
    "my_id": "$UUID",
    "my_secret": "$SECRET",
    "auth_path": "$APPDIR/intdash/.auth",
    "connection": {
      "host": "$SERVER",
      "path": "/api/v1/measurements"
    },
    "fifo_rx": "$RUNDIR/intdash/client_%s.rx",
    "fifo_tx": "$RUNDIR/intdash/client_%s.tx",
    "path": "$SBINDIR/intdash-edge-client",
    "stat": "$RUNDIR/intdash/client_%s.stat",
    "fast_net_check_cmd":"$BINDIR/intdash-edge-networkd.sh -q -t"
  },{
    "protocol": "mod_websocket.v2",
    "type": "control",
    "my_id": "$UUID",
    "my_secret": "$SECRET",
    "auth_path": "$APPDIR/intdash/.auth",
    "connection": {
      "host": "$SERVER",
      "port":443,
      "path":"/api/v1/ws/measurements",
      "ca":"/opt/vm2m/etc/ssl/certs/cacert.pem",
      "ssl":"secure",
      "opts":[]
    },
    "fifo_rx": "$RUNDIR/intdash/client_%s.rx",
    "fifo_tx": "$RUNDIR/intdash/client_%s.tx",
    "path": "$SBINDIR/intdash-edge-client",
    "stat": "$RUNDIR/intdash/client_%s.stat",
    "fast_net_check_cmd":"$BINDIR/intdash-edge-networkd.sh -q -t",
    "ctlr_id": "エッジデバイス 2 の UUID",
    "ctlr_flt_ids": [
      "/srv/resp:/add_two_ints",
      "/act/goal_resp:/fibonacci",
      "/act/fb:/fibonacci",
      "/act/result:/fibonacci",
      "/act/cancel_resp:/fibonacci",
      "/param/resp:/minimal_action_server/get_parameters",
      "/param/resp:/minimal_action_server/describe_parameters",
      "/param/resp:/minimal_action_server/get_parameter_types",
      "/param/resp:/minimal_action_server/get_parameters",
      "/param/resp:/minimal_action_server/list_parameters",
      "/param/resp:/minimal_action_server/set_parameters",
      "/param/resp:/minimal_action_server/set_parameters_atomically"
```

(次のページに続く)

(前のページからの続き)

```
      ],
      "ctlr_ch": 1,
      "ctlr_dtype": 14
    }
  ],
  "loggers": [
    {
      "path": "",
      "connections": [
        {
          "channel": 255,
          "fifo_rx": "$RUNDIR/intdash/logger_255.rx",
          "fifo_tx": "$RUNDIR/intdash/logger_255.tx"
        }
      ],
      "details": {
        "plugin": "status",
        "plugin_dir": "$LIBDIR/plugins",
        "plugin_arg": {
          "stintd": {
            "meas_root": "$RUNDIR/intdash/meas"
          },
          "stsys": {
            "storage_dir": "/"
          }
        }
      }
    },{
      "path": "",
      "connections": [
          {
            "channel": 1,
            "fifo_tx": "$RUNDIR/intdash/logger_001.tx",
            "fifo_rx": "$RUNDIR/intdash/logger_001.rx"
          },{
            "channel": 2,
            "fifo_tx": "$RUNDIR/intdash/logger_002.tx",
            "fifo_rx": "$RUNDIR/intdash/logger_002.rx"
          }
        ],
        "details": {
        "plugin": "fifo"
      }
    }
  ]
}
```

8.2 エッジデバイス 2 用 manager.conf のサンプル

dst_id と ctlr_id で、エッジデバイス 1 の UUID を設定する必要があります。

```
{
  "manager": {
    "meas_root": "$APPDIR/intdash/meas",
    "rawdir": "$APPDIR/intdash/raw",
    "basetime": "$RUNDIR/intdash/basetime",
    "stat": "$RUNDIR/intdash/manager.stat",
    "logger_stat": "$RUNDIR/intdash/logger_%03hhu.stat",
    "process_stat": "$RUNDIR/intdash/process.stat",
    "intdash_stat": "$RUNDIR/intdash/intdash.stat",
    "network_stat": "$RUNDIR/intdash/network.stat",
    "system_stat": "$RUNDIR/intdash/system.stat",
    "wwan_stat": "$RUNDIR/intdash/wwan.stat",
    "workdirs": [
      "$APPDIR/intdash/meas",
      "$RUNDIR/intdash"
    ],
    "filters": []
  },
  "clients": [
    {
      "protocol": "mod_websocket.v2",
      "type": "realtime",
      "my_token": "$TOKEN",
      "my_id": "$UUID",
      "my_secret": "$SECRET",
      "auth_path": "$APPDIR/intdash/.auth",
      "connection": {
        "host": "$SERVER",
        "path": "/api/v1/ws/measurements"
      },
      "fifo_rx": "$RUNDIR/intdash/client_%s.rx",
      "fifo_tx": "$RUNDIR/intdash/client_%s.tx",
      "path": "$SBINDIR/intdash-edge-client",
      "stat": "$RUNDIR/intdash/client_%s.stat",
      "fast_net_check_cmd":"$BINDIR/intdash-edge-networkd.sh -q -t",
      "dst_id": ["エッジデバイス 1 の UUID"]
    },
    {
      "protocol": "mod_http",
      "type": "resend",
      "my_token": "$TOKEN",
      "my_id": "$UUID",
      "my_secret": "$SECRET",
      "auth_path": "$APPDIR/intdash/.auth",
      "connection": {
```

(次のページに続く)

(前のページからの続き)

```
      "host": "$SERVER",
      "path": "/api/v1/measurements"
    },
    "fifo_rx": "$RUNDIR/intdash/client_%s.rx",
    "fifo_tx": "$RUNDIR/intdash/client_%s.tx",
    "path": "$SBINDIR/intdash-edge-client",
    "stat": "$RUNDIR/intdash/client_%s.stat",
    "fast_net_check_cmd":"$BINDIR/intdash-edge-networkd.sh -q -t"
  },
    {
    "protocol": "mod_websocket.v2",
    "type": "control",
    "my_id": "$UUID",
    "my_secret": "$SECRET",
    "auth_path": "$APPDIR/intdash/.auth",
    "connection": {
      "host": "$SERVER",
      "port":443,
      "path":"/api/v1/ws/measurements",
      "ca":"/opt/vm2m/etc/ssl/certs/cacert.pem",
      "ssl":"secure",
      "opts":[]
    },
    "fifo_rx": "$RUNDIR/intdash/client_%s.rx",
    "fifo_tx": "$RUNDIR/intdash/client_%s.tx",
    "path": "$SBINDIR/intdash-edge-client",
    "stat": "$RUNDIR/intdash/client_%s.stat",
    "fast_net_check_cmd":"$BINDIR/intdash-edge-networkd.sh -q -t",
    "ctlr_id": "エッジデバイス 1 の UUID",
    "ctlr_flt_ids": [
      "/topic:/chatter",
      "/srv/req:/add_two_ints",
      "/act/goal_req:/fibonacci",
      "/act/cancel_req:/fibonacci",
      "/param/req:/minimal_action_server/get_parameters",
      "/param/req:/minimal_action_server/get_parameters",
      "/param/req:/minimal_action_server/describe_parameters",
      "/param/req:/minimal_action_server/get_parameter_types",
      "/param/req:/minimal_action_server/get_parameters",
      "/param/req:/minimal_action_server/list_parameters",
      "/param/req:/minimal_action_server/set_parameters",
      "/param/req:/minimal_action_server/set_parameters_atomically"
    ],
    "ctlr_ch": 1,
    "ctlr_dtype": 14
  }
],
"loggers": [
```

(次のページに続く)

(前のページからの続き)

```
    {
      "path": "",
      "connections": [
        {
          "channel": 255,
          "fifo_rx": "$RUNDIR/intdash/logger_255.rx",
          "fifo_tx": "$RUNDIR/intdash/logger_255.tx"
        }
      ],
      "details": {
        "plugin": "status",
        "plugin_dir": "$LIBDIR/plugins",
        "plugin_arg": {
          "stintd": {
            "meas_root": "$RUNDIR/intdash/meas"
          },
          "stsys": {
            "storage_dir": "/"
          }
        }
      }
    },{
      "devicetype": "intdash_ros2bridge",
      "path": "",
      "connections": [
          {
            "channel": 1,
            "fifo_tx": "$RUNDIR/intdash/logger_001.tx",
            "fifo_rx": "$RUNDIR/intdash/logger_001.rx"
          },{
            "channel": 2,
            "fifo_tx": "$RUNDIR/intdash/logger_002.tx"
          }
        ],
        "details": {
        "plugin": "fifo"
      }
    }
  ]
}
```