intdash Bridge
デベロッパーガイド

intdash Bridge Version 1.0.0

第 2 版（2022 年 1 月）

aptpod

Rev. 512f6b6

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 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）。

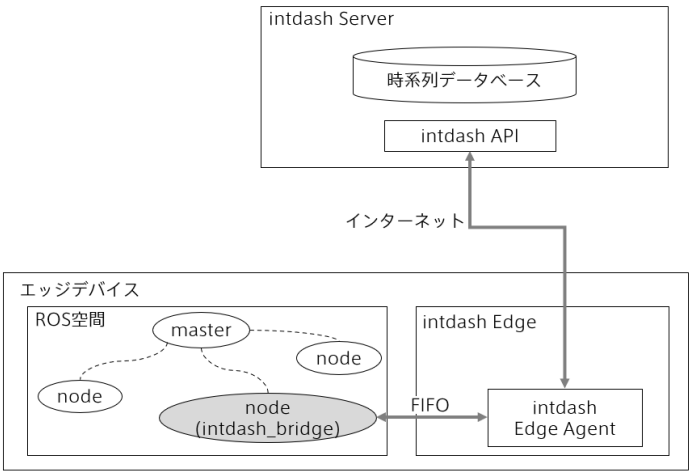

図 1　intdash Bridge の機能

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 のバージョンは以下のとおりです。

- ROS Kinetic
- ROS Melodic
- ROS Noetic

02 使用方法

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

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。

intdash Edge Agent と intdash Bridge 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
```

2. intdash Edge Agent をインストールします。

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

3. 使用する ROS のバージョンに合わせて intdash Bridge をインストールします。

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

2.2 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": [
      {
        "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. 7) 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. 7) 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. 12) 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. 8) 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。

2.3 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。

2.3.1 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) -->
      <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。

2.3.2 パラメーター設定ファイル `params.yaml` 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.yaml` 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。

```yaml
outgoing:
  enabled: true
  buffering: false
  advertise_interval: 5
  max_array_size: 500
  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／オプショナル	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	オプショナル（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	オプショナル（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	オプショナル（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。
`topics[].topic_name`	-	string	必須	ROS 空間でサブスクライブし、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	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。
`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。

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。

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。

2.4 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. 7) 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。

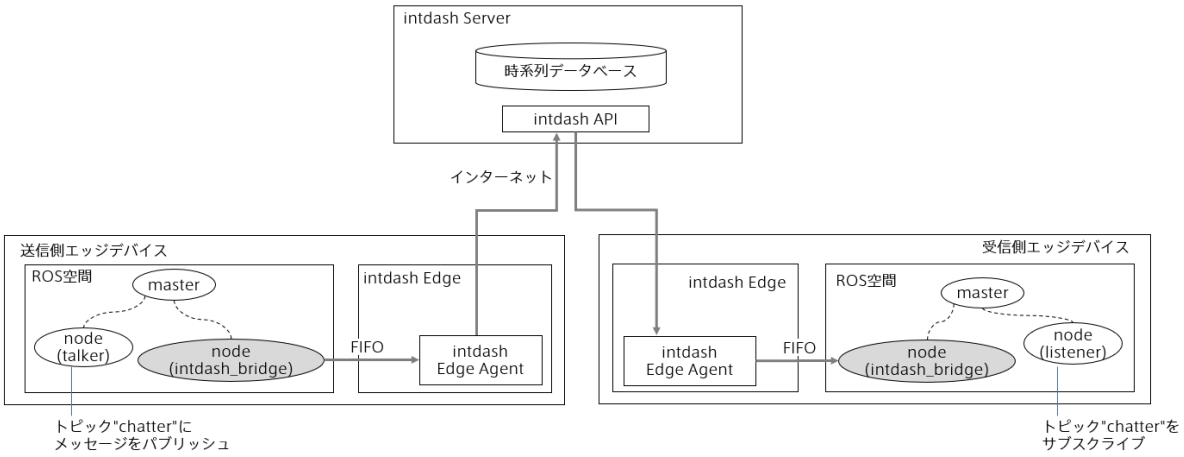

図 2　2 つの ROS 空間を接続する

- 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.github.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":"XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX",
         "my_id":"XXXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX",
         "connection":{
            "host":"xxxxxxx.intdash.jp",
            "path":"/api/v1/ws/measurements"
         },
         "dst_id":[
            "YYYYYYYYY-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":"XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX",
      "my_id":"XXXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXX",
      "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の 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.github.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="" />
    <rosparam 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":"YYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY",
      "my_id":"YYYYYYYYY-YYYY-YYYY-YYYY-YYYYYYYYYYYY",
      "connection":{
        "host":"xxxxxxxx.intdash.jp",
        "port":443,
        "path":"/api/v1/ws/measurements",
        "ca":"/opt/vm2m/etc/ssl/certs/cacert.pem",
        "ssl":"secure",
        "opts":[]
      },
      "ctlr_id":"XXXXXXXXX-XXXX-XXXX-XXXX-XXXXXXXXXXXX",
      "ctlr_flts":[
        {
          "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 `ctlr_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。

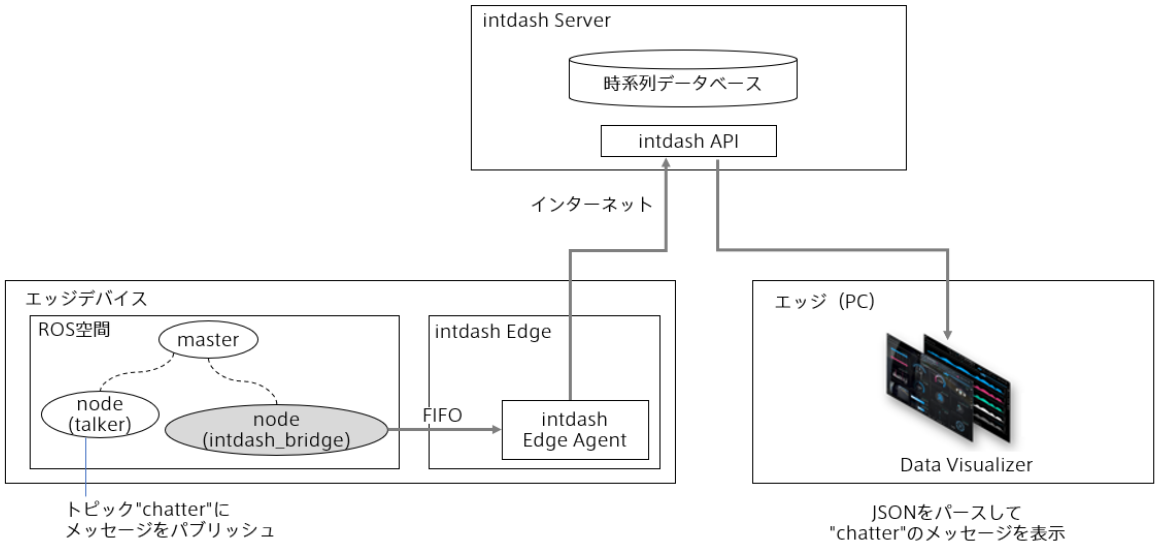

図 3　Data Visualizer で ROS メッセージを表示

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.github.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":"XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX",
         "my_id":"XXXXXXXXX-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":"XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX",
        "my_id":"XXXXXXXXX-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":{"/chatter/data":"hello world 1624436069.38"}}
```

4.2 Data Visualizer で JSON データを表示するためのデータ設定を準備する

intdash サーバーに送信された JSON データを Data Visualizer で可視化するためには、Data Visualizer でデータ設定を行う必要があります。JSON データをパースしてメッセージを取り出す設定を以下のように作成します。

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。

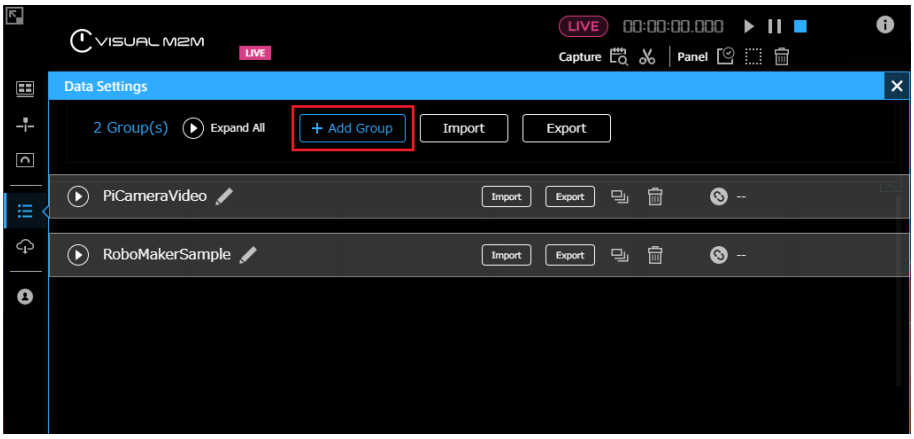

図 4　グループの追加

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。

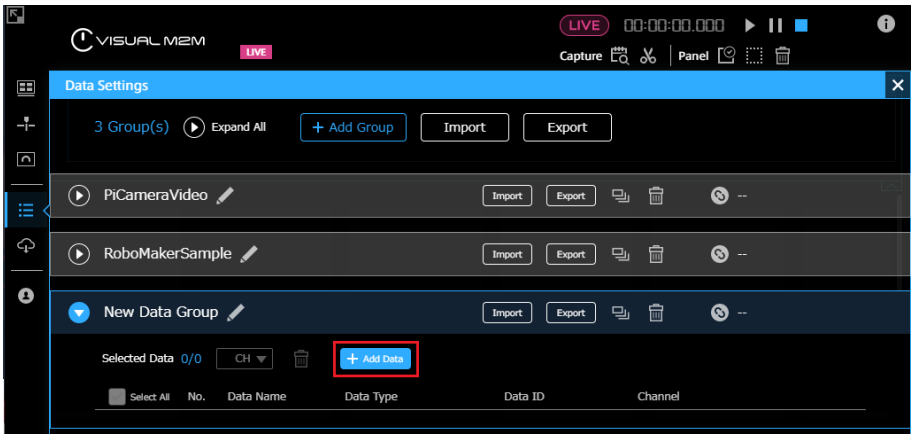

図 5　データの追加

4. String 型データを JSON としてパースし、`msg` 内の `/chatter/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./chatter/data`
        - Value Type: `String`

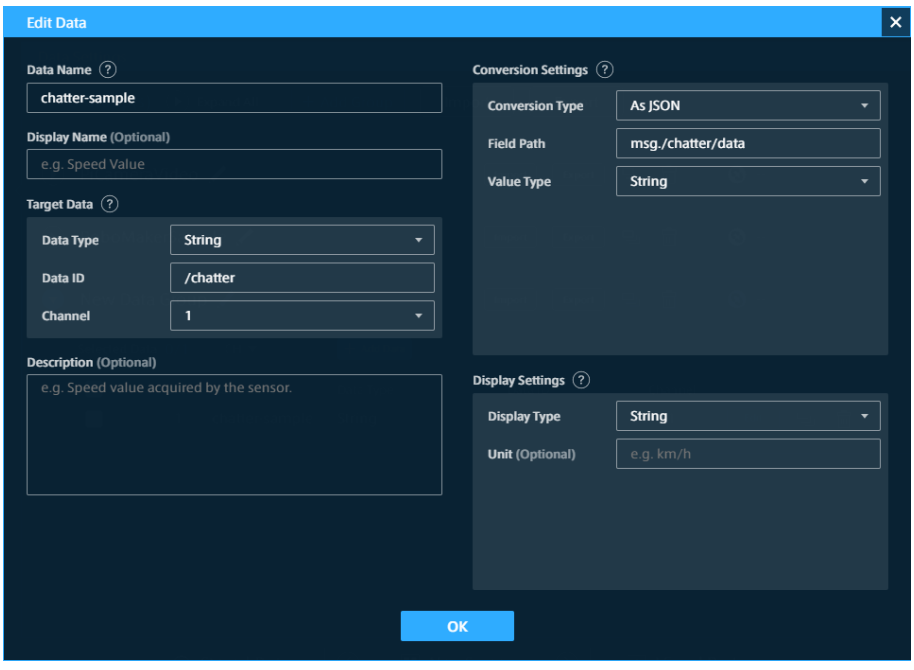

図 6　データ設定

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。

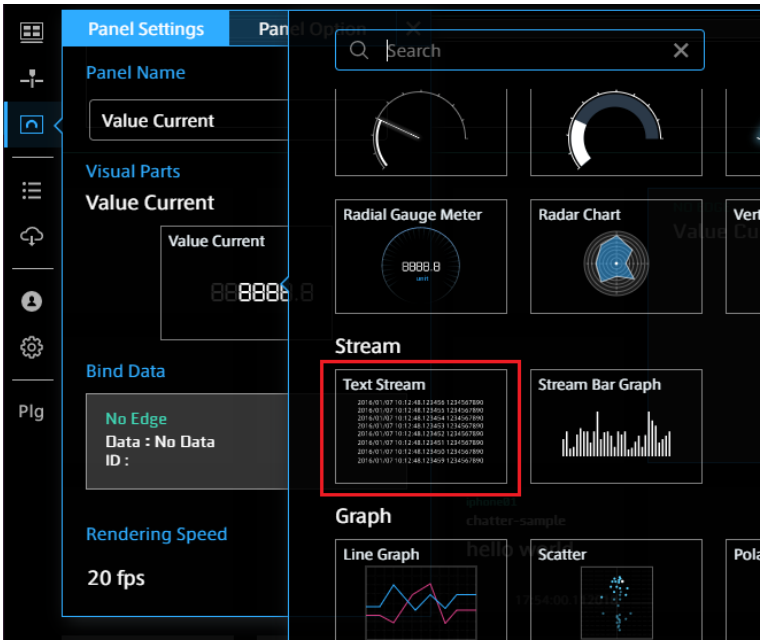

図 7　Text Stream

2. Data Visualizer で JSON データを表示するためのデータ設定を準備する (p. 25) 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。

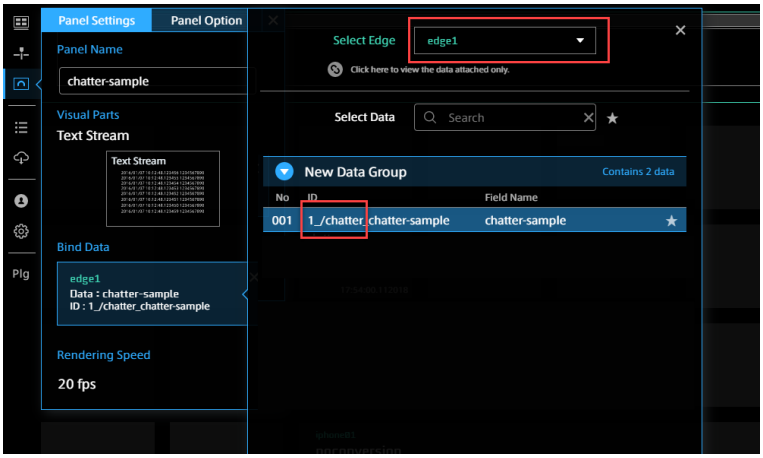

図 8　送信側エッジを選択し、パース設定を選択する

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。

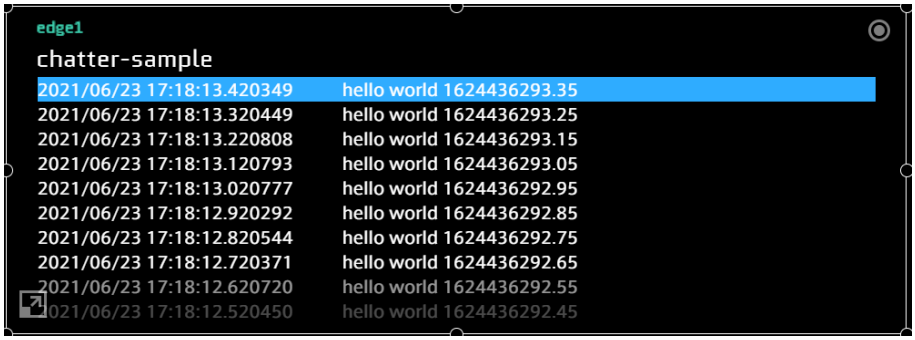

図 9　Text Stream でのメッセージの表示