Autoware デベロッパーズマニュアル

2016/JUN/8

名古屋大学

内容

[はじめに 3](#_Toc453183409)

[概要 3](#_Toc453183410)

[用語 3](#_Toc453183411)

[関連文書 4](#_Toc453183412)

[問い合わせ先 5](#_Toc453183413)

[全体構成 6](#_Toc453183414)

[構成 6](#_Toc453183415)

[主な機能 7](#_Toc453183416)

[環境構築の手順 9](#_Toc453183417)

[Linux 9](#_Toc453183418)

[ROS 9](#_Toc453183419)

[CUDA 10](#_Toc453183420)

[FlyCapture2 11](#_Toc453183421)

[Autoware 11](#_Toc453183422)

[AutowareRider 12](#_Toc453183423)

[canlib 13](#_Toc453183424)

[SSHの公開鍵の作成 13](#_Toc453183425)

[ノードの作成 15](#_Toc453183426)

[開発の流れ 15](#_Toc453183427)

[パッケージの作成 15](#_Toc453183428)

[ノードの作成 17](#_Toc453183429)

[ビルド 18](#_Toc453183430)

[動作確認 19](#_Toc453183431)

[Runtime Manager 22](#_Toc453183432)

[概要 22](#_Toc453183433)

[追加・変更例 22](#_Toc453183434)

[Computingタブから起動・終了するROSノードの追加例 22](#_Toc453183435)

[Computingタブから起動するROSノードへ与えるパラメータの設定例 24](#_Toc453183436)

[パラメータ追加例 26](#_Toc453183437)

[ダイアログで設定したパラメータをrosparamパラメータとして設定する例 29](#_Toc453183438)

[パラメータをコマンドライン引数として出力する場合 33](#_Toc453183439)

[パラメータ設定のその他のkind行指定 36](#_Toc453183440)

[Quick Startタブのボタンで起動・終了するコマンドの設定例 37](#_Toc453183441)

[Sensingタブのボタンで起動・終了するコマンドの設定例 40](#_Toc453183442)

# はじめに

## 概要

この文書は、LinuxとROS(Robot OS)をベースとした、自動運転を実現するためのオープンソースのソフトウェアパッケージ「Autoware」のデベロッパーズマニュアルです。

Autowareに独自の機能を追加するために必要な開発手順、その助けとなる情報について記述しています。

## 用語

* ROS (Robot Operating System)

ロボットソフトウェア開発のためのソフトウェアフレームワーク。ハードウェア抽象化や低レベルデバイス制御、よく使われる機能の実装、プロセス間通信、パッケージ管理などの機能を提供する。

* パッケージ (Package)

ROSを形成するソフトウェアの単位。ノードやライブラリ、環境設定ファイルなどを含む。

* ノード (Node)

単一の機能を提供するプロセス。

* メッセージ (Message)

ノード同士が通信する際のデータ構造。

* トピック (Topic)

メッセージを送受信する先。メッセージの送信を「Publish」、受信を「Subscribe」と呼ぶ。

* OpenCV (Open source Computer Vision library)

コンピュータビジョンを扱うための画像処理ライブラリ。

* Qt

アプリケーション・ユーザ・インタフェースのフレームワーク。

* CUDA (Compute Unified Device Architecture)

NVIDIA社が提供する、GPUを使った汎用計算プラットフォームとプログラミングモデル。

* FlyCapture SDK

PointGrey社のカメラを制御するためのSDK。

* FOT (Field Operation Test)

実道実験。

* GNSS (Global Navigation Satellite System)

衛星測位システム。

* Ladybug SDK

PointGrey社のカメラ「Ladybug」を制御するためのSDK。

* LIDAR (Light Detection and Ranging または Laser Imaging Detection and Ranging)

レーザー照射を利用して距離などを計測する装置。

* DPM (Deformable Part Model)

物体検出手法。

* KF (Kalman Filter)

過去の観測値をもとに将来の状態を推定する手法。

* KLT (Kanade–Lucas–Tomasi feature tracker)

特徴点を抽出し追跡を行う手法。

* NDT (Normal Distributions Transform)

位置推定手法。

* キャリブレーション

カメラに投影された点と3次元空間中の位置を合わせるための、カメラのパラメータを求める処理。

* センサ・フュージョン

複数のセンサ情報を組合せて、位置や姿勢をより正確に算出するなど、高度な認識機能を実現する手法。

* TF (TransForm?)

ROSの座標変換ライブラリ?

* オドメトリ(Odometry)

車輪の回転角と回転角速度を積算して位置を推定する手法。

* SLAM(Simultaneous Localization and Mapping)

自己位置推定と環境地図作成を同時に行うこと。

* CAN(Controller Area Network)

自動車等の内部で相互接続された機器間のデータ転送に使用される規格。

* IMU(Inertial Measurement Unit)

慣性計測装置。角速度や加速度を計測する装置。

* DMI(Distance Measuring Instrument)

走行距離計。

## 関連文書

* Autoware

<http://www.pdsl.jp/fot/autoware/>

* ROS

<http://www.ros.org/>

* OpenCV

<http://opencv.org/>

<http://opencv.jp/>

* Qt

<http://www.qt.io/>

<http://qt-users.jp/>

* CUDA

<http://www.nvidia.com/object/cuda_home_new.html>

<http://www.nvidia.co.jp/object/cuda-jp.html>

* FlyCapture SDK

<https://www.ptgrey.com/flycapture-sdk>

* Ladybug SDK

<https://www.ptgrey.com/ladybug-sdk>

## 問い合わせ先

Autoware Developers (autoware@googlegroups.com)

Autoware Developersメーリングリストの過去の投稿:

<https://groups.google.com/d/forum/autoware>

Autoware Developersメーリングリストへの参加方法:

* Googleアカウントをお持ちの場合、

<https://groups.google.com/d/forum/autoware> にアクセスして、

「グループへの参加を申し込む」ボタンをクリックしてください。

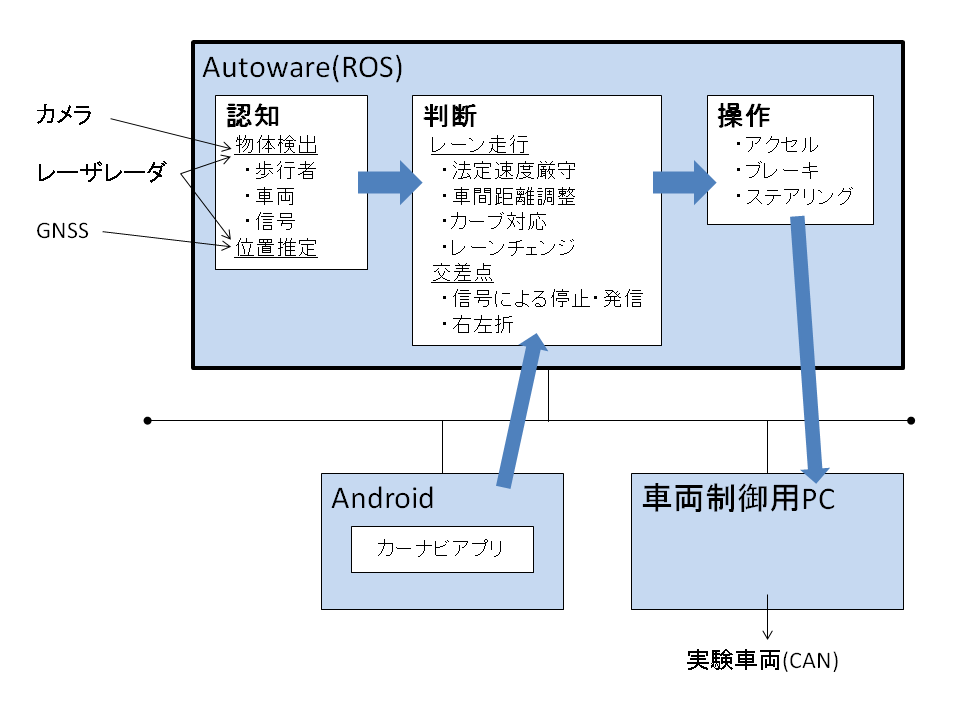
* Googleアカウントをお持ちでない場合:

[autoware+subscribe@googlegroups.com](mailto:autoware+subscribe@googlegroups.com) にメールを送信してください。

# 

# 全体構成

Autowareは、LinuxとROSをベースとした、自動運転を実現するためのオープンソースのソフトウェアパッケージです。レーザレーダ、カメラ、GNSSなどの環境センサを使用して、自車位置や周囲物体を認識しながら、カーナビから与えられたルート上を自立走行することができます。



## 構成

Autowareによる自動運転の機能は、自己位置推定や周囲物体の検出などを行う「認知」、レーンや交差点での走行・停止の「判断」、実際の車両の「操作」、の3つに分けられます。

* ros/src/computing/perception/

認知・判断

* ros/src/computing/planning/

判断・操作

* ros/src/data/

3次元地図などのデータの読み込み(DB、ファイル)

* ros/src/sensing/

各種センサドライバ、キャリブレーション、フュージョンなど

* ros/src/socket/

スマートフォン用アプリケーションとのインタフェース

* ros/src/util/

Runtime Manager、サンプルデータ、擬似ドライバなど

* ui/tablet/

スマートフォン用アプリケーション

* vehicle/

車両の制御、情報取得など

## 主な機能

Autowareには以下のような機能があります。

また、これらを実施するためのユーザインタフェース(Runtime Manager)も用意されています。

* 自己位置推定

3次元点群地図と3次元LIDARデータを入力として、NDTアルゴリズムをベースとしたスキャンマッチングを行うことで、自車位置を10cm程度の誤差で推定することができます。

* 3次元地図生成

SLAM技術を用いて、3次元地図をリアルタイムに生成することができます。

生成した3次元地図を、既存の3次元地図に追加することも可能です。この機能により、3次元地図のオンライン更新も実現できます。

3次元地図から地物データを抽出することで、ベクタ形式の3次元地図を生成することもできます。

* 信号機検出

自己位置推定の結果と高精度3次元地図から、信号機の位置を正確に算出し、信号機の3次元位置をセンサフュージョンによってカメラ画像上に射影します。そこから画像処理によって色判別することで、信号機を検出することができます。

* 物体検出

カメラ画像を入力として、DPMアルゴリズムによる画像認識を行うことで、車両や歩行者を検出することができます。

KFやKLTを利用してトラッキングを行うことも可能です。トラッキング機能を導入すると、個々の物体を追跡でき、かつ誤認識を削減できます。

また、3次元LIDARデータをフュージョンすることで、検出した物体までの距離も算出できます。

* 経路生成

自動運転の経路は、スマートフォンのカーナビアプリケーション(MapFanを使用した経路データ生成アプリケーション)から入力できます。経路には適切な速度情報も含まれ、その速度を目安に自立走行します。

* 経路追従

生成した経路に1m間隔の目印(way point)を設定し、その目印を追っていくことで経路追従を行います。カーブでは近くのway point、直線では遠くのway pointを参照することで、自立走行を安定化しています。

経路から逸脱した場合は、近傍のway pointを目指して経路に戻ります。

# 

# 環境構築の手順

PCに、以下の手順で、Linux、ROS、Autowareなどをインストールする手順を示します。

CUDA、FlyCapture SDKおよびcanlibは、必須ではありません。

NVIDIA社のグラフィックボードに搭載されたGPUを使って計算を行う場合は、CUDAが必要です。また、PointGrey社のカメラを使用する場合は、FlyCapture SDKが必要です。

## Linux

現時点で、Autowareが対応しているLinuxディストリビューションは以下の通りです。

* Ubuntu 14.04
* Ubuntu 15.04

インストールメディアおよびインストール手順については、以下のサイトを参考にしてください。

* Ubuntu Japanese Team

<https://www.ubuntulinux.jp/>

* Ubuntu

<http://www.ubuntu.com/>

## ROS

* Ubuntu14.04の場合は、下記の手順でROSおよび必要なパッケージをインストールします。

$ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu trusty main" > \

/etc/apt/sources.list.d/ros-latest.list'

$ wget http://packages.ros.org/ros.key -O - | sudo apt-key add -

$ sudo apt-get update

$ sudo apt-get install ros-indigo-desktop-full ros-indigo-nmea-msgs \

ros-indigo-nmea-navsat-driver ros-indigo-sound-play

$ sudo apt-get install libnlopt-dev freeglut3-dev qtbase5-dev libqt5opengl5-dev \

libssh2-1-dev libarmadillo-dev libpcap-dev gksu

* Ubuntu15.04の場合は、下記の手順でROSおよび必要なパッケージをインストールします。

$ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb\_release -sc) main" > \

/etc/apt/sources.list.d/ros-latest.list'

$ sudo apt-key adv --keyserver hkp://pool.sks-keyservers.net:80 \

--recv-key 0xB01FA116

$ sudo apt-get install ros-jade-desktop-full ros-jade-nmea-msgs \

ros-jade-nmea-navsat-driver ros-jade-sound-play

$ sudo apt-get install libnlopt-dev freeglut3-dev qt5-default libqt5opengl5-dev \

libssh2-1-dev libarmadillo-dev libpcap-dev gksu

* ~/.bashrcなどに以下を追加します。

Ubuntu14.04の場合:

[ -f /opt/ros/indigo/setup.bash ] && . /opt/ros/indigo/setup.bash

Ubuntu15.04の場合:

[ -f /opt/ros/jade/setup.bash ] && . /opt/ros/jade/setup.bash

## CUDA

注) CUDAに対応したビデオカードがなければ、この作業は不要です。

<http://docs.nvidia.com/cuda/cuda-getting-started-guide-for-linux/> を参考に、以下の手順でインストールします。

1. 環境の確認

$ lspci | grep -i nvidia

(NVIDIAのボードの情報が出力されることを確認)

$ uname -m

(x86\_64であることを確認)

$ gcc --version

(インストールされていることを確認)

1. CUDAのインストール

<http://developer.nvidia.com/cuda-downloads> からCUDAをダウンロード

(以下、cuda-repo-ubuntu1404\_7.0-28\_amd64.deb と想定)

$ sudo dpkg -i cuda-repo-ubuntu1404\_7.0-28\_amd64.deb

$ sudo apt-get update

$ sudo apt-get install cuda

1. システムを再起動 (…は不要かもしれません)

$ lsmod | grep nouveau

(nouveauドライバがロードされていないことを確認)

1. 確認

$ cat /proc/driver/nvidia/version

(カーネルモジュール、gccのバージョンが表示される)

$ cuda-install-samples-7.0.sh ~

$ cd ~/NVIDIA\_CUDA-7.0\_Samples/1\_Utilities/deviceQuery/

$ make

$ ./deviceQuery

1. CUDAを普段から使う場合は、以下の設定を .bashrc などに書く

export PATH=”/usr/local/cuda:$PATH”

export LD\_LIBRARY\_PATH=”/usr/local/cuda/lib:$LD\_LIBRARY\_PATH”

## FlyCapture2

PointGray社のカメラを使用する場合は、以下の手順でFlyCapture SDKをインストールします。(使用しない場合、この作業は不要です。)

1. PointGrey社のサイト(<http://www.ptgrey.com/>)から、FlyCapture SDKをダウンロードします。(ユーザ登録が必要です。)
2. 以下の手順で、事前にパッケージをインストールします。

$ sudo apt-get install libglademm-2.4-1c2a libgtkglextmm-x11-1.2-dev libserial-dev

1. ダウンロードしたアーカイブを展開します。

$ tar xvfz flycapture2-2.6.3.4-amd64-pkg.tgz

1. インストーラを起動します。

$ cd flycapture2-2.6.3.4-amd64/

$ sudo sh install\_flycapture.sh

This is a script to assist with installation of the FlyCapture2 SDK.  
 Would you like to continue and install all the FlyCapture2 SDK packages?  
 (y/n)$ y ← 「y」と答えます  
 ...  
 Preparing to unpack updatorgui-2.6.3.4\_amd64.deb ...  
 Unpacking updatorgui (2.6.3.4) ...  
 updatorgui (2.6.3.4) を設定しています ...  
 Processing triggers for man-db (2.6.7.1-1ubuntu1) ...  
 Would you like to add a udev entry to allow access to IEEE-1394 and USB hardware?  
 If this is not ran then your cameras may be only accessible by running flycap as sudo.  
 (y/n)$ y ← 「y」と答えます

## Autoware

以下の手順でAutowareを入手し、ビルドおよびインストールを行います。

* githubから最新を入手する場合

$ git clone https://github.com/CPFL/Autoware.git  
 $ cd Autoware/ros/src  
 $ catkin\_init\_workspace  
 $ cd ../  
 $ ./catkin\_make\_release

$ source devel/setup.bash

* アーカイブを使用する場合

$ wget http://www.pdsl.jp/app/download/10394444574/Autoware-beta.zip  
 $ unzip Autoware-beta.zip

$ cd Autoware-beta/ros/src  
 $ catkin\_init\_workspace  
 $ cd ../  
 $ ./catkin\_make\_release

$ source devel/setup.bash

## AutowareRider

注) Android Tabletから操作するのでなければ、この作業は不要です。

以下のURLからAPKファイルを入手し、インストールを行います。

* 本体
  + AutowareRider.apk  
    <https://github.com/CPFL/Autoware/blob/master/ui/tablet/AutowareRider/AutowareRider.apk>
* 経路データ生成アプリケーション

注) 経路データを生成しない場合、インストールは不要です。

* + AutowareRoute.apk  
    <https://github.com/CPFL/Autoware/blob/master/ui/tablet/AutowareRoute/AutowareRoute.apk>
* CANデータ収集アプリケーション

注) CANデータを収集しない場合、インストールは不要です。

* + CanDataSender.apk  
    <https://github.com/CPFL/Autoware/blob/master/vehicle/general/android/CanDataSender/bin/CanDataSender.apk>
  + CanGather.apk  
    <https://github.com/CPFL/Autoware/blob/master/vehicle/general/android/CanGather/apk/CanGather.apk>
  + CarLink\_CAN-BT\_LS.apk  
    <https://github.com/CPFL/Autoware/blob/master/vehicle/general/android/CarLink/apk/CarLink_CAN-BT_LS.apk>
  + CarLink\_CANusbAccessory\_LS.apk  
    <https://github.com/CPFL/Autoware/blob/master/vehicle/general/android/CarLink/apk/CarLink_CANusbAccessory_LS.apk>

CanGatherはAPKファイル以外に、設定ファイルを用意する必要があります。

詳細は、以下のURLを参考にしてください。

<https://github.com/CPFL/Autoware/tree/master/vehicle/general/android#cangather-%E3%81%AE%E5%A0%B4%E5%90%88>

## canlib

kvaser のサイト([http://](http://sourceforge.net/projects/opencvlibrary/)[www.kvaser.com/downloads/](http://www.kvaser.com/downloads/)) の ”Kvaser LINUX Driver and SDK” よりソースコード linuxcan.tar.gz を入手し、以下の手順でインストールを行います。

$ tar xzf linuxcan.tar.gz

$ cd linuxcan

$ make

$ sudo make install

## SSHの公開鍵の作成

注) ダイナミックマップを使用しない場合、この作業は不要です。

また、現状では名古屋大のサーバにのみ対応しています。

pos\_db は、SSHを介してデータベースにアクセスします。その際、パスフレーズなしのSSH鍵を使用します。

そのため、pos\_db を使用する場合は、データベースサーバ用のSSH鍵を以下の手順で作成し、SSH公開鍵をデータベースサーバに登録する必要があります。

1. SSH鍵の作成方法
   * 以下のコマンドを実行して鍵を作成します。
     + $ ssh-keygen -t rsa
   * その際、パスフレーズは空(文字列を入力せずにエンターキーを押す)にして作成してください。
   * DSA を使用する場合は -t dsa と指定してください。
2. SSH公開鍵をデータベースサーバに登録する
   * 作成したSSH公開鍵を以下のコマンドでサーバーにコピーします。
     + $ ssh-copy-id -i ~/.ssh/id\_rsa.pub posup@db3.ertl.jp

(posupはユーザ名、db3.ertl.jpはデータベースサーバ名)

* + その際にパスワードを聞かれるので適宜入力してください。

# ノードの作成

ここでは、Autowareで利用可能なノードを作成するための大まかな手順を示します。

基本的には、ROSのノード生成の手順と同じです。ROSのチュートリアルが参考になります。

* <http://wiki.ros.org/ja/ROS/Tutorials> (日本語)
* <http://wiki.ros.org/ROS/Tutorials> (英語)

## 開発の流れ

開発の流れは以下の通りです。

1. パッケージを作成する
2. ノードを作成する
3. ノードをビルドする
4. ノードの動作確認を行う

## パッケージの作成

ROSの場合、以下の条件を満たしたものが、パッケージとみなされます。

* パッケージのメタ情報が記述された設定ファイルpackage.xmlを直下に含む
* ビルドシステムCMake(<http://www.cmake.org/>)の設定ファイルCMakeLists.txtを直下に含む
* 1つのディレクトリに1つのパッケージのみ存在する (入れ子になっていない)

Autowareでは、上記に加え、以下の条件を満たすことが推奨されています。

* ros/src/sensing/fusionなどの各カテゴリにpackagesディレクトリがあり、その直下にパッケージを作成する
* nodesディレクトリにノード名と同じディレクトリを作成し、その下にノードのソースコードなどを配置する。
* msgディレクトリにメッセージファイルを配置する
* ノード等で共通の処理をライブラリ化している場合は、libやincludeなどのディレクトリに、ライブラリのソースコードを配置する

(注: 現在は、1つのノードをネームスペースによって使い分ける方法が推奨されています)

上記をまとめると、以下のようになります。

ros/src/カテゴリ..../packages/パッケージ名/

package.xml

CMakeLists.txt

nodes/

ノード名1/

ノード名1.cpp や ノード名1.py などのソースコード

ノード名2/

ノード名2.cpp や ノード名2.py などのソースコード

...

msg/

メッセージ名1.msg

メッセージ名1.msg

…

include/

ライブラリ等のヘッダファイル...

lib/

ライブラリのソースコード

パッケージを作成するには、catkin\_create\_pkgコマンドを使用します。

catkin\_create\_pkg パッケージ名 依存するパッケージ名...

Autowareでは、packagesディレクトリに移動して、以下のように実行します。

$ cd ros/src/カテゴリ/packags/

$ catkin\_create\_pkg mypkg roscpp std\_msgs

Successfully created files in /foo/ros/src/bar/packages/mypkg. Please adjust the values in package.xml.

$ ls mypkg/

CMakeLists.txt include/ package.xml src/

CMakeLists.txtおよびpackage.xmlを、適宜修正します。

CMakeLists.txtに関しては、ビルドの手順で述べます。

package.xmlに関しては、maintainerやlicense、description、versionなどを適切な値に変更してください。

rospackコマンドで、パッケージに関する情報を確認できます。第一引数にfind、第二引数にパッケージ名を指定して実行すると、パッケージの格納場所を確認できます。

$ rospack find roscpp

/opt/ros/indigo/share/roscpp

$ rospack find foo

[rospack] Error: package 'foo' not found

パッケージの依存関係は、depends1(直接依存)やdepends(間接依存)で確認できます。

$ rospack depends1 mypkg

roscpp

std\_msgs

$ rospack depends mypkg

cpp\_common

rostime

roscpp\_traits

roscpp\_serialization

…

## ノードの作成

ここでは、簡単なノードsimplenodeをC++で記述する例を示します。

(Pythonによる記述例は、前述のROSのチュートリアルに記載されています。)

まず、使用しないincludeとsrcディレクトリを削除し、新たにnodes/simplenodeディレクトリを作成します。

$ pwd

/foo/ros

$ pushd src/bar/packages/mypkg/

$ rm -rinclude

$ rmdir src

$ mkdir -p nodes/mynode

次に、ノードのソースコードnodes/mynode/mynode.cppを記述します。

以下に例を示します。

#include "ros/ros.h"

#include "std\_msgs/String.h"

#include "std\_msgs/Int32.h"

static ros::Publisher pub;

// 購読したトピックのコールバック関数

static void sub\_callback(const std\_msgs::String& smsg)

{

std\_msgs::Int32 pmsg;

// メッセージの作成

pmsg.data = smsg.data.size();

// トピックの配信

pub.publish(pmsg);

}

int main(int argc, char \*argv[])

{

// ROSの初期化

ros::init(argc, argv, "mynode");

ros::NodeHandle n;

// 配信するトピックの設定

pub = n.advertise<std\_msgs::Int32>("mypubval", 10);

// 購読するトピックの設定

ros::Subscriber sub = n.subscribe("mysubstr", 10, sub\_callback);

// メインループ

ros::spin();

return 0;

}

mysubstrという文字列型(std\_msgs::String)のトピックを購読し、その文字数(std\_msgs::Int32)をmypubvalというトピックで配信する、単純なノードです。

## ビルド

ビルドする前に、CMakeLists.txtを修正します。

まず、Autowareでは、C++のコンパイル時に特定のオプションを指定しています。そのため、CMakeLists.txtに以下の1行を追加します。

set(CMAKE\_CXX\_FLAGS "-std=c++0x -O2 -Wall ${CMAKE\_CXX\_FLAGS}")

mynodeのソースコードがnodes/mynode/mynode.cppであることを示す、以下の1行を追加します。

add\_executable(mynode nodes/mynode/mynode.cpp)

ROSの一般的なライブラリをリンク時に使用するよう、以下の1行を追加します。

target\_link\_libraries(mynode

${catkin\_LIBRARIES}

)

トップディレクトリに戻って、catkin\_make\_releaseスクリプトを実行します。

$ popd

$ pwd

/foo/ros

$ ./catkin\_make\_release

ただし、catkin\_make\_releaseスクリプトは、すでにビルド済の実行ファイルやライブラリをすべて削除し、最初からビルドを実行し直します。毎回このスクリプトを実行すると時間がかかるため、通常はcatkin\_makeコマンドでビルドを行います。

$ catkin\_make -DCMAKE\_BUILD\_TYPE=Release

## 動作確認

まず、ROSの基本的なプログラム(MasterやParameter Serverなど)を起動するため、roscoreコマンドを起動します。(あるいは、./run の起動でも構いません。)

$ . devel/setup.bash

$ roscore

次に、作成したmynodeを、rosrunコマンドで起動します。引数は、パッケージ名とノード名です。

先のroscore実行により、その擬似端末はroscoreに専有されるため、別の擬似端末から実行します。

$ . devel/setup.bash

$ rosrun mypkg mynode

引数にlistを指定してrosnodeコマンドを実行すると、実行中のノードを確認できます。

$ rosnode list

/mynode

/rosout

同様に、引数にlistを指定してrostopicコマンドを実行すると、配信もしくは購読されているトピックの一覧を確認できます。

$ rostopic list

/mypubval

/mysubstr

/rosout

/rosout\_agg

本来は他のノードとトピックを送受信しますが、ここでは代わりにコマンドを使用します。

mynodeはmypubvalトピックを配信するため、引数にechoを指定してrostopicコマンドを実行することで、トピックを購読します。

$ rostopic echo /mypubval

トピックの配信にもrostopicコマンドを使用します。引数には、pubと、配信するトピック名、トピックの型、そして値を指定します。

$ rostopic pub -1 /mysubstr std\_msgs/String "hello world"  
publishing and latching message for 3.0 seconds

$

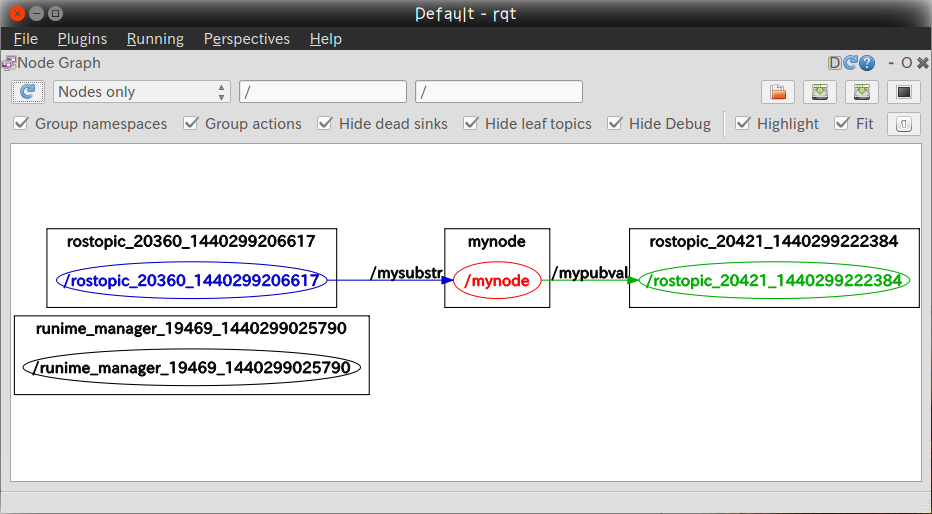
mynodeが”hello world”を購読し、文字数をmypubvalトピックに配信します。よって、先ほどのrostopic echo が以下を出力します。

$ rostopic echo /mypubval  
data: 11  
---

また、rqt\_graphを使うと、ノード間がどのトピックでつながっているかを、グラフで確認できます。

$ rosrun rqt\_graph rqt\_graph

Runtime ManagerのRQTボタンをクリックすると、rqtが実行されますので、rqt上でPlugins→Introspection→Node Graphを選択すると、rqt\_graphが表示されます。



実際のノードの開発で、トピックによる購読や配信がうまく行われない場合は、rostopicコマンドやrqt\_graphで、トピックのつながりなどを確認してください。

# 

# Runtime Manager

## 概要

Runtime Managerから起動・終了するROSノードを追加する方法、起動するROSノードへ与えるパラメータを設定する方法を示す。

## 追加・変更例

### Computingタブから起動・終了するROSノードの追加例

Computingタブに表示される各欄の項目は、次のパスの設定ファイルに記述されている。

ros/src/util/packages/runtime\_manager/scripts/computing.yaml

例えば、Localization/ndt\_localizer欄ndt\_matching項目の設定は、設定ファイル中の次の箇所に記述されている。

name : Computing

subs :

:

<略>

:

- name : ndt\_localizer

subs :

:

<略>

:

- name : ndt\_matching

cmd : roslaunch ndt\_localizer ndt\_matching.launch

param: ndt

ndt\_matching項目のチェックボックスをONにすると、サブプロセスを起動し、cmd行に記述されたコマンド"roslaunch ndt\_localizer ndt\_matching.launch"を実行し、ndt\_localizerパッケージのndt\_matching.launchスクリプトを起動する。

チェックボックスをOFFにすると、起動しているサブプロセスを終了し、起動しているndt\_maching.luanchスクリプトを終了させる。

Motion Planning欄直下の階層の末尾に、新たにExample欄を追加し、そこにTurtleSim項目を追加して、turtlesimパッケージのturtlesim\_nodeノードを起動・終了させる場合について、設定の追加例を示す。

name : Computing

subs :

:

<略>

:

- name : Motion Planning

subs :

- name : driver\_lanner

subs :

- name : obstacle\_avoidance

cmd : rosrun driving\_lanner obstacle\_avlidance

:

<略>

:

- name : car\_simulation

cmd : roslaunch waypoint\_follower car\_simulator.launch

param: car\_simulator

gui :

:

<略>

:

yaw:

depend : use\_pose

depend\_bool : ‘lambda v : v == “Initial Pos”’

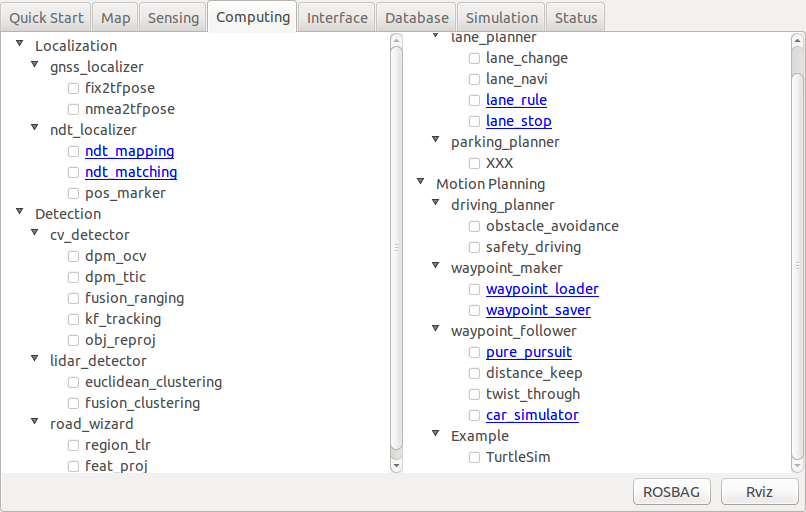
flags : [ no\_category, nl ]

- name : Example # この行を追加

subs : # この行を追加

- name : TurtleSim # この行を追加

cmd : rosrun turtlesim turtlesim\_node # この行を追加

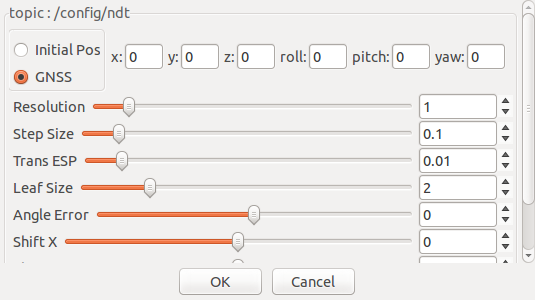


Computingタブ追加項目の表示

### 

### Computingタブから起動するROSノードへ与えるパラメータの設定例

例えば、Localization/ndt\_localizer欄ndt\_matching項目は、リンクが設定された状態で表示され、項目をクリックすると、パラメータを調整するダイアログが表示される。



パラメータを調整するダイアログ

この例では、パラメータの値を変更すると、パラメータはトピック /config/ndt として発行され、ndt\_matching.launchスクリプトから起動しているノードで購読される。

ダイアログに表示されるパラメータは、次のパスの設定ファイルに記述されている。

ros/src/util/packages/runtime\_manager/scripts/computing.yaml

Localization/ndt\_localizer欄ndt\_matching項目の設定は、設定ファイル中の次の箇所に記述されている。

name : Computing

subs :

:

<略>

:

- name : ndt\_localizer

subs :

:

<略>

:

- name : ndt\_matching

cmd : roslaunch ndt\_localizer ndt\_matching.launch

param: ndt

param行の ndt の記述は、パラメータ名が ndt であり、ダイアログに表示するパラメータの詳細が、後方のparams行以降にある "name : ndt" に記述されている事を表す。

params :

:

<略>

:

- name : ndt

topic : /config/ndt

msg : ConfigNdt

vars :

- name : init\_pos\_gnss

kind : radio\_box

choices:

- Initial Pos

- GNSS

v : 1

- name : x

label : 'x:'

v : 0.0

- name : y

label : 'y:'

v : 0.0

- name : z

label : 'z:'

v : 0.0

- name : roll

label : 'roll:'

v : 0.0

- name : pitch

label : 'pitch:'

v : 0.0

- name : yaw

label : 'yaw:'

v : 0.0

:

<略>

:

- name : shift\_y

label : Shift Y

min : -2.0

max : 2.0

v : 0

- name : shift\_z

label : Shift Z

min : -2.0

max : 2.0

v : 0

この設定例では、topic行に発行するトピック名、msg行にトピックで使用するメッセージ型名、vars行以下に、メッセージに含まれる各パラメータの設定が記述されている。

vars行以下の各パラメータの設定では、name行にメッセージ型のメンバ名、label行にダイアログで表示するラベル文字列、min行にパラメータの最小値、max行にパラメータの最大値、v行にパラメータの初期値が記述されている。

### パラメータ追加例

Motion Planning欄直下の階層の末尾に、新たにExample欄を追加し、そこにTurtleSim項目を追加した後、Int32型のパラメータを追加して、メッセージのパラメータをトピックとして発行する設定例を示す。

まず、設定ファイルにTrutleSim項目を追加する。

name : Computing

subs :

:

<略>

:

- name : Motion Planning

subs :

- name : driver\_lanner

subs :

- name : obstacle\_avoidance

cmd : rosrun driving\_lanner obstacle\_avlidance

:

<略>

:

- name : car\_simulation

cmd : roslaunch waypoint\_follower car\_simulator.launch

param: car\_simulator

gui :

:

<略>

:

yaw:

depend : use\_pose

depend\_bool : ‘lambda v : v == “Initial Pos”’

flags : [ no\_category, nl ]

- name : Example # この行を追加

subs : # この行を追加

- name : TurtleSim # この行を追加

cmd : rosrun turtlesim turtlesim\_node # この行を追加

次に、パラメータ名 example\_param を指定するparam行を追加する。

- name : Example

subs :

- name : TurtleSim

cmd : rosrun turtlesim turtlesim\_node

param: example\_param # この行を追加

さらに、後方のparams行以降に、example\_paramの詳細設定を追加する。

params :

:

<略>

:

- name : dispersion

label : Coefficient of Variation

min : 0.0

max : 5.0

v : 1.0

- name : example\_param # この行を追加

topic : /example\_topic # この行を追加

msg : Int32 # この行を追加

vars : # この行を追加

- name : data # この行を追加

label : Parameter # この行を追加

min : 0 # この行を追加

max : 100 # この行を追加

v : 50 # この行を追加

この例では、トピック名を /example、メッセージ型を Int32、メッセージ型 Int32 に含まれるメンバ data について、ダイアログに表示するラベル文字列を 'Parameter'、最小値を0、最大値を100、初期値を50に設定している。

メッセージ型 Int32 は、Runtime Mangerで使用してない型なので、Runtime Mananger のPythonスクリプト

(ros/src/util/packages/runtime\_manager/scripts/runtime\_manager\_dialog.py)

冒頭のinclude行の箇所に、メッセージ型 Int32 の include行を追加する。

:

<略>

:

from runtime\_manager.msg import accel\_cmd

from runtime\_manager.msg import steer\_cmd

from runtime\_manager.msg import brake\_cmd

from runtime\_manager.msg import traffic\_light

from std\_msgs.msg import Int32 # この行を追加

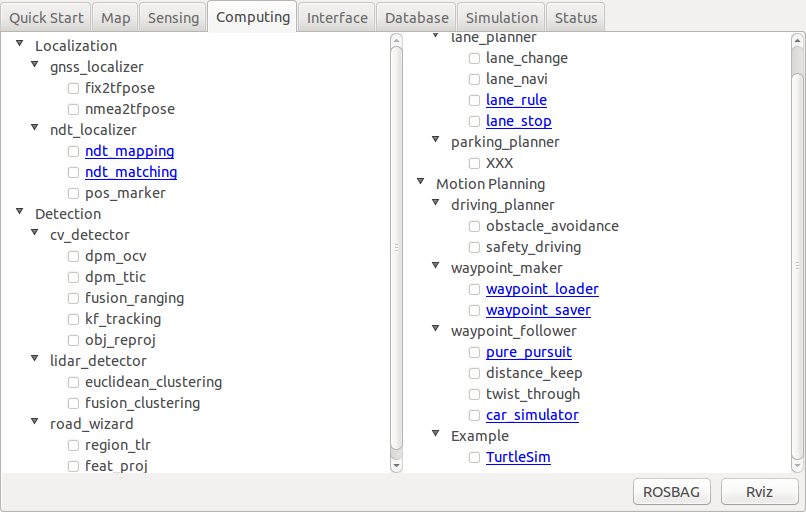
class MyFrame(rtmgr.MyFrame):

:

<略>

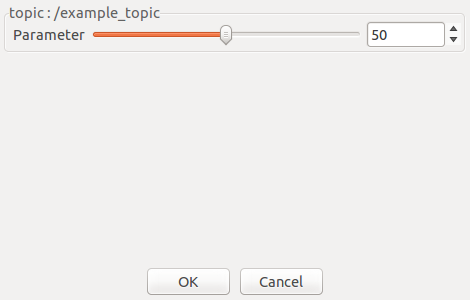
:

Runtime Mangerを起動すると、Computingタブに追加した項目が、リンク設定された状態で表示される。



Computingタブ追加項目のリンク設定表示

項目をクリックするとダイアログが表示される。



追加項目のパラメータ設定ダイアログ

トピックを表示するため、別端末で次のコマンドを実行する。

$ rostopic echo /example\_topic

ダイアログでパラメータを変更すると、発行トピックの内容が表示される。

data: 51

---

data: 52

---

data: 53

---

#### 小数値のパラメータおよびスライダー表示

パラメータ設定のmin行、max行、v行のいずれかが小数点を含む値の場合は、小数値のパラメータと解釈される。

また、パラメータの設定にmin行、max行の指定が無い場合は、最大値、最小値が判らないためスライダーは表示されない。

### ダイアログで設定したパラメータをrosparamパラメータとして設定する例

リンク設定からパラメータ設定ダイアログを開き、ファイルパス文字列を設定して、そのパス文字列をrosparamパラメータとして設定する例を示す。

先の例で追加したExample欄TrutleSim項目のパラメータexample\_paramに、ファイルパスの設定項目を追加する。

設定ファイルcomputing.yamlにファイルパスの設定を追加する。

:

<略>

:

- name : example\_param

topic : /example\_topic

msg : Int32

vars :

- name : data

label : Parameter

min : 0

max : 100

v : 50

- name : data\_file\_path # この行を追加

kind : path # この行を追加

v : /tmp/foo # この行を追加

rosparam : /example\_param/data\_path\_1 # この行を追加

name 行は、トピックとしてメッセージ出力する場合は、メッセージ中のメンバ名を指定する。

ここでは、ファイルパス文字列は、トピックのメッセージ中に存在せず、パス文字列をrosparamパラメータとして設定する例を示す。

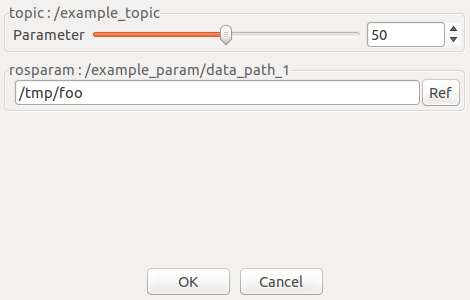
この場合、name行の指定はvars内の識別用として、他と重複しない任意の名前を指定すればよい。

kind行は、ファイルパス文字列を表す"path"を指定する。

v行は、デフォルト値のパスを指定する。

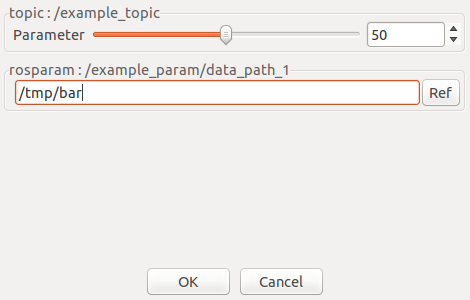
rosparam行は、rosparamパラメータの名前を指定する。

Runtime Managerを起動し、ComputingタブのTrutleSim項目のリンクをクリックするとダイアログが表示される。



ファイルパス設定を追加したパラメータ設定ダイアログ

Refボタンからファイルを選択したり、テキストボックスにパスを入力しENTERキーで設定すると、指定のrosparamパラメータに設定した値がセットされる。



パラメータ設定ダイアログでパスを入力

rosparamパラメータを表示するため、別端末で次のコマンドを実行する。

$ rosparam get /example\_param/data\_path\_1

/tmp/bar

$

Cancelボタンでダイアログを閉じた場合は、指定のrosparamパラメータの値が、ダイアログでを開いた時点の値に戻される。

Runtime Managerを終了すると、ダイアログで設定したパラメータの値は、パラメータ保存ファイルに保存される。

パラメータ保存ファイルros/src/util/packages/runtime\_manager/scripts/param.yaml

:

<略>

:

TurtleSim:

data: 50

data\_file\_path: /tmp/bar

:

<略>

:

#### ディレクトリを選択する場合

Refボタンでファイルではなく、ディレクトリを選択したい場合は、パラメータdata\_file\_pathの設定に、path\_type行でdir指定を追加する。

設定ファイルcomputing\_launch\_cmd.yaml

:

<略>

:

- name : example\_param

topic : /example\_topic

msg : Int32

vars :

- name : data

label : Parameter

min : 0

max : 100

v : 50

- name : data\_file\_path

kind : path

path\_type: dir # この行を追加

v : /tmp # 適宜変更

rosparam : /example\_param/data\_path\_1

Runtime Managerを再起動する際は、パラメータ保存ファイル中に保存されている、パス文字列の設定を削除してから起動する。

$ cd ros/src/util/packages/runtime\_manager/scripts

$ cp param.yaml param.yaml-

$ sed -e '/data\_file\_path:/d' param.yaml- > param.yaml

Runtime Managerを起動し、ComputingタブのTrutleSim項目のリンクをクリックするとダイアログが表示される。

Refボタンでディレクトリを選択するダイアログが表示されるようになる。

### パラメータをコマンドライン引数として出力する場合

設定したファイルパス文字列を、チェックボックスで起動するコマンドの、コマンドライン引数として与えたい場合の設定例を示す。

確認のため、TurtleSim項目の実行コマンドとして設定している文字列を、"echo"に変更しておく。

設定ファイルcomputing.yaml

:

<略>

:

- name : Example

subs :

- name : TurtleSim

#cmd : rosrun turtlesim turtlesim\_node # 変更

cmd : echo # 変更

param: example\_param

パラメータexample\_paramに、新たなファイルパス設定を追加し、コマンドライン引数として出力するよう設定する。

設定ファイルcomputing\_launch\_cmd.yaml

:

<略>

:

- name : example\_param

topic : /example\_topic

msg : Int32

vars :

- name : data

label : Parameter

min : 0

max : 100

v : 50

- name : data\_file\_path

kind : path

path\_type: dir

v : /tmp

rosparam : /example\_param/data\_path\_1

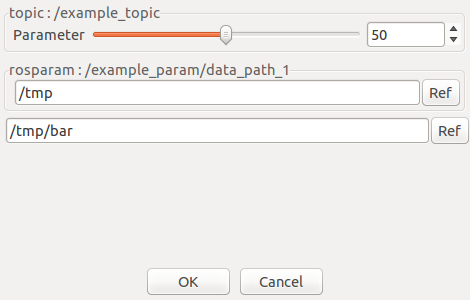
- name : data\_file\_path\_2 # 追加

kind : path # 追加

v : /tmp/bar # 追加

cmd\_param: # 追加

delim : '' # 追加



パラメータ設定ダイアログ

OKボタンでダイアログを閉じ、TurtleSim項目のチェックボックスをONにすると、Runtime Managerを起動した端末に、次の表示が出る。

['echo', '/tmp/bar']

/tmp/bar

cmd行に設定したechoコマンドの引数として、ファイルパスが指定されて実行される。

#### cmd\_param行の設定

cmd\_param行の設定として、dash行、var\_name行、delim行を指定することが出来る。

実行コマンドとコマンドライン引数は、次の並びに配置される。

<cmd行の値><空白><dash行の値><var\_name行の値><delim行の値><パラメータの値>

dash行が存在しない場合は、<dash行の値><var\_name行の値> の部分を出力しない。

delim行が存在しない場合は、<delim行の値><パラメータの値> の部分を出力しない。

var\_name行が存在しない場合は、デフォルトとしてname行の値が使われる。

dash行で ''(空文字列)を指定した場合は、<dash行の値><var\_name行の値> の部分は <var\_name行の値> だけになる。

delim行で ''(空文字列)を指定した場合は、<delim行の値><パラメータの値> の部分は <パラメータの値> だけになる。

設定例

cmd\_param:

dash : '--'

delim : '='

コマンドラインの配置

echo --data\_file\_path\_2=/tmp/bar

cmd\_param:

dash : '-'

var\_name : f

delim : ' '　　　　　# 1つの空白文字

コマンドラインの配置

echo -f /tmp/bar

cmd\_param:

dash : '' # 空文字

delim : ':='

コマンドラインの配置

echo data\_file\_path\_2:=/tmp/bar

cmd\_param:

delim : '' # 空文字

コマンドラインの配置

echo /tmp/bar

cmd\_param:

dash : '--'

var\_name : '' # 空文字

delim : '' # 空文字

コマンドラインの配置

echo --/tmp/bar

### パラメータ設定のその他のkind行指定

kind行でチェックボックス、トグルボタン、ラジオボックス、メニューを指定出来る。

チェックボックス、トグルボタンではBOOL値(True/False)を扱い、ラジオボックス(複数のラジオボタンをまとめた部品)、メニューでは選択されている項目のインデックス値(0から項目数-1までの整数値)を扱う。

設定ファイルcomputing.yaml

:

<略>

:

- name : example\_param

topic : /example\_topic

msg : Int32

vars :

- name : data

label : Parameter

min : 0

max : 100

v : 500

- name : data\_file\_path

kind : path

path\_type: dir

v : /tmp

rosparam : /example\_param/data\_path\_1

- name : data\_file\_path\_2

kind : path

v : /tmp/bar

cmd\_param:

delim : ''

- name : sw\_1 # 追加

label : Enable # 追加

kind : checkbox # 追加

v : True # 追加

- name : sw\_2 # 追加

label : Alert # 追加

kind : toggle\_button # 追加

v : False # 追加

- name : sel\_1 # 追加

kind : radio\_box # 追加

label : 'Edit:' # 追加

choices : [ cut, copy, paste ] # 追加

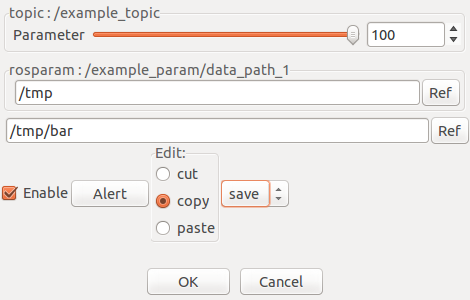
v : 1 # 追加

- name : sel\_2 # 追加

kind : menu # 追加

choices : [ open, close, save, load ] # 追加

v : 2 # 追加



部品が追加されたパラメータ設定ダイアログ

この例では、ダイアログにパラメータの部品が追加されるだけで、パラメータの値は出力されない。

追加したパラメータを出力するには、次の3つの方法がある。

* name行を、トピックのメッセージに含まれるメンバ名に設定すれば、トピックとして出力される。
* rosparam行でrosparamパラメータ名を指定すれば、rosparamパラメータとしてセットされる。
* cmd\_param行を指定すれば、項目のチェックボックスONでコマンドを起動する際に、コマンド引数として指定される。

### Quick Startタブのボタンで起動・終了するコマンドの設定例

Quick Startタブの最下行のトグルボタンで起動・終了するコマンドは、次のパスの設定ファイルに記述されている。

ros/src/util/packages/runtime\_manager/scripts/qs.yaml

例えば、Sensingトグルボタンの設定は、設定ファイル中の次の箇所に記述されている。

buttons:

:

<略>

:

sensing\_qs:

run : roslaunch

param : sensing\_qs

:

<略>

:

SensingトグルボタンをONにすると、サブプロセスを起動し、run行に記述されたコマンド"roslunch"をparam行で指定された内容のコマンドライン引数(.launchファイル)を与えて実行する。

トグルボタンをOFFにすると、起動しているサブプロセスを終了し、実行しているコマンド(roslaunchコマンド)を終了させる。

param行の sensing\_qs の記述は、パラメータ名が sensing\_sq であり、コマンドライン引数として追加するパラメータの詳細が、後方のparams行以降にある "name : sensing\_sq"に記述されている事を表す。

params :

:

<略>

:

- name : sensing\_sq

vars :

- name : file

kind : path

v : ''

cmd\_param :

delim : ''

must : True

:

<略>

:

この設定例では、vars行以下にコマンドライン引数として追加するパラメータの設定が記述されている。

vars行以下のパラメータ設定では、name行でvars内の識別用の名前としてfileを指定し、kind行で、ファイルパス文字列を表す"path"を指定し、v行で、値のパスとして""(空文字列)を指定し、cmd\_param行以下の設定で、上記のパス文字列をコマンドライン引数として指定する際の形式を指定している。

この設定例の形式では、v行の値のパス文字列のみをコマンドライン引数として与えるように指定している。

cmd\_param行の設定の詳細は「パラメータをコマンドライン引数として出力する場合」「cmd\_param行の設定」を参照。

また、kind行として"patth"が指定されている場合は、Runtime Managerスクリプト内で、v行の値をパス文字列として扱い、絶対パスに変換してからコマンドライン引数として配置している。

Sengingテキストボックスにパス文字列”~/.autoware/launch\_files/sensing.launch”を入力し、

SensingトグルボタンをONにすると、次のコマンドが実行される。

roslaunch (ユーザのホームディレクトリの絶対パス)/.autoware/launch\_files/sensing.launch

例えば、設定ファイルの内容を次のように変更すると、実行するコマンドのコマンドライン引数を確認できる。

buttons:

:

<略>

:

sensing\_qs:

#run : roslaunch # この行を変更

run : echo # この行を追加

param : sensing\_qs

:

<略>

:

params :

:

<略>

:

- name : sensing\_qs

vars :

- name : file

kind : path

#v : ''# この行を変更

v : /tmp/foo # この行を追加

cmd\_param :

dash : '--' # この行を追加

delim : '=' # この行を変更

must : True

- name : xval # この行以降を追加

v : 12.3

cmd\_param :

dash : '-'

delim : ' '

:

<略>

:

Runtime Manager終了後、パラメータ保存ファイルに保存されている変更前のパスを削除する。

パラメータ保存ファイル ros/src/util/packages/runtime\_manager/scripts/param.yaml

:

<略>

:

sensing\_qs:

file: ~/.autoware/launch\_files/sensor.launch

:

<略>

:

上記のsensor行とfile行の2行を削除する。

Runtime Managerを起動しなおし、SensingトグルボタンをONにすると、Runtime Mangerを起動した端末に、次の表示が出る。

['echo', '--file=/tmp/foo', '-xval', '12.3']

--file=/tmp/foo -xval 12.3

### Sensingタブのボタンで起動・終了するコマンドの設定例

Sensingタブ右側に配置されたトグルボタンで起動・終了するコマンドは、次のパスの設定ファイルに記述されている。

ros/src/util/packages/runtime\_manager/scripts/sensing.yaml

例えば、Points Imageトグルボタンの設定は、設定ファイル中の次の箇所に記述されている。

:

<略>

:

buttons:

:

<略>

:

points\_image :

run : rosrun points2image points2image

:

<略>

:

Points ImageトグルボタンをONにすると、サブプロセスを起動し、run行に記述されたコマンド

"rosrun points2image points2image"を実行する。

トグルボタンをOFFにすると、起動しているサブプロセスを終了し、実行しているコマンド(rosrunコマンド)を終了させる。

例えば、設定ファイルの内容を次のように変更して、コマンド実行の様子を確認できる。

:

<略>

:

buttons:

:

<略>

:

points\_image :

#run : rosrun points2image points2image # この行を変更

run : echo hello # この行を追加

:

<略>

:

Runtime Managerを起動しSensingタブの画面を選択し、Points ImageトグルボタンをONにすると、Runtime Mangerを起動した端末に、次の表示が出る。

['echo', 'hello']

hello

//