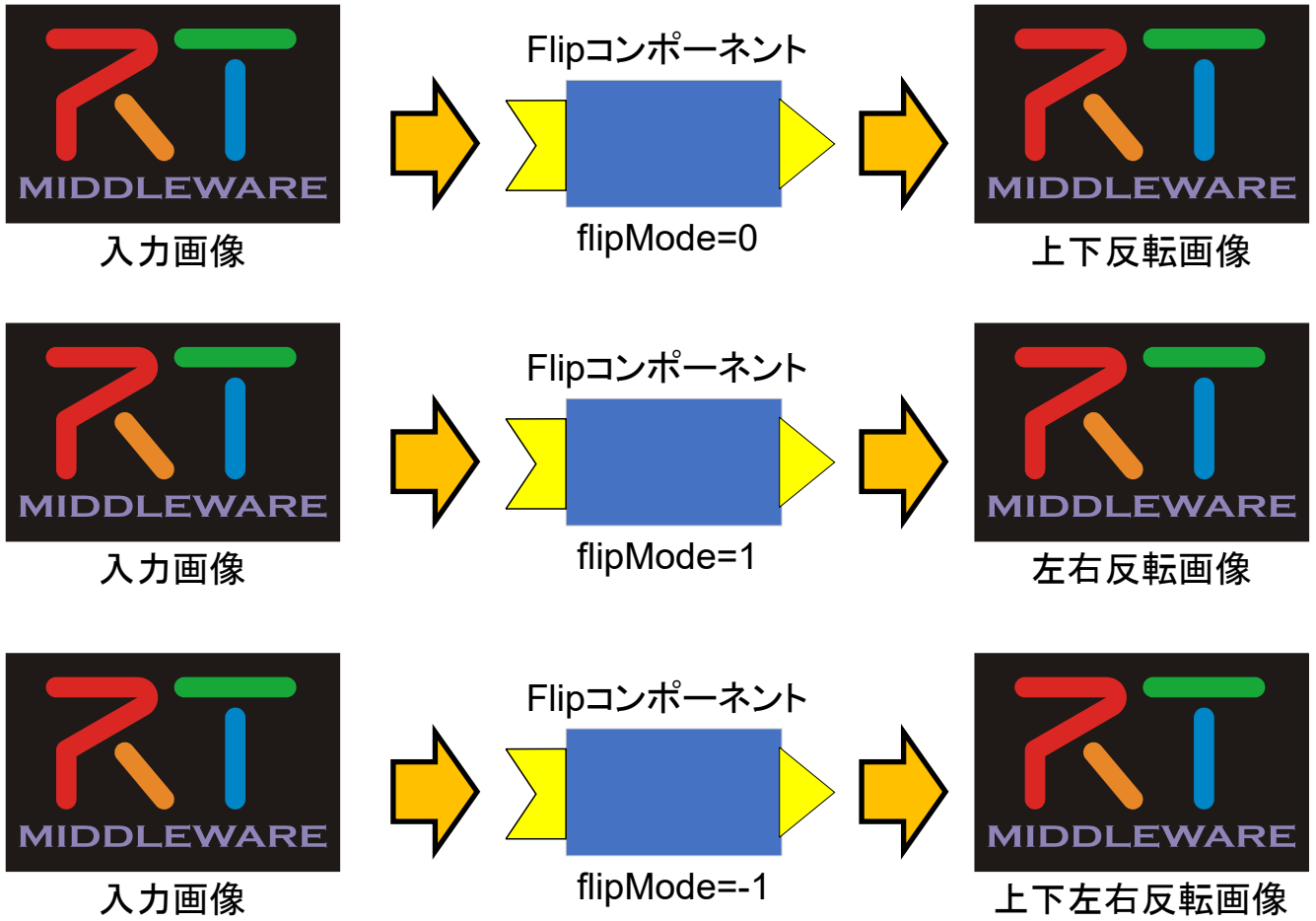
RTコンポーネント作成入門

宮本 信彦
国立研究開発法人産業技術総合研究所
インダストリアルCPS研究センター
ソフトウェアプラットフォーム研究チーム

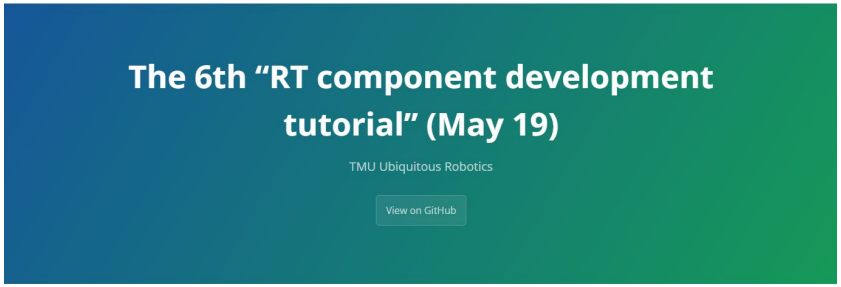
実習内容

- 入力画像を反転して出力するFlipコンポーネントを作成する。



実習内容

- 資料
  - https://sealbreeder.github.io/TMU-Ubiquitous-Robotics/220518



The 6th “RT component development tutorial” (May 19)

第6回「RTコンポーネントの開発実習」（5月18日）

  - https://openrtm.org/openrtm/ja/node/7151

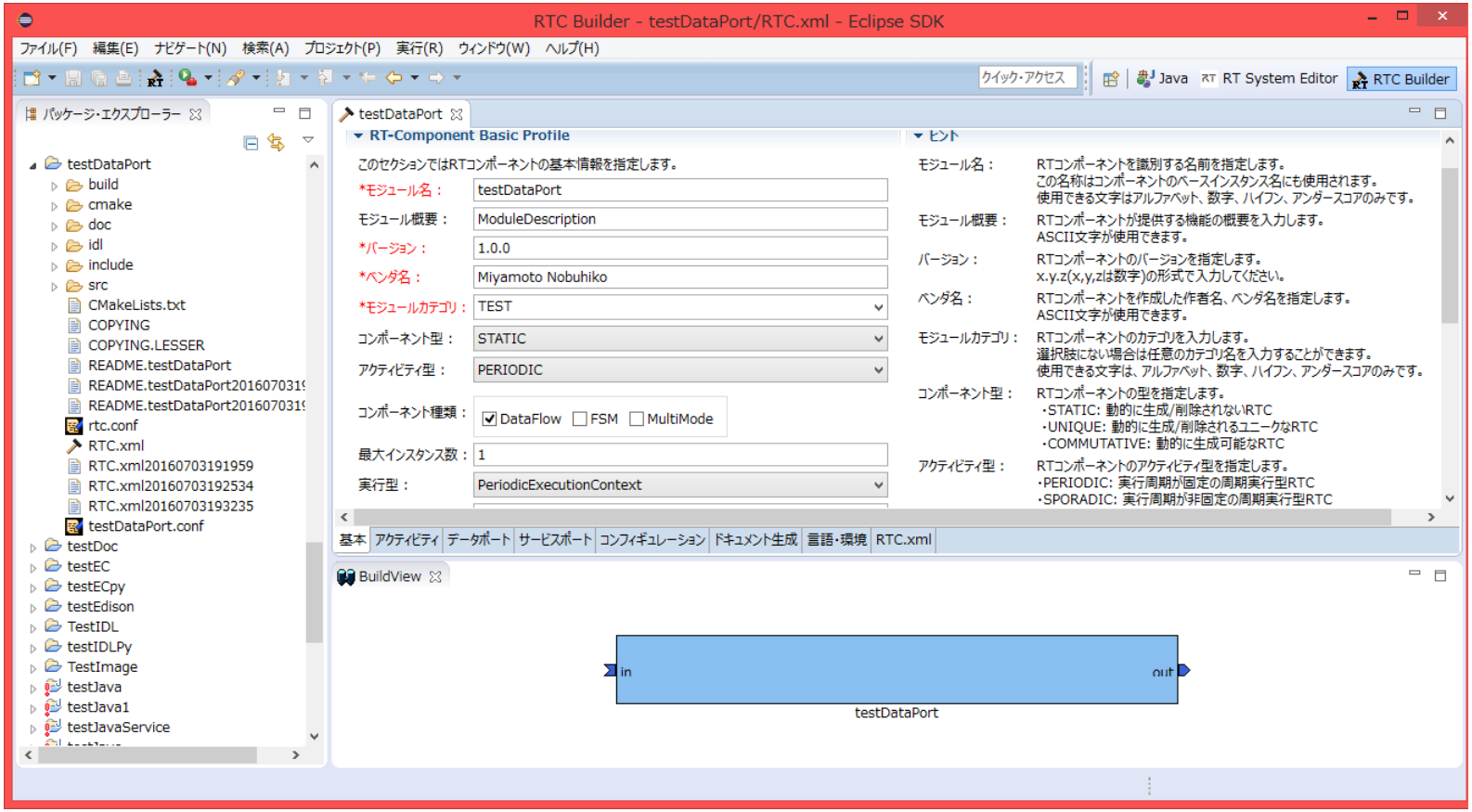
全体の手順

- RTC Builderによるソースコード等のひな型の作成
- ソースコードの編集、ビルド
  - ビルドに必要な各種ファイルを生成
    - CMakeにより各種ファイル生成
  - ソースコードの編集
    - Flip.h、Flip.cppの編集
  - ビルド
    - Visual Studio、Code::Blocks
- RTシステムエディタによるRTシステム作成、動作確認
  - RTシステム作成
    - データポート接続、コンフィギュレーションパラメータ設定

コンポーネント開発ツール RTC Builderについて

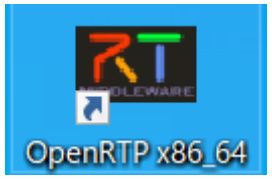
RTC Builder

- コンポーネントのプロファイル情報を入力し，ソースコード等のひな型を生成するツール
  - C++、Python、Java、Luaのソースコードを出力



RTC Builderの起動

- 起動する手順
  - Windows(OpenRTM-aist 1.2、2.0)
    - デスクトップのショートカットをダブルクリック



  - デスクトップのショートカットがない場合
    - Windows 7
      - 「スタート」→「すべてのプログラム」→「OpenRTM-aist 1.2.0」→「Tools」→「OpenRTP」
    - Windows 8.1
      - 「スタート」→「アプリビュー(右下矢印)」→「OpenRTM-aist 1.2.0」→「OpenRTP」
      - ※同じフォルダに「RTSystemEditorRCP」がありますが、これはRTC Builderが使えないので今回は「OpenRTP」を起動してください。
    - Windows 10
      - 左下の「ここに入力して検索」にOpenRTPと入力して、表示されたOpenRTPを起動
  - Ubuntu
    - 以下のコマンドを入力
    - $ openrtp

RTC Builderの起動

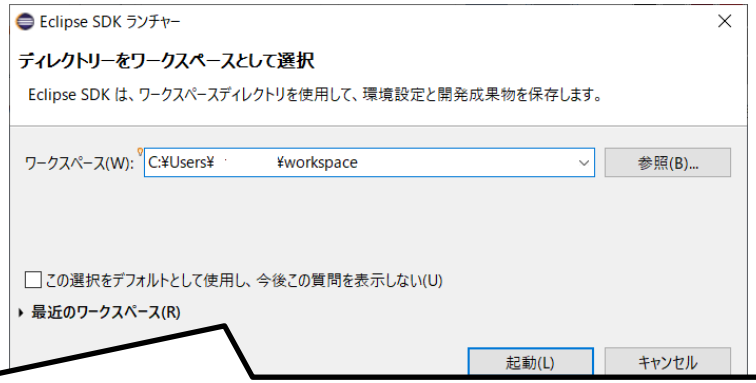
- Windows 8.1

RTC Builderの起動

- Windows 10

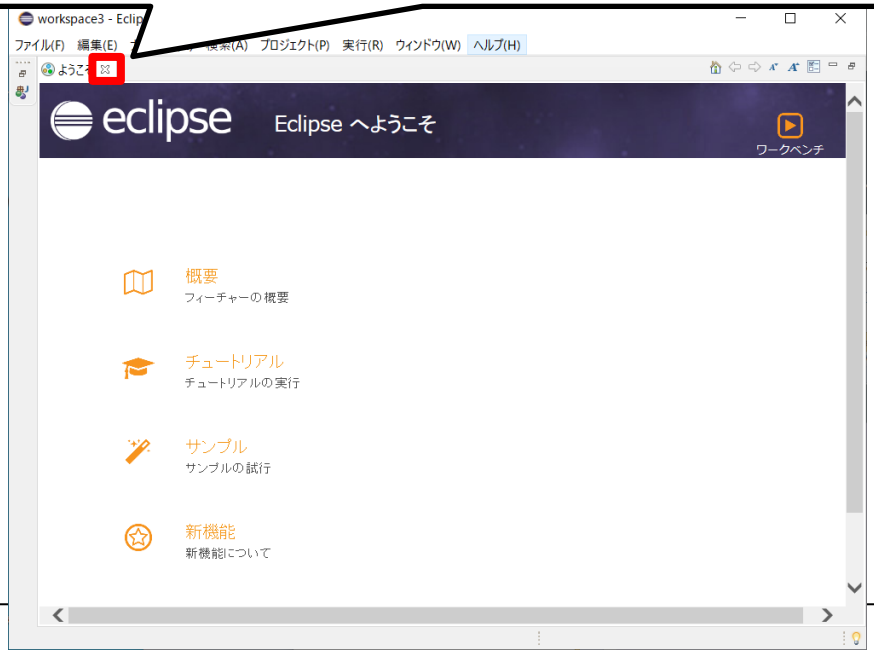
左下の「ここに入力して検索」に「OpenRTP」と入力

RTC Builderの起動



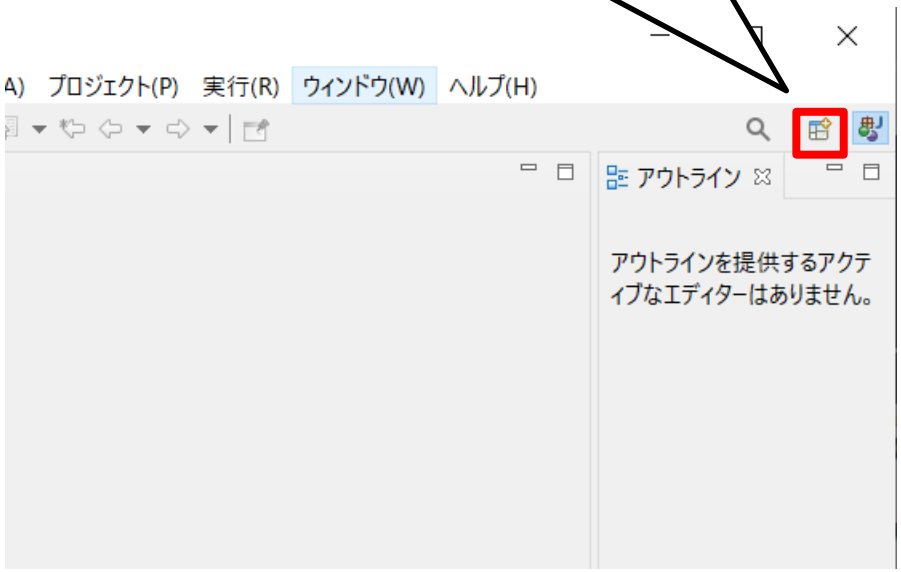
ワークスペースに適当な場所を指定して起動をクリックする

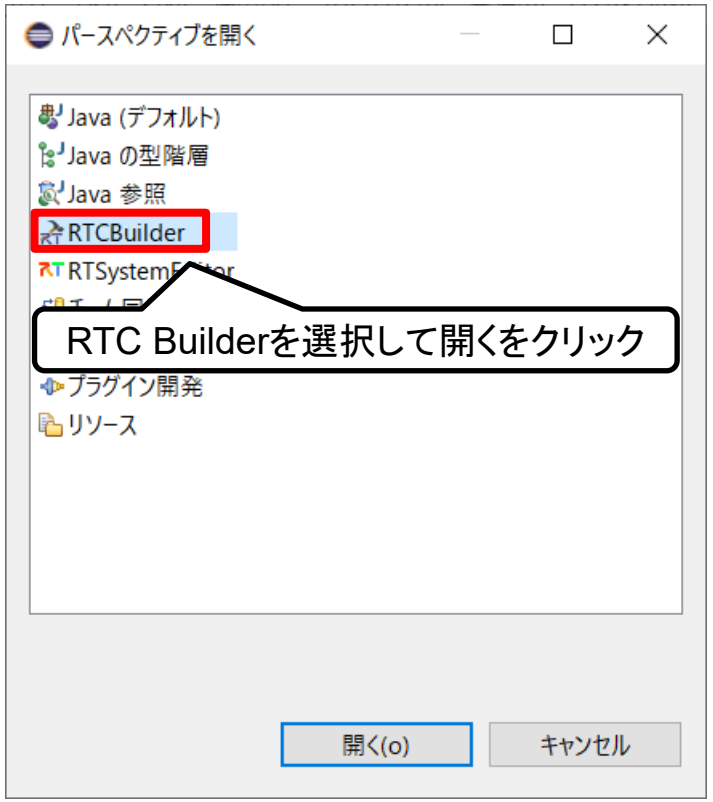
最初に起動したときはwelcomeページが開くため×を押して閉じる



RTC Builderの起動

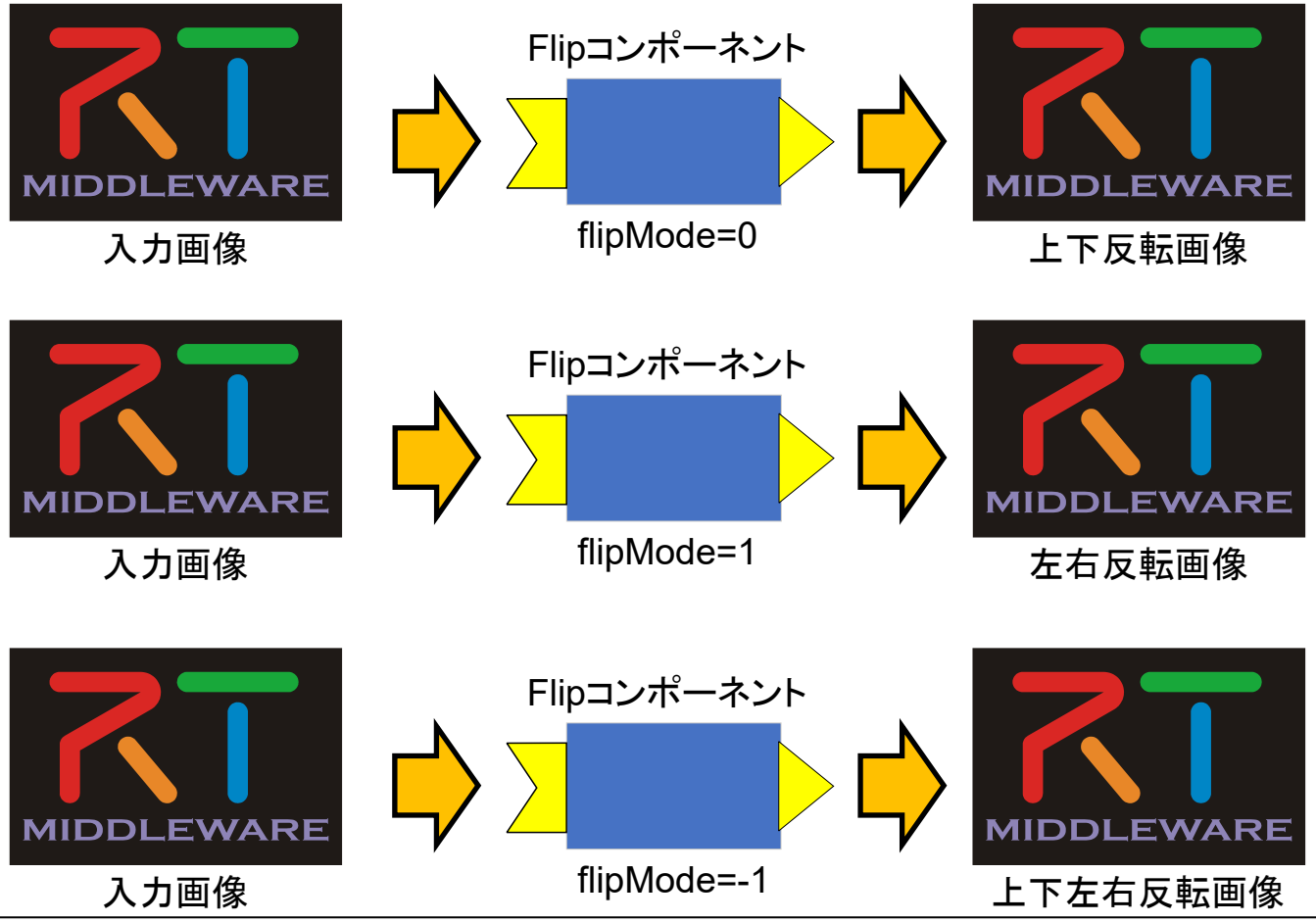
右上の「パースペクティブを開く」ボタンをクリック



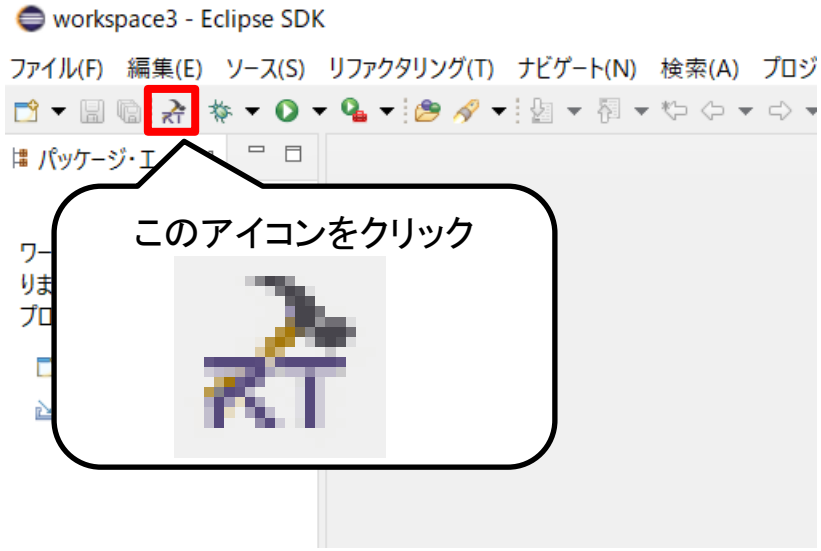


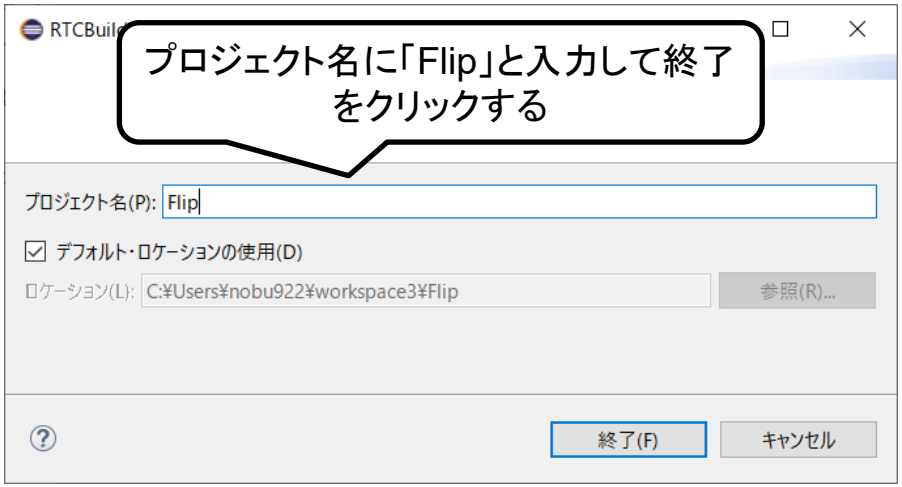
プロジェクト作成

- Flipコンポーネントのスケルトンコードを作成する。
  - CameraImage型のInPort、OutPort
  - 画像反転処理



プロジェクト作成

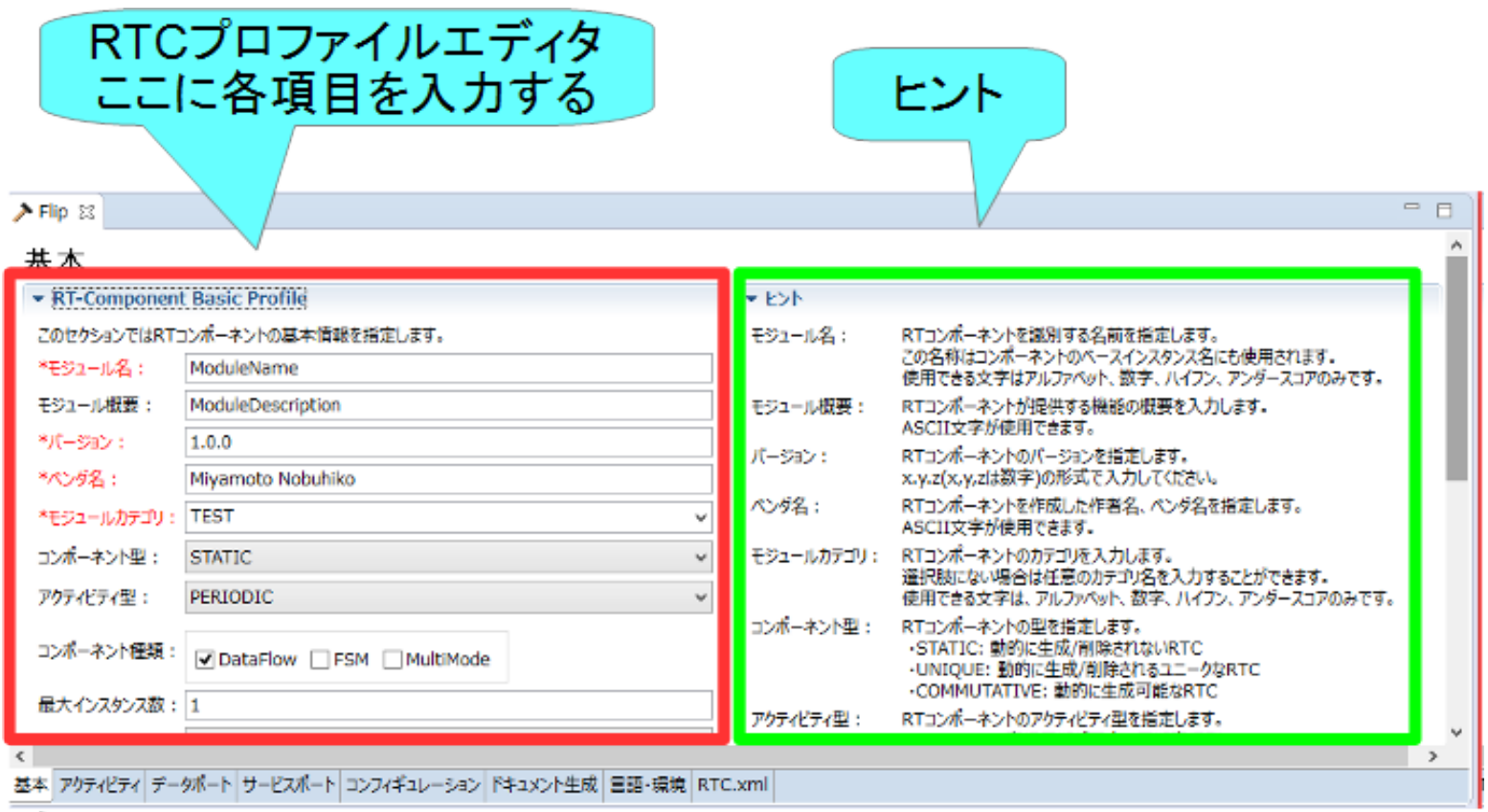




- Eclipse起動時にワークスペースに指定したディレクトリに「Flip」というフォルダが作成される
  - この時点では「RTC.xml」と「.project」のみが生成されている
- 以下の項目が設定する
  - 基本プロファイル
  - アクティビティ・プロファイル
  - データポート・プロファイル
  - サービスポート・プロファイル
  - コンフィギュレーション
  - ドキュメント
  - 言語環境
  - RTC.xml

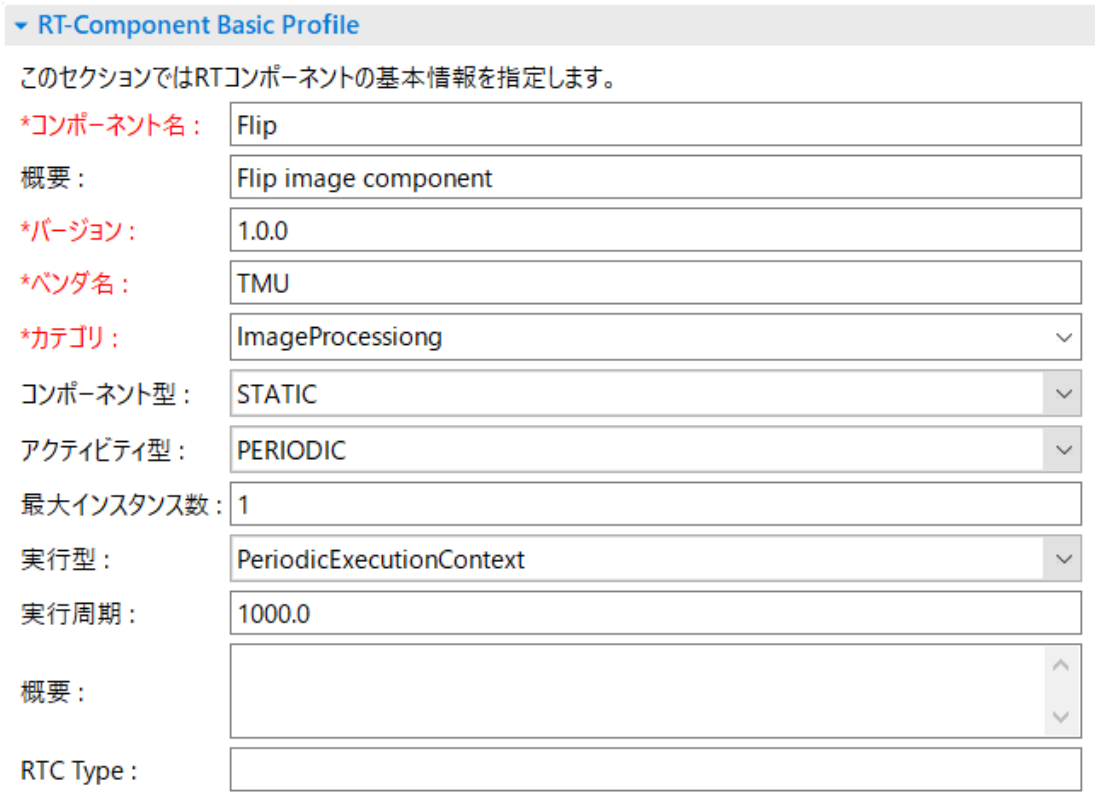
基本プロファイルの入力

- RTコンポーネントのプロファイル情報など，コンポーネントの基本情報を設定．
- コード生成，インポート/エクスポート，パッケージング処理を実行



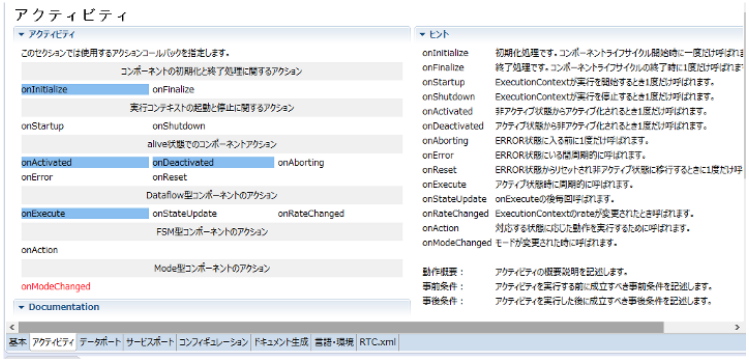
基本プロファイルの入力

- **コンポーネント名**
  - **Flip**
- モジュール概要
  - 任意(Flip image component)
- バージョン
  - 任意(1.0.0)
- ベンダ名
  - 任意
- モジュールカテゴリ
  - 任意(ImageProcessing)
- コンポーネント型
  - STATIC
- アクティビティ型
  - PERIODIC
- コンポーネントの種類
  - DataFlow
- 最大インスタンス数
  - 1
- 実行型
  - PeriodicExecutionContext
- 実行周期
  - 1000.0
- 概要
  - 任意



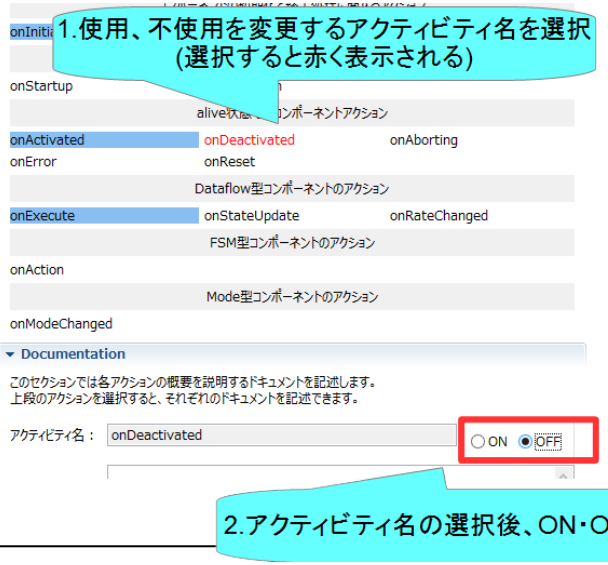
アクティビティの設定

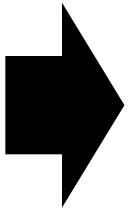
- 使用するアクティビティを設定する

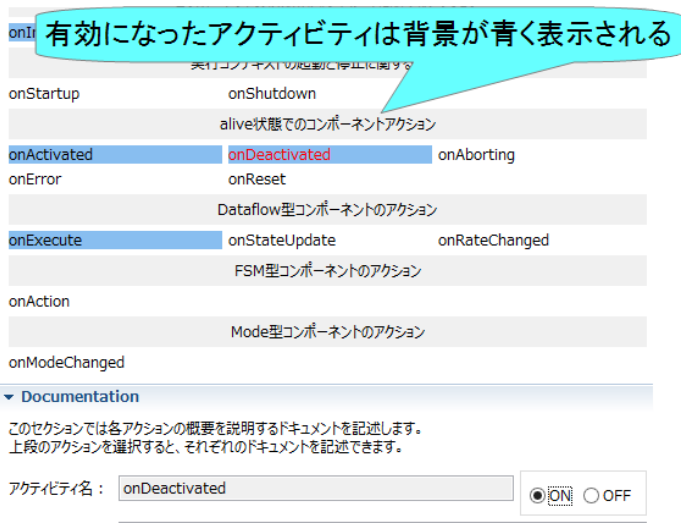


「アクティビティ」タブを選択

- 指定アクティビティを有効にする手順





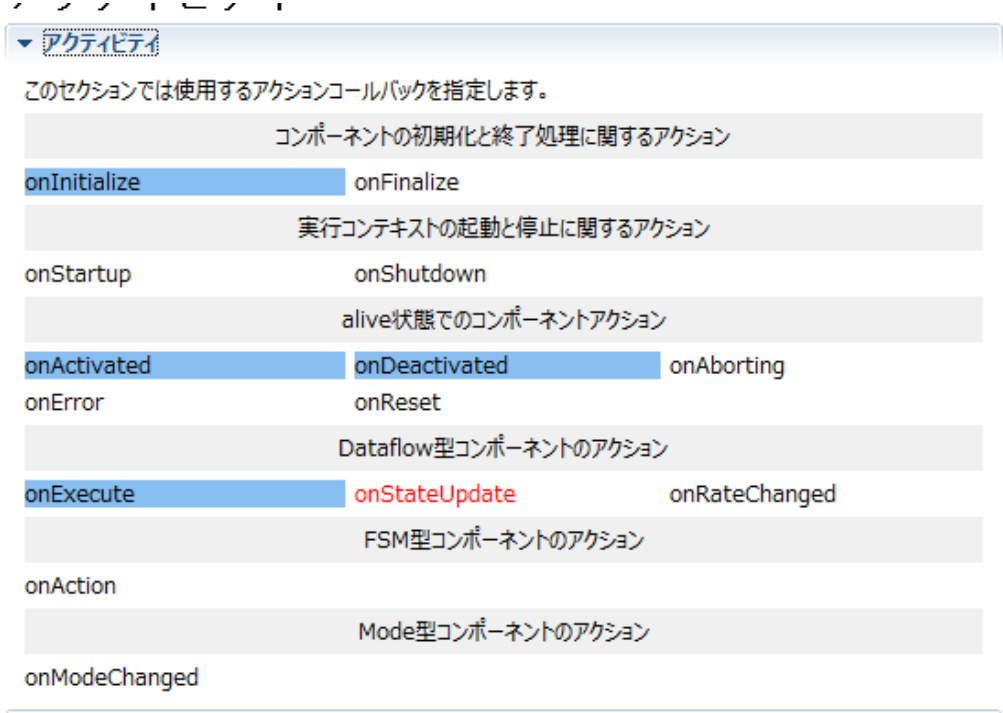


アクティビティの設定

コールバック関数	処理
onInitialize	初期化処理
onActivated	アクティブ化されるとき1度だけ呼ばれる
onExecute	アクティブ状態時に周期的に呼ばれる
onDeactivated	非アクティブ化されるとき1度だけ呼ばれる
onAborting	ERROR状態に入る前に1度だけ呼ばれる
onReset	resetされる時に1度だけ呼ばれる
onError	ERROR状態のときに周期的に呼ばれる
onFinalize	終了時に1度だけ呼ばれる
onStateUpdate	onExecuteの後毎回呼ばれる
onRateChanged	ExecutionContextのrateが変更されたとき1度だけ呼ばれる
onStartup	ExecutionContextが実行を開始するとき1度だけ呼ばれる
onShutdown	ExecutionContextが実行を停止するとき1度だけ呼ばれる

アクティビティの設定

- 以下のアクティビティを有効にする
  - onInitialize
  - **onActivated**
  - **onDeactivated**
  - **onExecute**
- 今回は練習のため、Documentationは空白でも大丈夫です

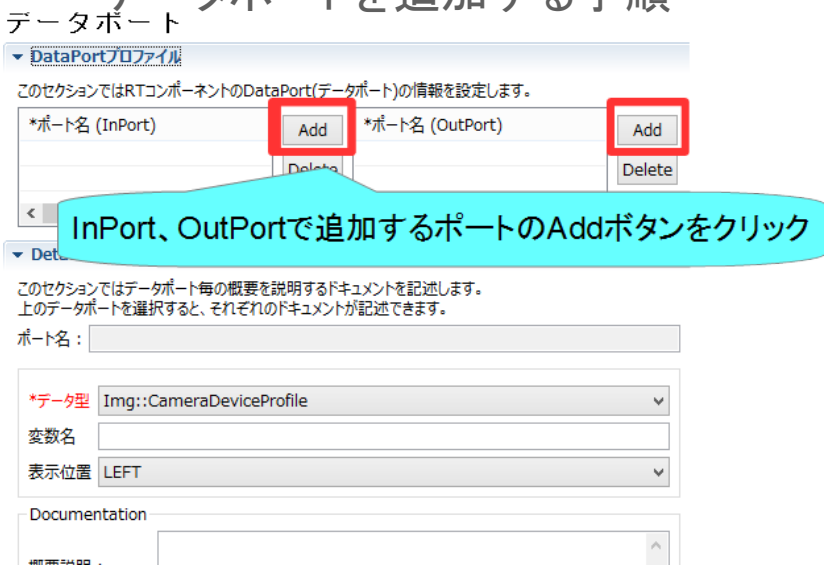


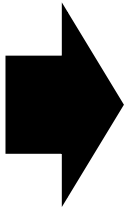
データポートの設定

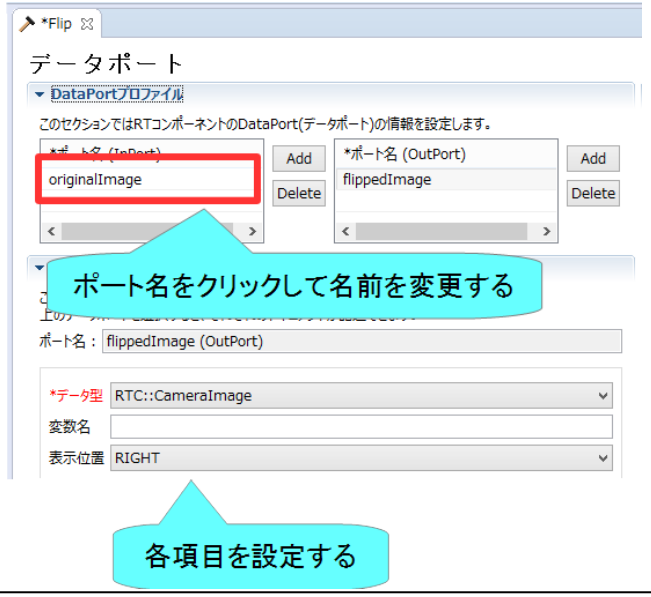
- InPort、OutPortの追加、設定を行う

「データポート」タブを選択

- データポートを追加する手順

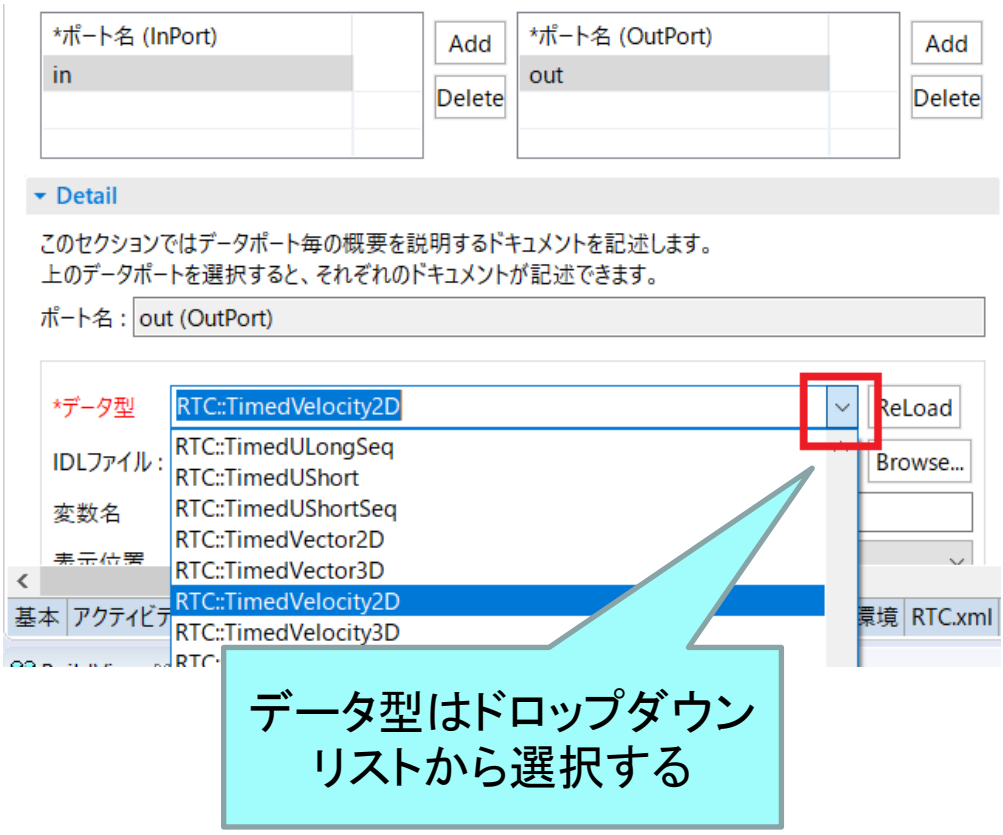






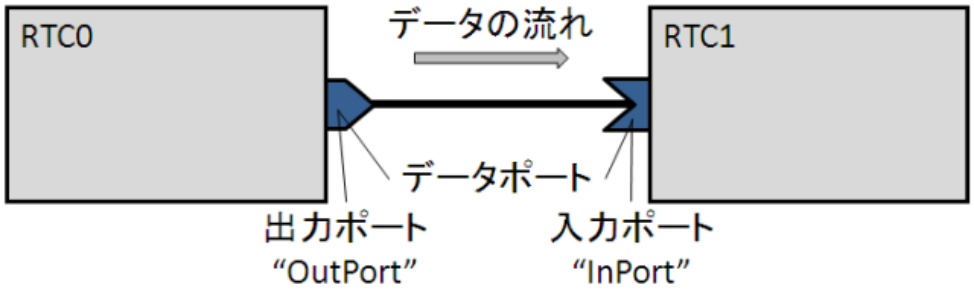
データポートの設定

- 以下のInPortを設定する
  - **originalImage**
    - データ型：
      RTC::CameraImage
    - 他の項目は任意
    - ※Img::CameraImage型と間違えないようにしてください。
- 以下のOutPortを設定する
  - **flippedImage**
    - データ型：
      RTC::CameraImage
    - 他の項目は任意

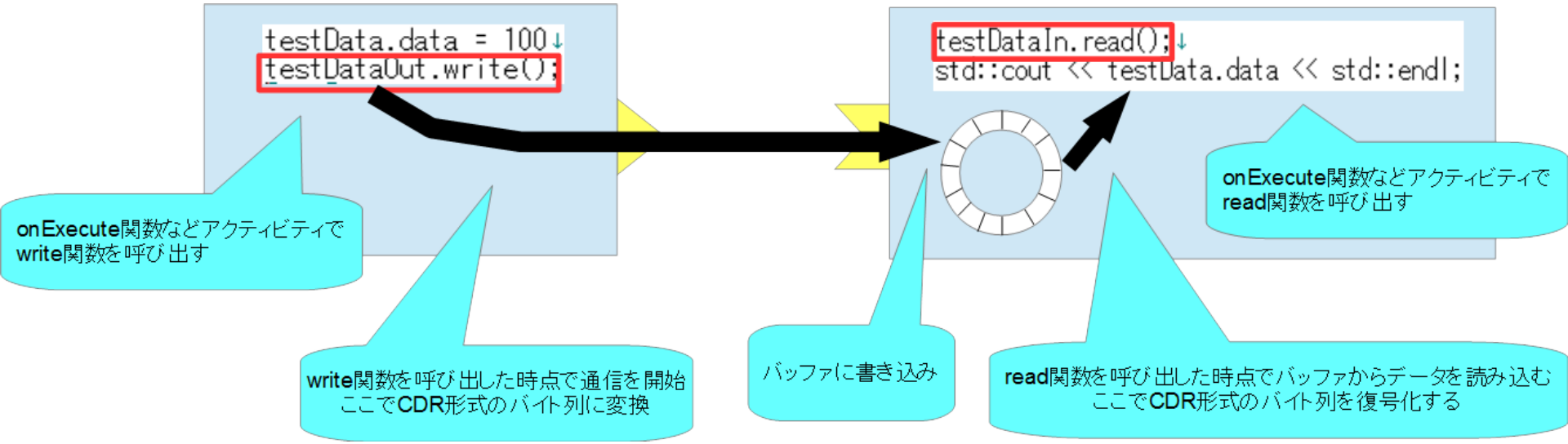


データポートについて

- 連続したデータを通信するためのポート

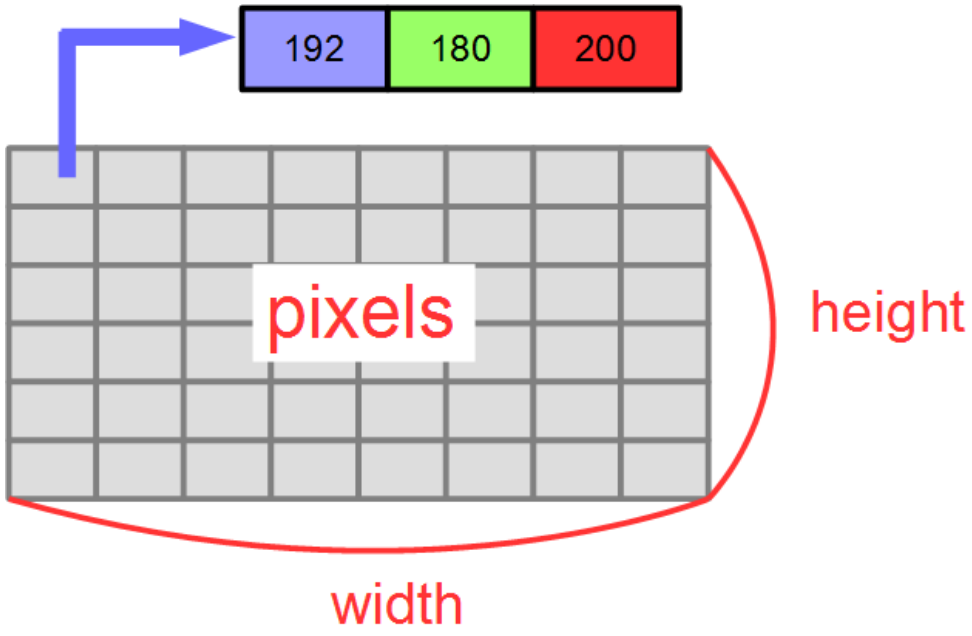


- 以下の例はデータフロー型がpush、サブスクリプション型がflush、インターフェース型がcorba_cdrの場合



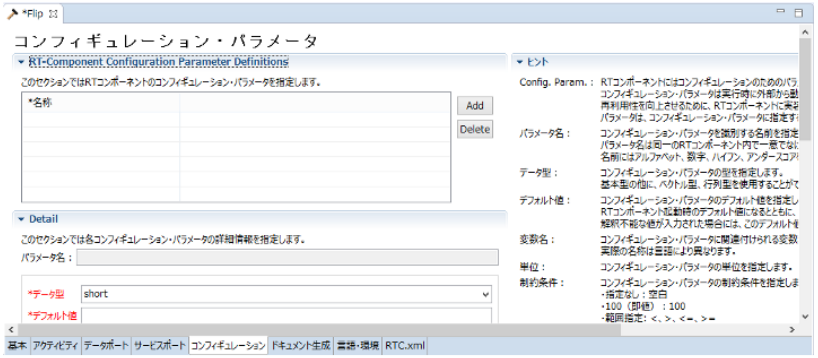
RTC::Camera型について

- InterfaceDataTypes.idlで定義されている**画像データ**を表現するためのデータ型
  - **with**:画像の幅
  - **height**:画像の高さ
  - **bpp**:色深度
  - **format**:フォーマット名(jpeg、png)
  - **fDiv**:スケールファクタ
  - **pixels**:画像データ



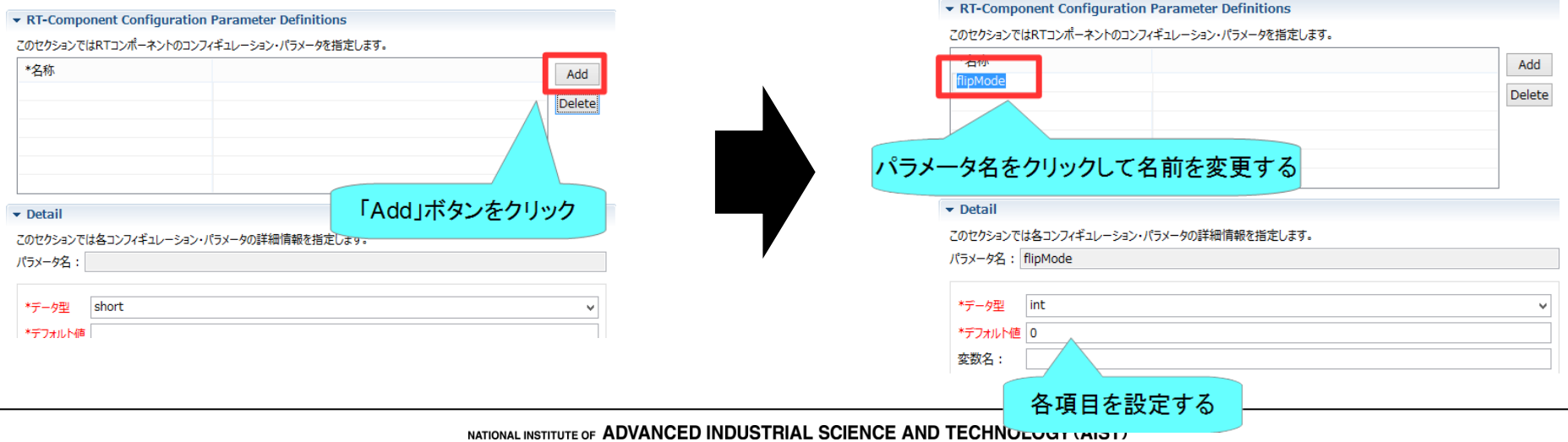
コンフィギュレーションの設定

- コンフィギュレーションパラメータの追加、設定を行う



「コンフィギュレーション」タブを選択

- コンフィギュレーションパラメータを追加する手順



コンフィギュレーションの設定

- 以下のコンフィギュレーションパラメータを設定する
  - **flipMode**
    - データ型：int
    - デフォルト値： 0
    - 制約条件：(0, -1, 1)
    - Widget：radio
    - 他の項目は任意

GUI(ラジオボタン)にで反転方向の操作ができるようにする

コンフィギュレーションパラメータの制約、Widgetの設定

- RT System Editorでコンフィギュレーションパラメータを編集する際にGUIを表示する

- Widget：text

- 制約条件：0<=x<=100
- Widget：spin
- Step：10

- 制約条件：0<=x<=100
- Widget：slider
- Step：10

コンフィギュレーションパラメータの制約、Widgetの設定

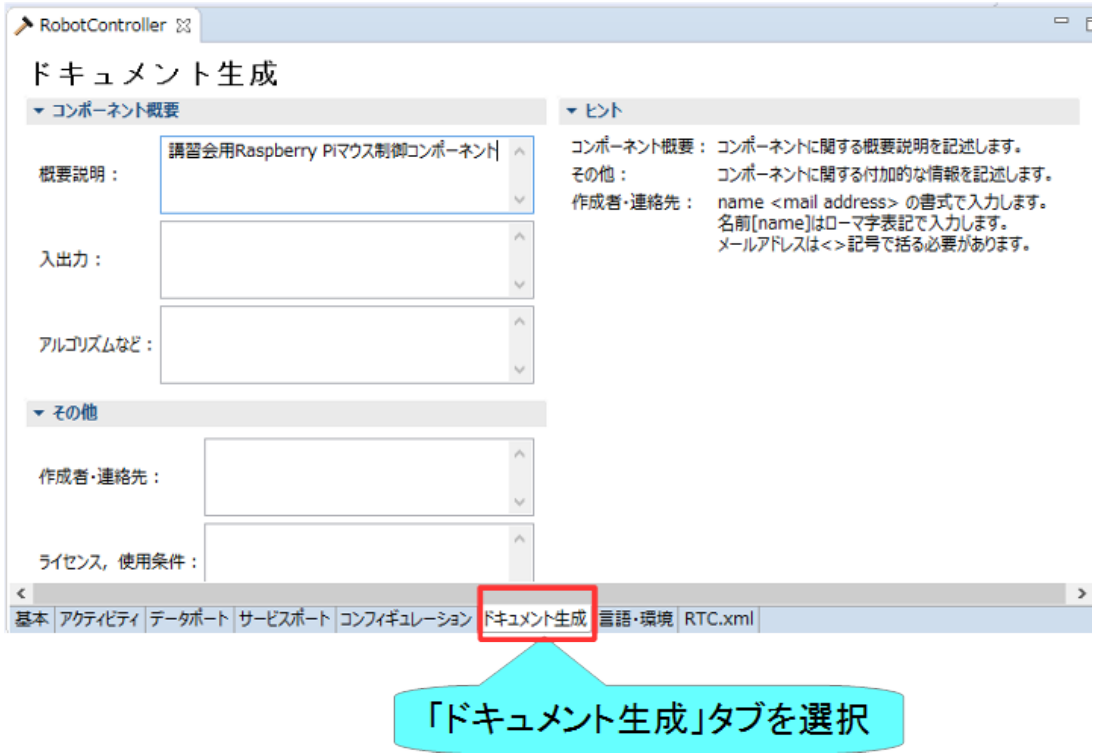
- 制約条件：(0,1,2,3)
- Widget：radio

- 制約条件：(0,1,2,3)
- Widget：checkbox

- 制約条件：(0,1,2,3)
- Widget：ordered_list

ドキュメントの設定

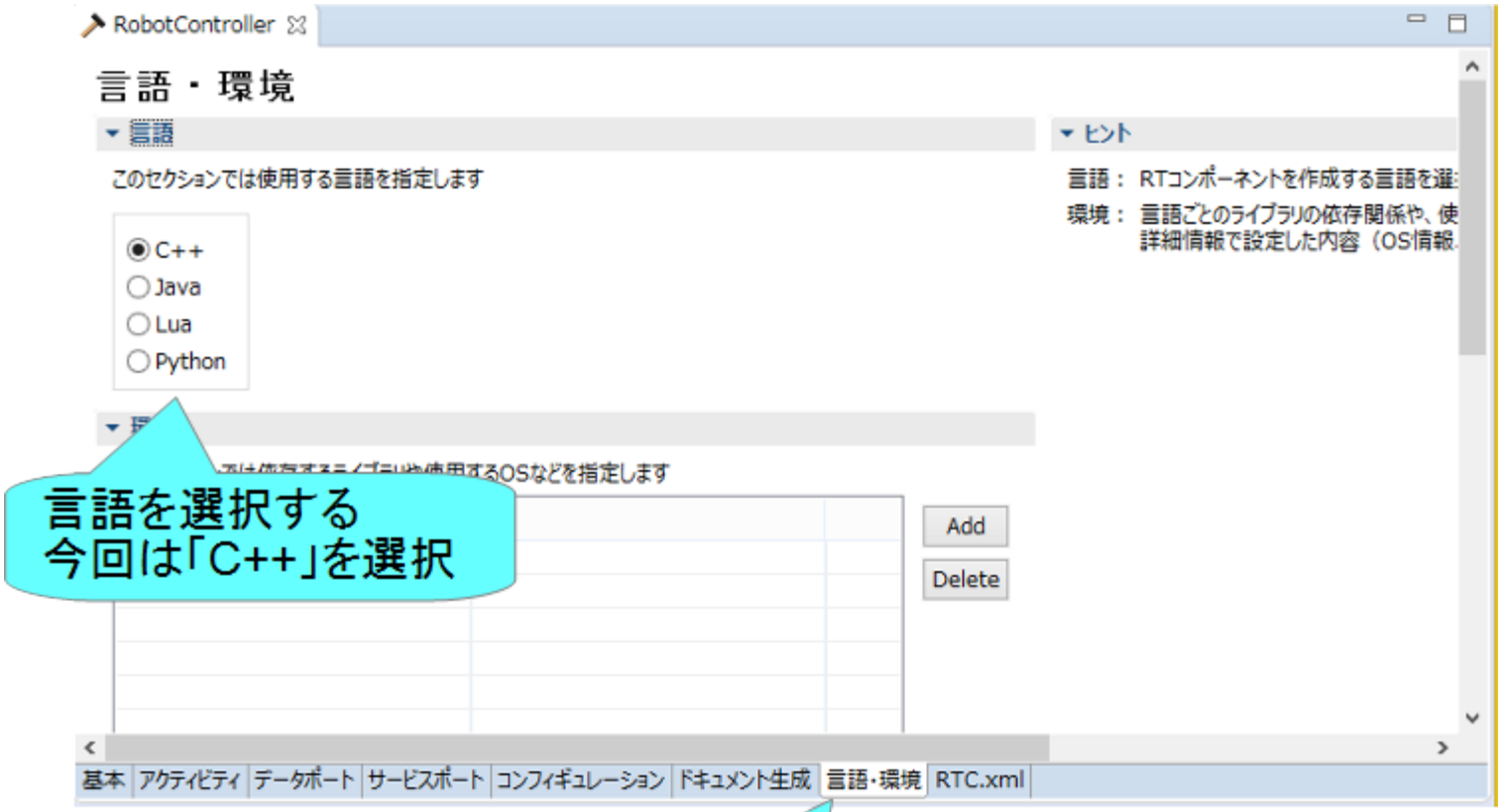
- 各種ドキュメント情報を設定



- 今回は適当に設定しておいてください。
  - 空白でも大丈夫です

言語の設定(OpenRTM-aist 1.2)

- 実装する言語，動作環境に関する情報を設定
  - OpenRTM-aist 1.2と2.0で仕様が変わっているので注意してください
  - 以下は1.2での設定方法

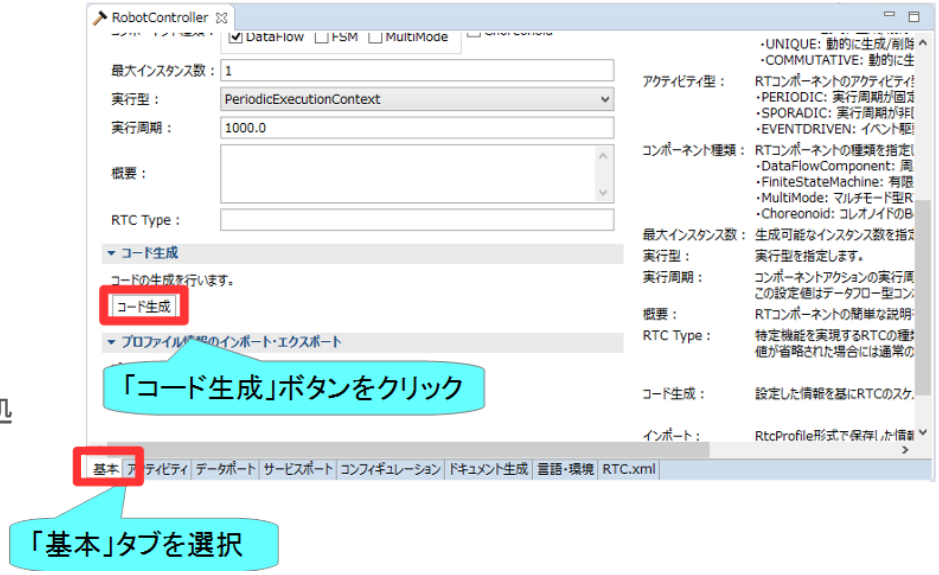


言語の設定(OpenRTM-aist 2.0)

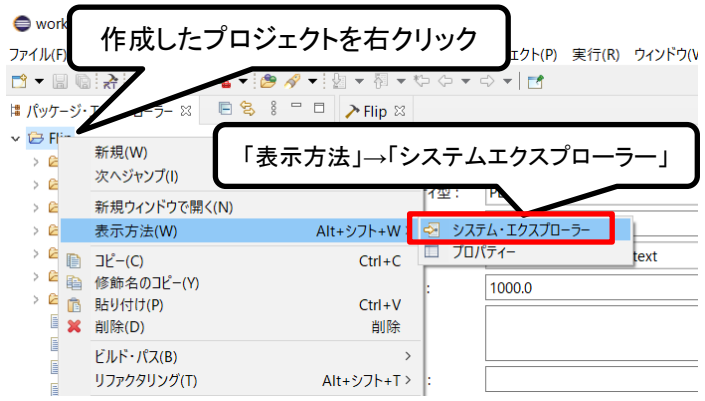
- 実装する言語，動作環境に関する情報を設定
  - OpenRTM-aist 1.2と2.0で仕様が変わっているので注意してください
  - 以下は2.0での設定方法

スケルトンコードの生成

- 基本タブからコード生成ボタンを押すことでスケルトンコードが生成される
  - Workspace¥Flip以下に生成
    - ソースコード
      - C++ソースファイル(.cpp)
      - ヘッダーファイル(.h)
        - » このソースコードにロボットを操作する処理を記述する
    - CMakeの設定ファイル(CMakeLists.txt)
    - rtc.conf、Flip.conf
    - 以下略



- 生成したファイルの確認
  - 作成したプロジェクトを右クリックして、「表示方法」→「システムエクスプローラー」を選択する
  - エクスプローラーでワークスペースのフォルダが開くため、上記のファイルが存在するかを確認する



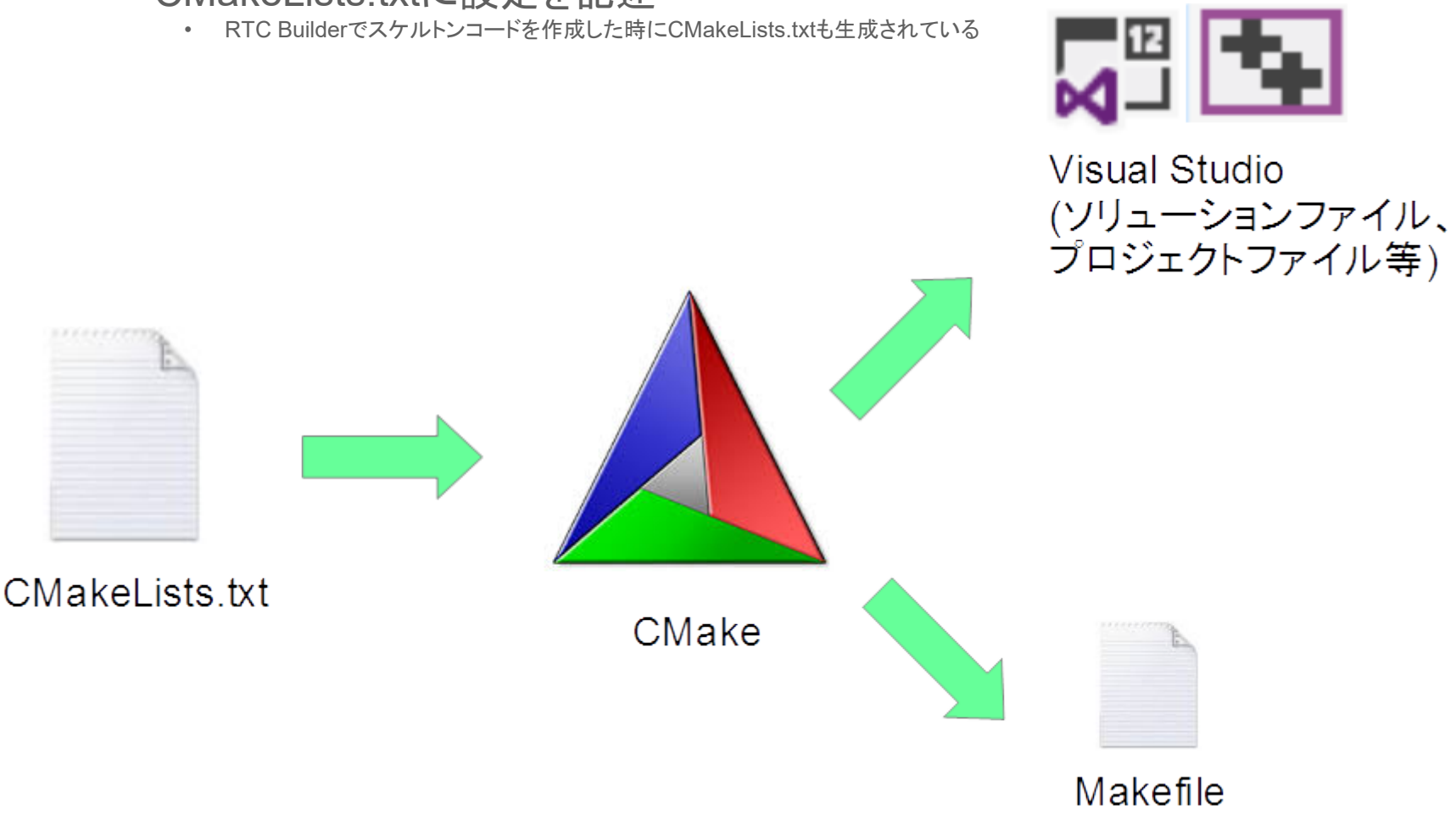
手順

- ビルドに必要な各種ファイルを生成
  - src/CMakeLists.txt
  - CMakeにより各種ファイル生成
- ソースコードの編集
  - Flip.hの編集
  - Flip.cppの編集
- ビルド
  - Windows：Visual Studio
  - Ubuntu：Code::Blocks

ソースコードの編集、RTCのビルド

CMake

- ビルドに必要な各種ファイルを生成
  - CMakeLists.txtに設定を記述
    - RTC Builderでスケルトンコードを作成した時にCMakeLists.txtも生成されている



CMakeLists.txtの編集

- Flip/src/CMakeLists.txtをメモ帳などで開いて編集する
  - OpenCVのライブラリ使用のための設定を行う
    - インクルードファイル、ライブラリ等の検出
    - ライブラリのリンク

```
set(standalone_srcs FlipComp.cpp)

find_package(OpenCV REQUIRED)

if(${OPENRTM_VERSION_MAJOR} LESS 2)
```

4行目あたりに追加

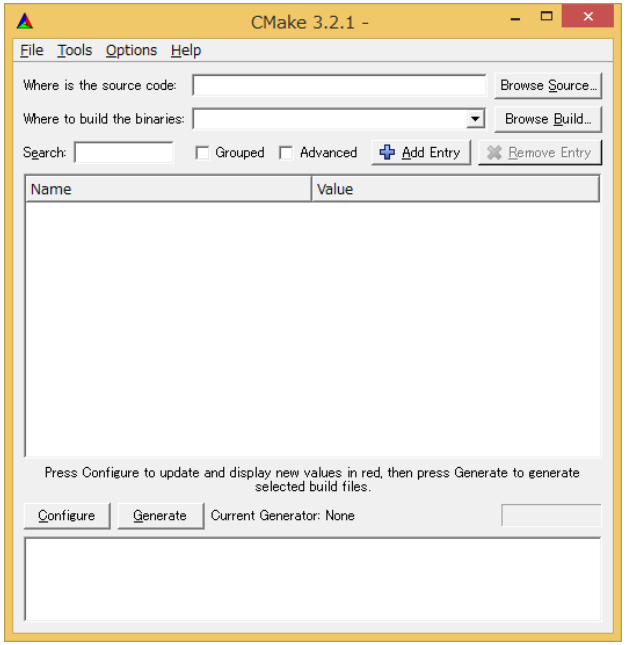
target_link_libraries に**${OpenCV_LIBS}**を追加

```
add_dependencies(${PROJECT_NAME} ALL_IDL_TGT)
target_link_libraries(${PROJECT_NAME} ${OPENRTM_LIBRARIES} ${OpenCV_LIBS})
```

```
add_dependencies(${PROJECT_NAME}Comp ALL_IDL_TGT)
target_link_libraries(${PROJECT_NAME}Comp ${OPENRTM_LIBRARIES} ${OpenCV_LIBS})
```

ビルドに必要なファイルの生成

- CMakeを使用する
  - Windows 7
    - 「スタート」→「すべてのプログラム」→「CMake」→「CMake (cmake-gui)」
  - Windows 8.1
    - 「スタート」→「アプリビュー(右下矢印)」→「CMake」→「CMake (cmake-gui)」
  - Windows 10
    - 左下の「ここに入力して検索」にCMakeと入力して表示されたCMake(cmake-gui)を起動
  - Ubuntu
    - コマンドで「cmake-gui」を入力

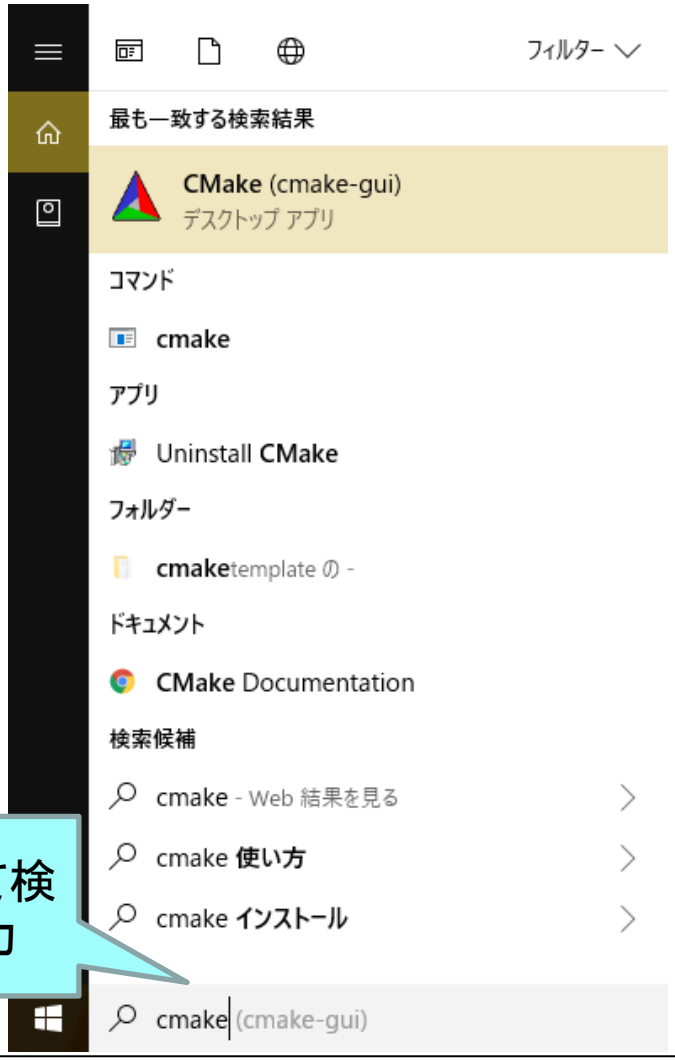


cmake-guiの起動

- Windows 8.1

cmake-guiの起動

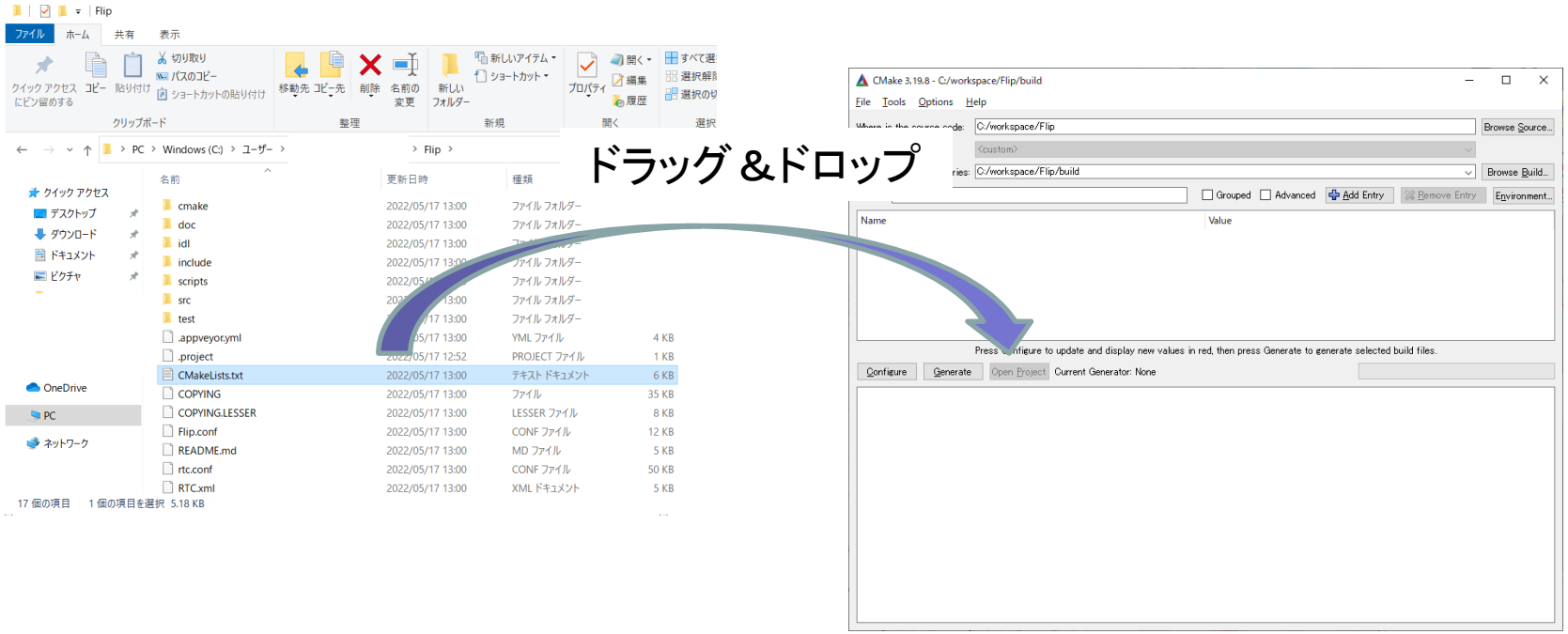
- Windows 10



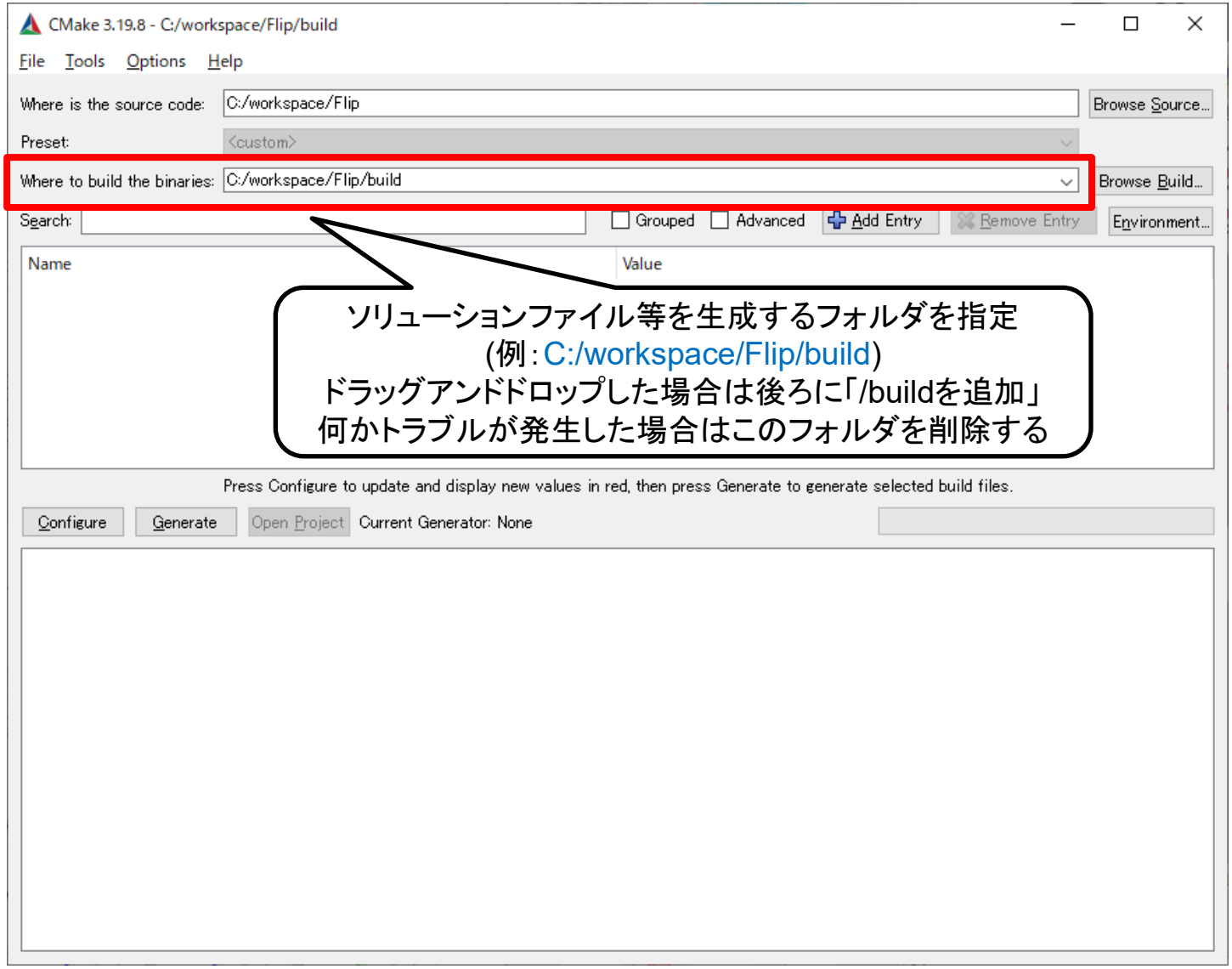
左下の「ここに入力して検索」に「cmake」と入力

ビルドに必要なファイルの生成

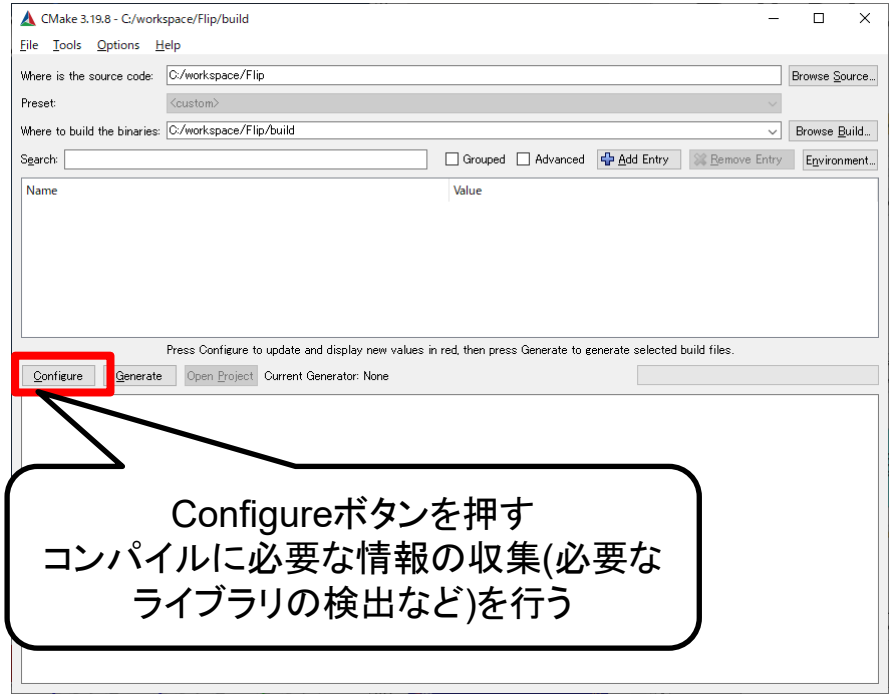
- **CMakeLists.txt**をcmake-guiにドラッグアンドドロップ
  - CMakeLists.txtはRTC Builderで生成したプロジェクトのフォルダ
    - (例：C:¥workspace¥Flip)
    - ※先ほど編集した**src/CMakeLists.txtではなく**、一つ上のディレクトリのファイル



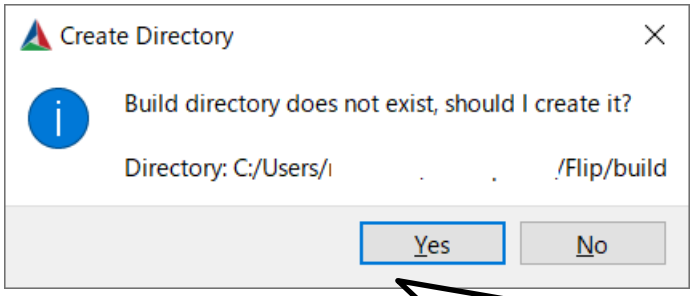
ビルドに必要なファイルの生成



ビルドに必要なファイルの生成



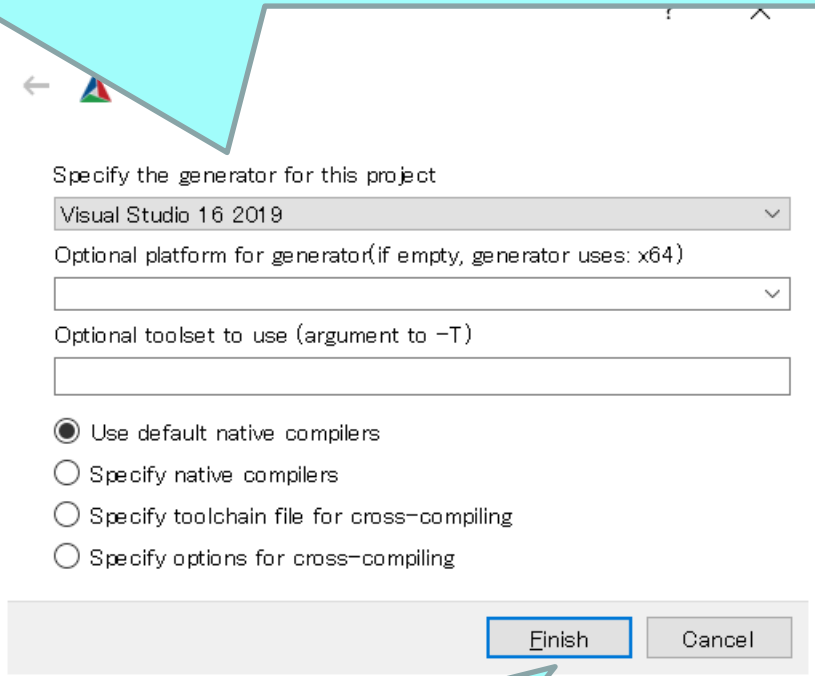
Configureボタンを押す
コンパイルに必要な情報の収集(必要なライブラリの検出など)を行う



Buildフォルダが存在しない場合は作成するかどうか確認してくるため「Yes」を選択

CMake 3.14以降の場合

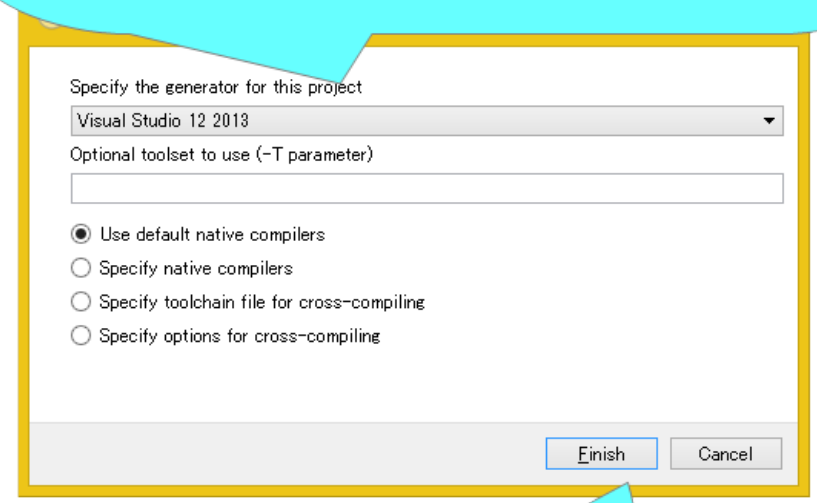
ビルド環境の設定
Visual Studio 2019 → Visual Studio 16 2019
Visual Studio 2022 → Visual Studio 17 2022



設定後、Finishボタンを押す

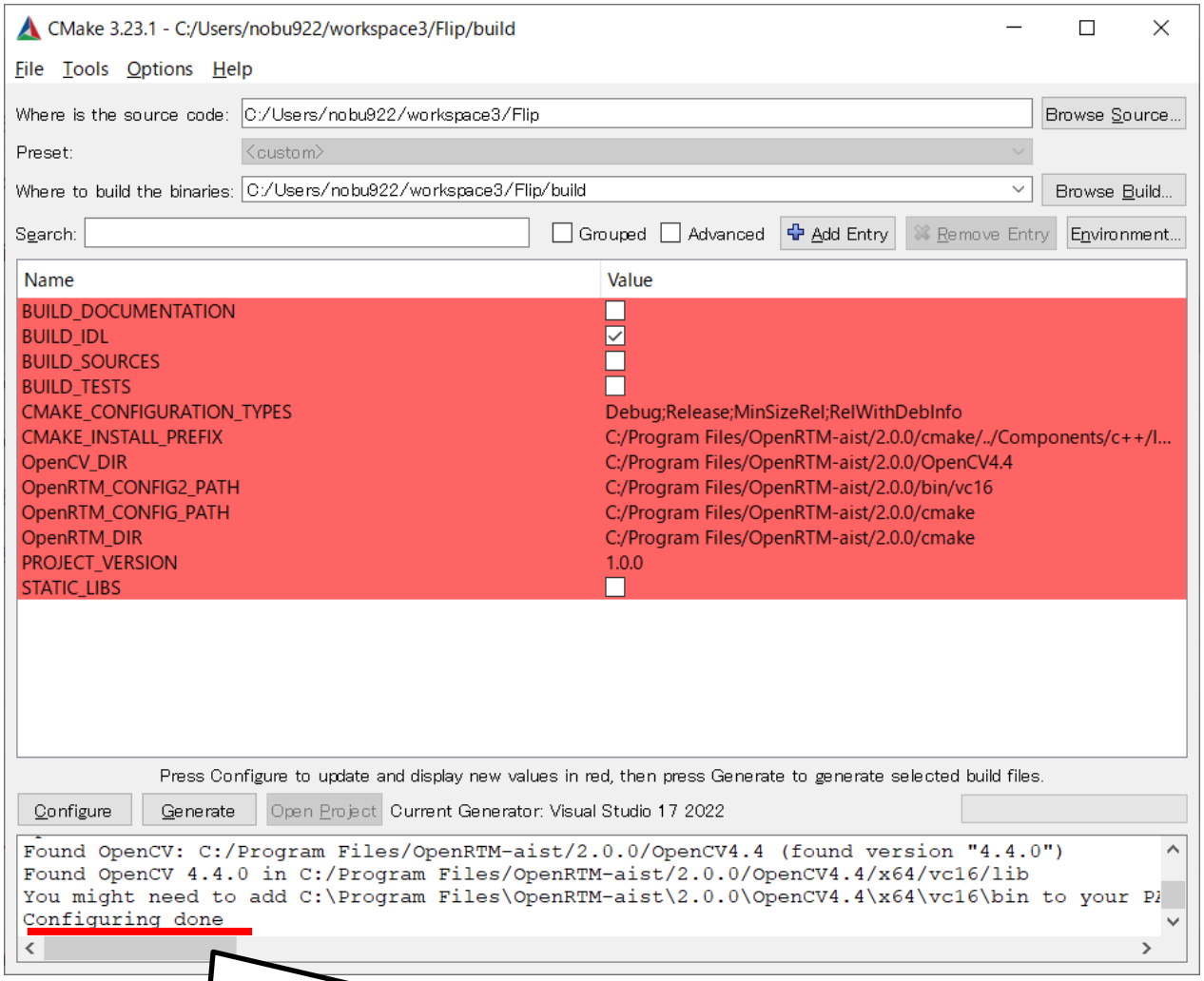
CMake 3.13以前の場合

ビルド環境の設定
Visual Studio 2013 32bit → Visual Studio 12 2013
Visual Studio 2013 64bit → Visual Studio 12 2013 Win64
Visual Studio 2017 32bit → Visual Studio 15 2017
Visual Studio 2017 64bit → Visual Studio 15 2017 Win64
Code::Blocks → CodeBlocks-Unix Makefiles
※32bitか64bitかはインストールした
OpenRTM-aistが32bitか64bitかで選択



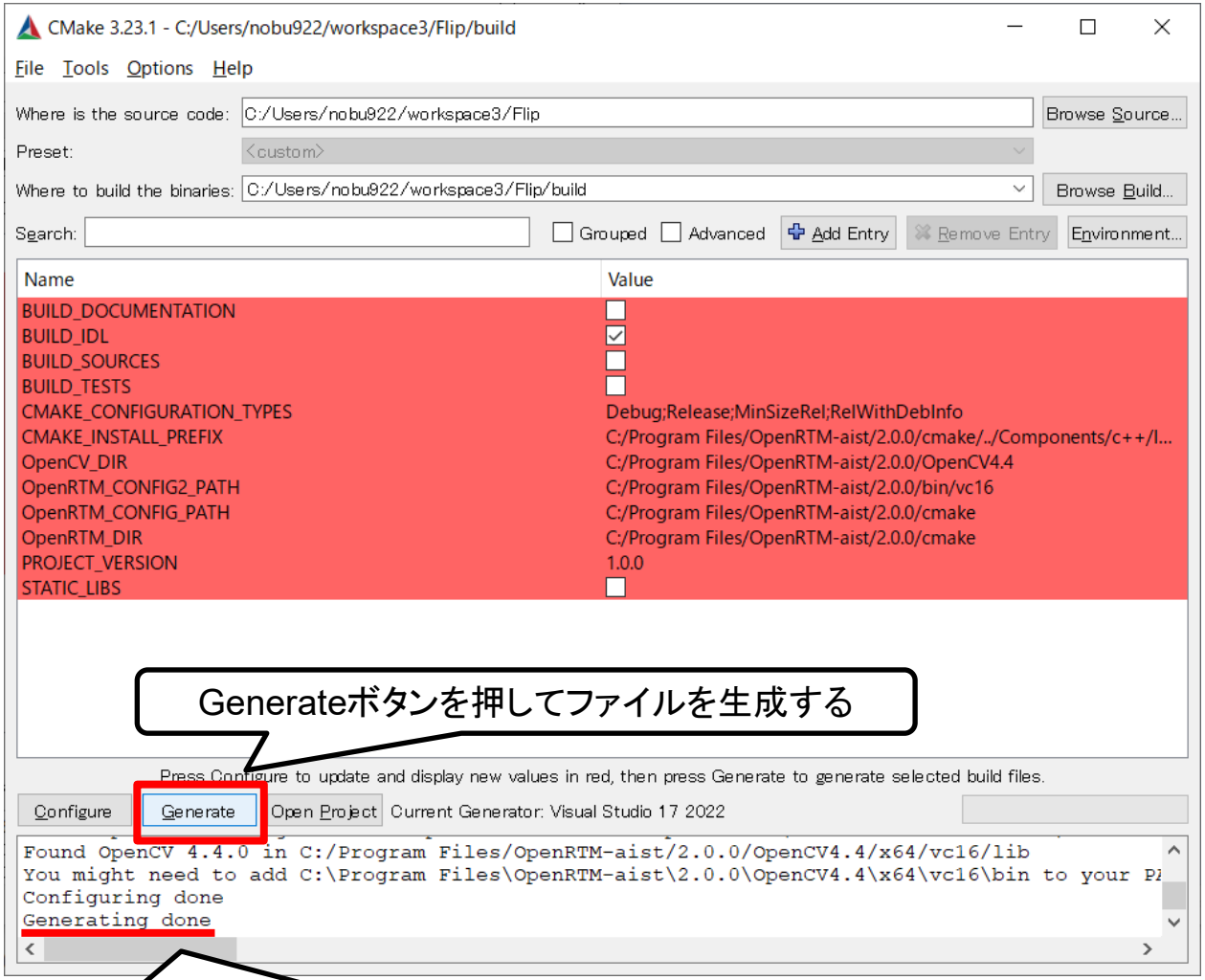
設定後、Finishボタンを押す

ビルドに必要なファイルの生成

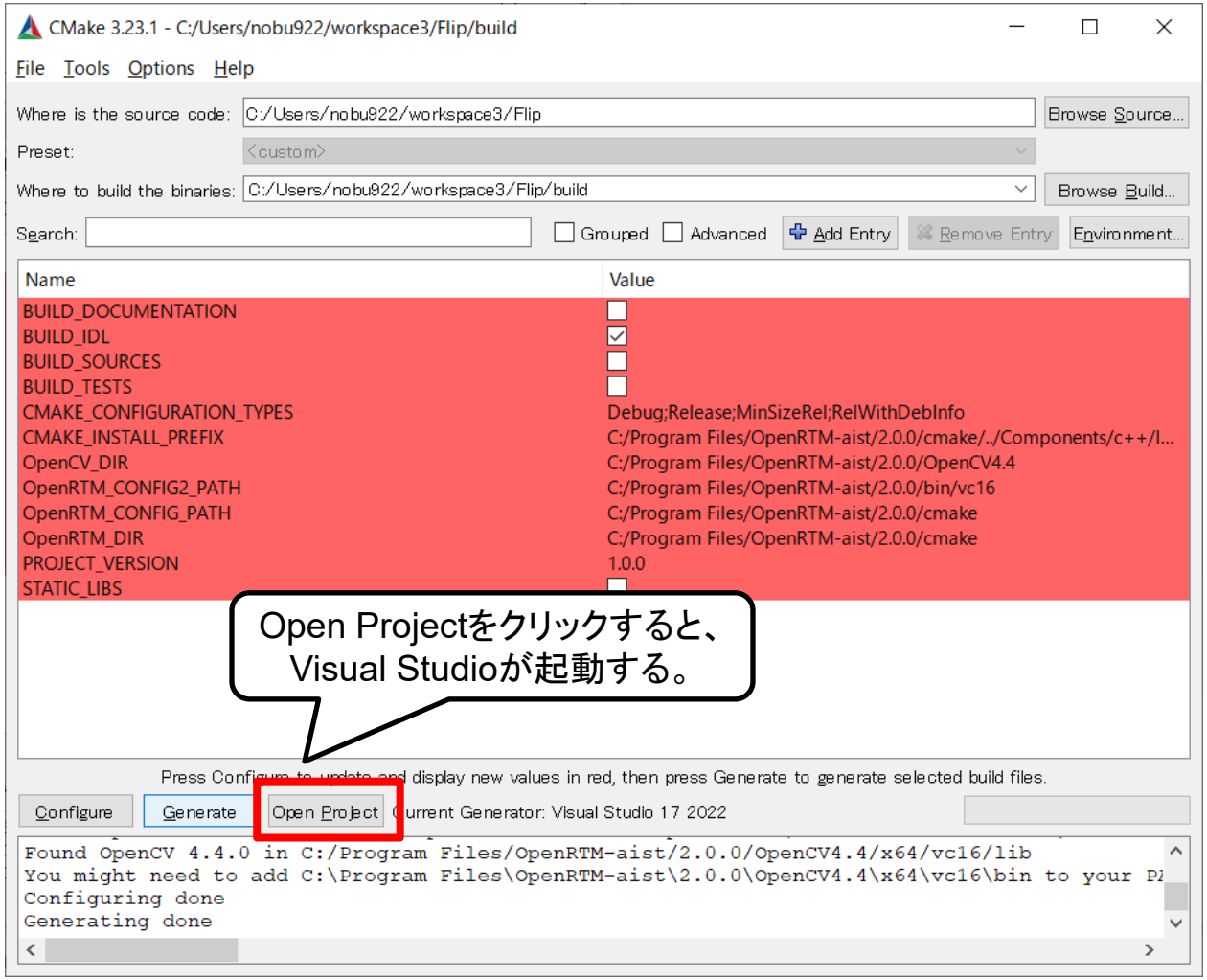


「Configure done」が表示されていれば成功

ビルドに必要なファイルの生成

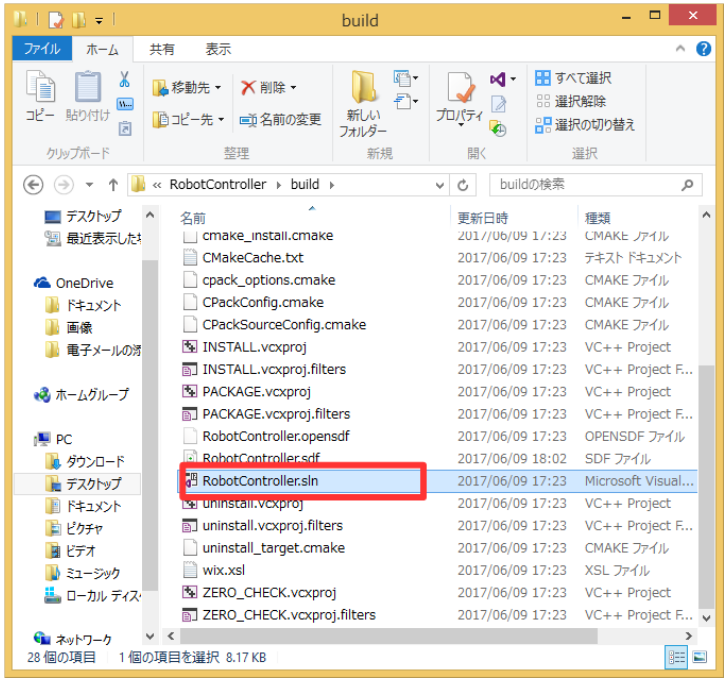


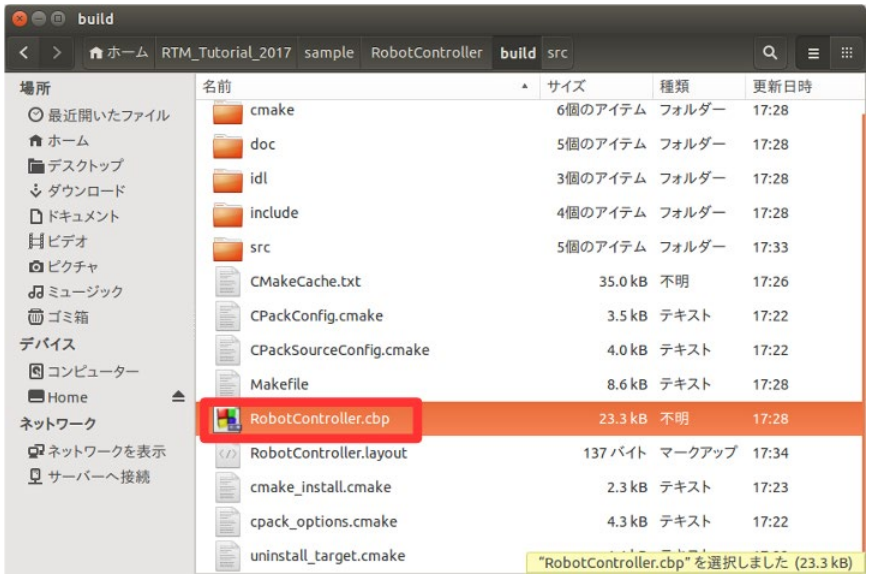
ソースコードの編集


CMake 3.23.1 - C:/Users/nobu922/workspace3/Flip/build
File Tools Options Help
Where is the source code: C:/Users/nobu922/workspace3/Flip
Browse Source...
Preset: <custom>
Where to build the binaries: C:/Users/nobu922/workspace3/Flip/build
Browse Build...
Search:
Grouped
Advanced
Add Entry
Remove Entry
Environment...
Name
Value
BUILD_DOCUMENTATION
BUILD_IDL
BUILD_SOURCES
BUILD_TESTS
CMAKE_CONFIGURATION_TYPES
Debug;Release;MinSizeRel;RelWithDebInfo
CMAKE_INSTALL_PREFIX
C:/Program Files/OpenRTM-aist/2.0.0/cmake/../Components/c++/l...
OpenCV_DIR
C:/Program Files/OpenRTM-aist/2.0.0/OpenCV4.4
OpenRTM_CONFIG2_PATH
C:/Program Files/OpenRTM-aist/2.0.0/bin/vc16
OpenRTM_CONFIG_PATH
C:/Program Files/OpenRTM-aist/2.0.0/cmake
OpenRTM_DIR
C:/Program Files/OpenRTM-aist/2.0.0/cmake
PROJECT_VERSION
1.0.0
STATIC_LIBS
Open Projectをクリックすると、
Visual Studioが起動する。
Press Configure to update and display new values in red, then press Generate to generate selected build files.
Configure
Generate
Open Project
Current Generator: Visual Studio 17 2022
Found OpenCV 4.4.0 in C:/Program Files/OpenRTM-aist/2.0.0/OpenCV4.4/x64/vc16/lib
You might need to add C:\Program Files\OpenRTM-aist\2.0.0\OpenCV4.4\x64\vc16\bin to your P
Configuring done
Generating done

ソースコードの編集

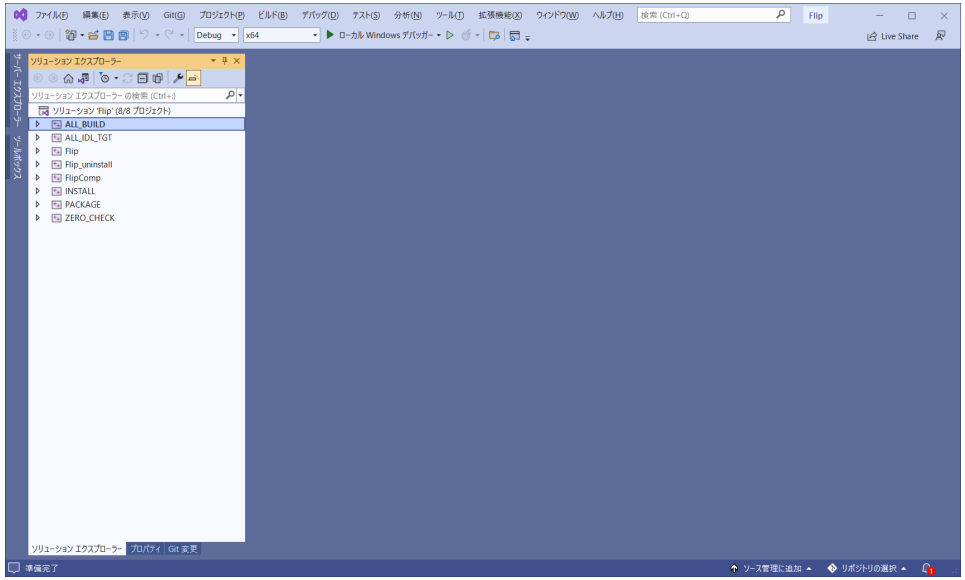
- CMake-guiのバージョンが古い場合は「Open Project」ボタンがないため、ファイルをダブルクリックして開く
  - Windows
    - buildフォルダの「Flip.sln」をダブルクリックして開く
  - Ubuntu
    - buildフォルダの「Flip.cbp」をダブルクリックして開く

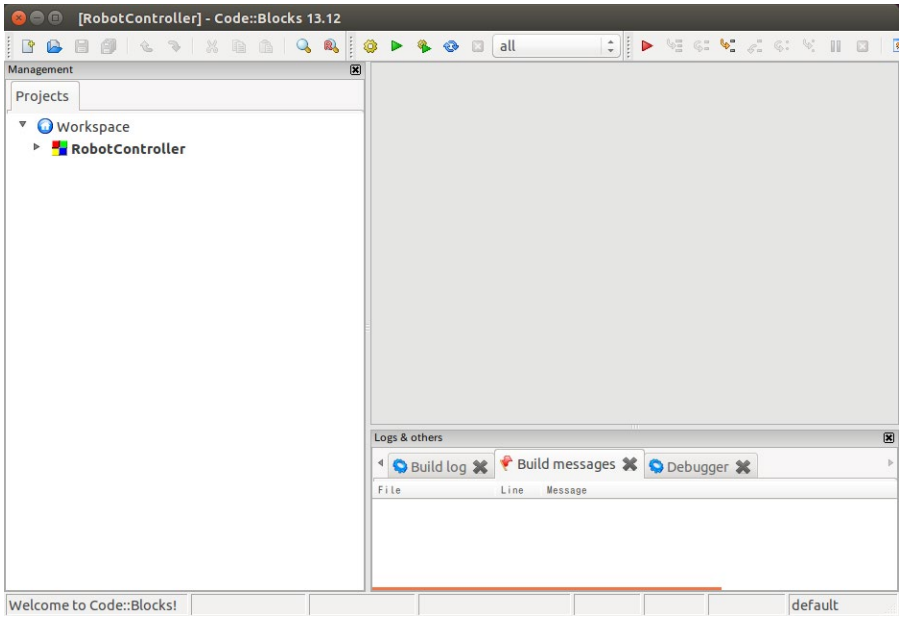




ソースコードの編集

- Windows
  - Visual Studioが起動
- Ubuntu
  - Code::Blocksが起動

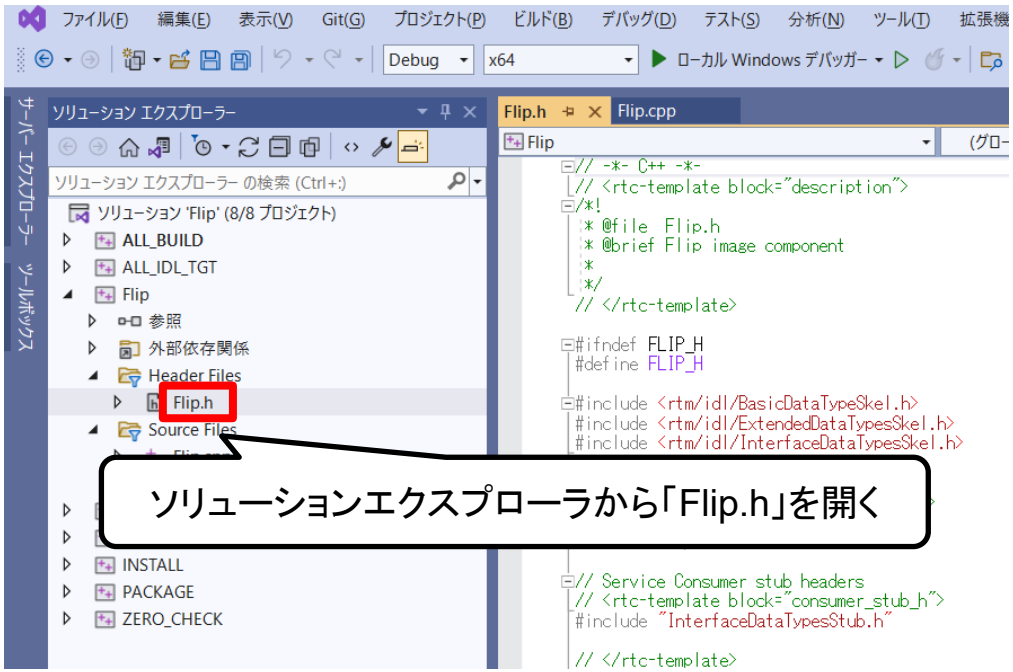




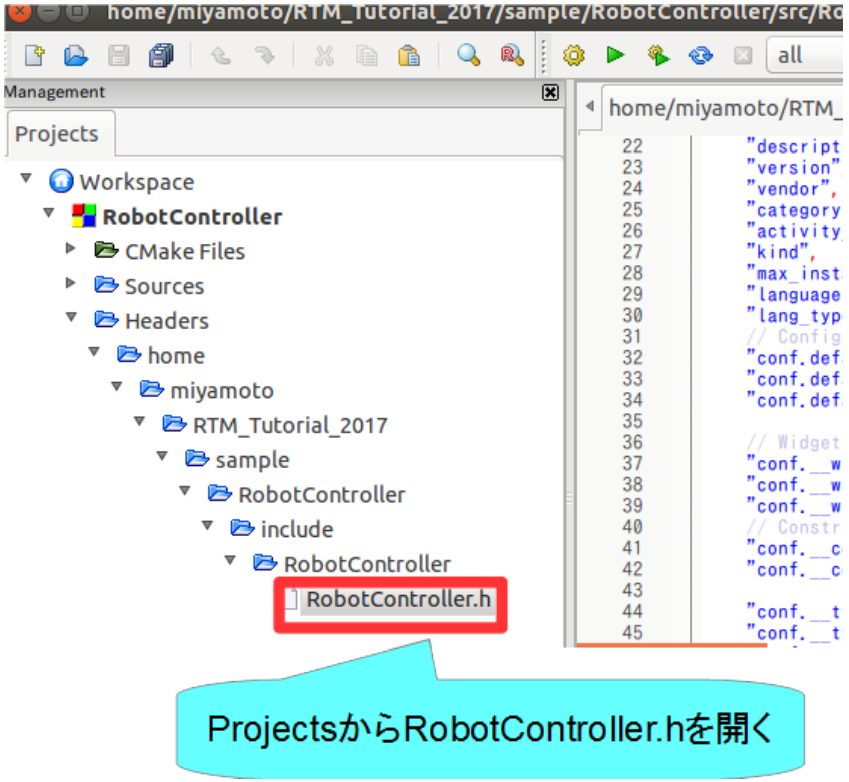
ソースコードの編集

- Flip.hの編集

Visual Studio



Code::Blocks



ソースコードの編集

- Flip.hの編集

```
#include <rtm/DataOutPort.h>

//OpenCVのヘッダーファイルをインクルード
#include <opencv2/opencv.hpp>

// <rtc-template block="component_description">
```

35行目あたりに追加

```
private:
  // <rtc-template block="private_attribute">

  // </rtc-template>

  // <rtc-template block="private_operation">

  // </rtc-template>
  //変換前画像を格納する変数
  cv::Mat m_imageBuff;

  //変換後画像を格納する変数
  cv::Mat m_flipImageBuff;
```

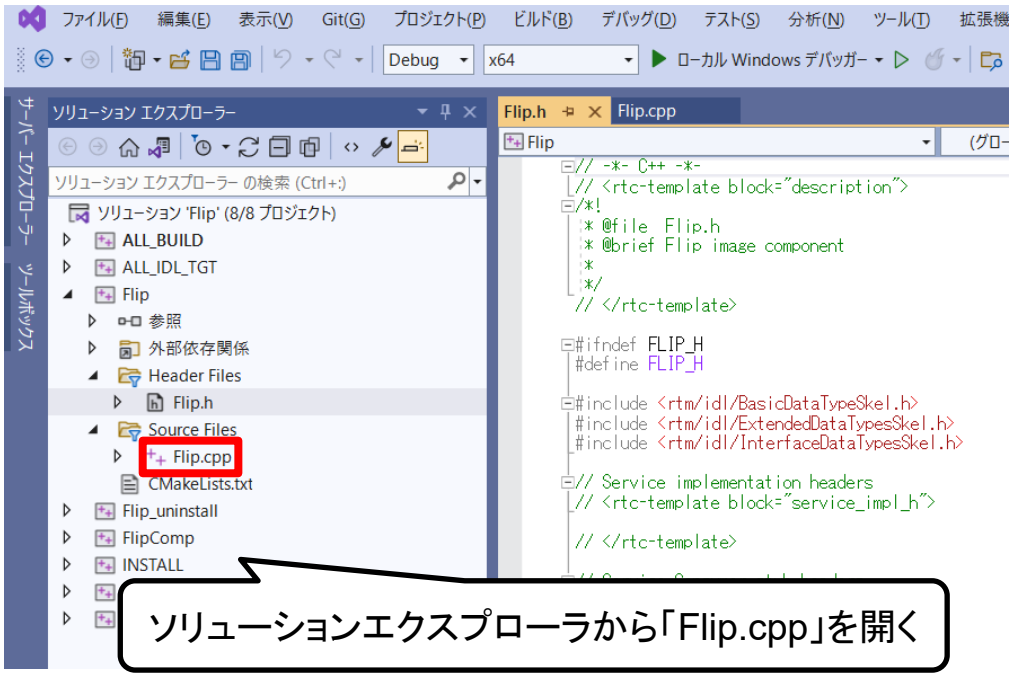
271行目あたりに追加

274行目あたりに追加

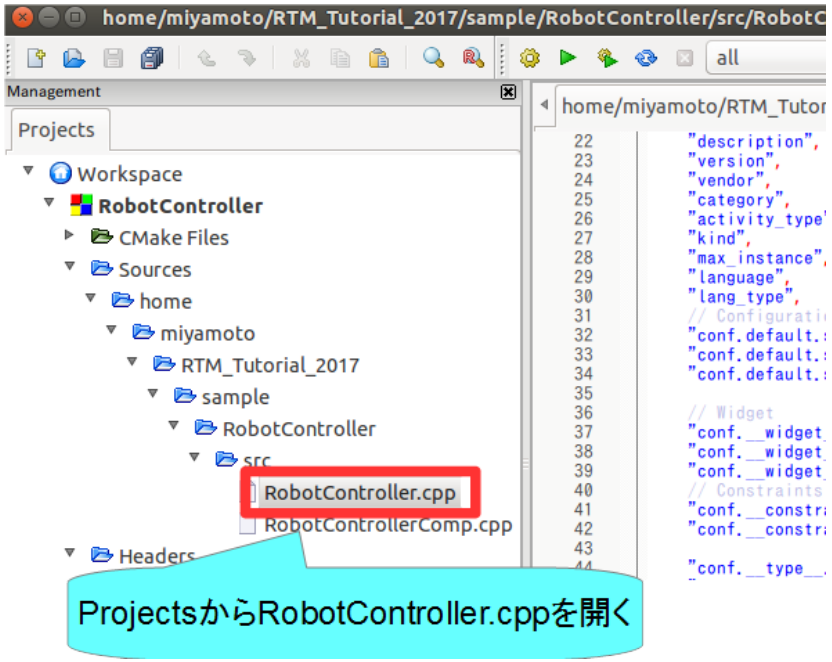
ソースコードの編集

- Flip.cppの編集
  - 詳細はWEBページの資料を参考にしてください

Visual Studio



Code::Blocks



ソースコードの編集

- Flip.cppの編集

```
RTC::ReturnCode_t Flip::onActivated(RTC::UniqueId /*ec_id*/)
{
  // OutPortの画面サイズの初期化
  m_flippedImage.width = 0;
  m_flippedImage.height = 0;
  return RTC::RTC_OK;
}
```

onActivated関数に追加

```
RTC::ReturnCode_t Flip::onDeactivated(RTC::UniqueId /*ec_id*/)
{
  if (!m_imageBuff.empty())
  {
      // 画像用メモリの解放
      m_imageBuff.release();
      m_flipImageBuff.release();
  }
  return RTC::RTC_OK;
}
```

onDeactivated関数に追加

ソースコードの編集

- Flip.cppの編集

```
RTC::ReturnCode_t Flip::onExecute(RTC::UniqueId /*ec_id*/)
{
  // 新しいデータのチェック
  if (m_originalImageIn.isNew()) {
      // InPortデータの読み込み
      m_originalImageIn.read();

      // InPortとOutPortの画面サイズ処理およびイメージ用メモリの確保
      if (m_originalImage.width != m_flippedImage.width || m_originalImage.height != m_flippedImage.height)
      {
          m_flippedImage.width = m_originalImage.width;
          m_flippedImage.height = m_originalImage.height;

          m_imageBuff.create(cv::Size(m_originalImage.width, m_originalImage.height), CV_8UC3);
          m_flipImageBuff.create(cv::Size(m_originalImage.width, m_originalImage.height), CV_8UC3);


      }

      // InPortの画像データをm_imageBuffにコピー
      memcpy(m_imageBuff.data, (void*)&(m_originalImage.pixels[0]), m_originalImage.pixels.length());

      // InPortからの画像データを反転する。 m_flipMode 0: X軸周り、1: Y軸周り、-1: 両方の軸周り
      cv::flip(m_imageBuff, m_flipImageBuff, m_flipMode);

      // 画像データのサイズ取得
      int len = m_flipImageBuff.channels() * m_flipImageBuff.cols * m_flipImageBuff.rows;
      m_flippedImage.pixels.length(len);

      // 反転した画像データをOutPortにコピー
      memcpy((void*)&(m_flippedImage.pixels[0]), m_flipImageBuff.data, len);

      // 反転した画像データをOutPortから出力する。
      m_flippedImageOut.write();
  }
  return RTC::RTC_OK;
}
```

onExecute関数に追加

ソースコードの編集

- データを読み込む手順

isNew関数で新規に書き込まれたデータが存在するかを確認

read関数でデータ読み込み

read関数を呼び出した時点で、変数m_originalImageにデータが格納される

```
if (m_originalImageIn.isNew()) {
    // InPortデータの読み込み
    m_originalImageIn.read();

    // InPortとOutPortの画
    if (m_originalImage.width != m_flippedImage.width || m_originalImage.height != m_flippedImage.height)
    {
        m_flippedImage.width = m_originalImage.width;
        m_flippedImage.height = m_originalImage.height;

        m_imageBuff.create(cv::Size(m_originalImage.width, m_originalImage.height), CV_8UC3);
        m_flipImageBuff.create(cv::Size(m_originalImage.width, m_originalImage.height), CV_8UC3);


    }

    // InPortの画像データをm_imageBuffにコピー
    memcpy(m_imageBuff.data, (void*)&(m_originalImage.pixels[0]), m_originalImage.pixels.length());
```

補足：ameraImage型はpixelsに画像データ、witdthに画像幅、heightに画像高さが格納されているため、これを利用する

ソースコードの編集

- データを書き込む手順

補足：コンフィギュレーションパラメータを変更すると対応する変数(m_flipMode)に値が格納される

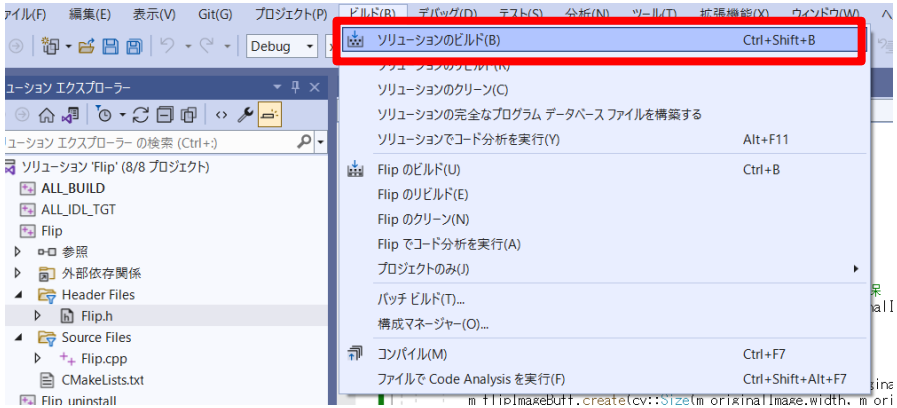
```
// InPortからの画像データを反転する。 m_flipMode 0: X軸周り、1: Y軸周り、-1: 両方の軸周り
cv::flip(m_imageBuff, m_flipImageBuff, m_flipMode);

// 画像データのサイズ取得
int len = m_flipImageBuff.channels() * m_flipImageBuff.cols * m_flipImageBuff.rows;
m_flippedImage.pixels.length(len);

// 反転した画像データをOutPortにコピー
memcpy((void*)&(m_flippedImage.pixels[0]), m_flipImageBuff.data, len);

// 反転した画像データをOutPortから出力する。
m_flippedImageOut.write();
```

変数m_flippedImageのpixelsに画像データをコピーする。
※witdthに画像幅、heightに画像高さは設定済み

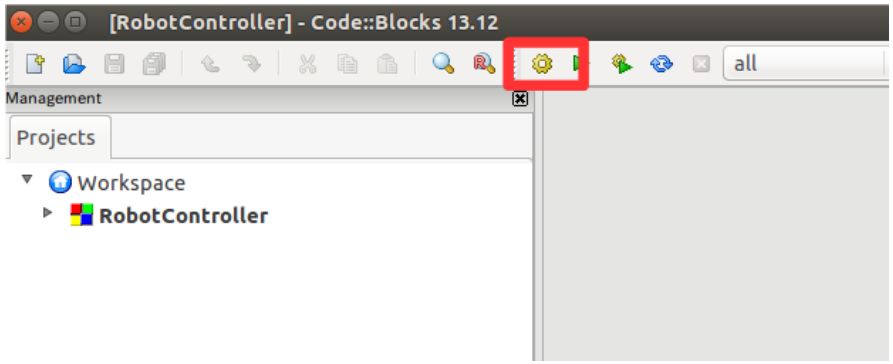
write関数でデータ書き込み

ソースコードのコンパイル

Visual Studio



Code::Blocks

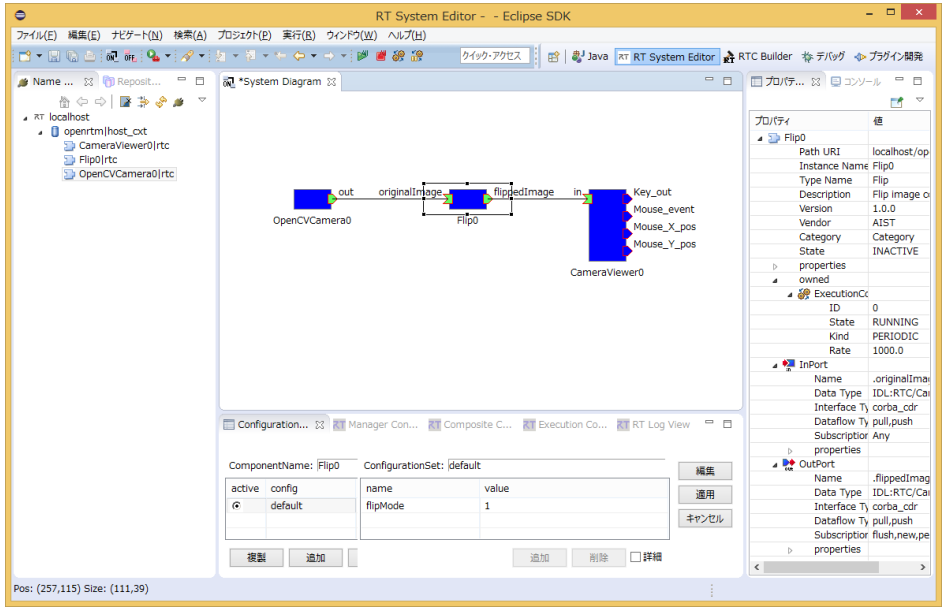


- 成功した場合、実行ファイルが生成される
  - Windows
    - **build¥src**フォルダの**Release**(もしくは**Debug**)フォルダ内にFlipComp.exeが生成される
  - Ubuntu
    - **build/src**フォルダにFlipCompが生成される

システム構築支援ツール
RT System Editorについて

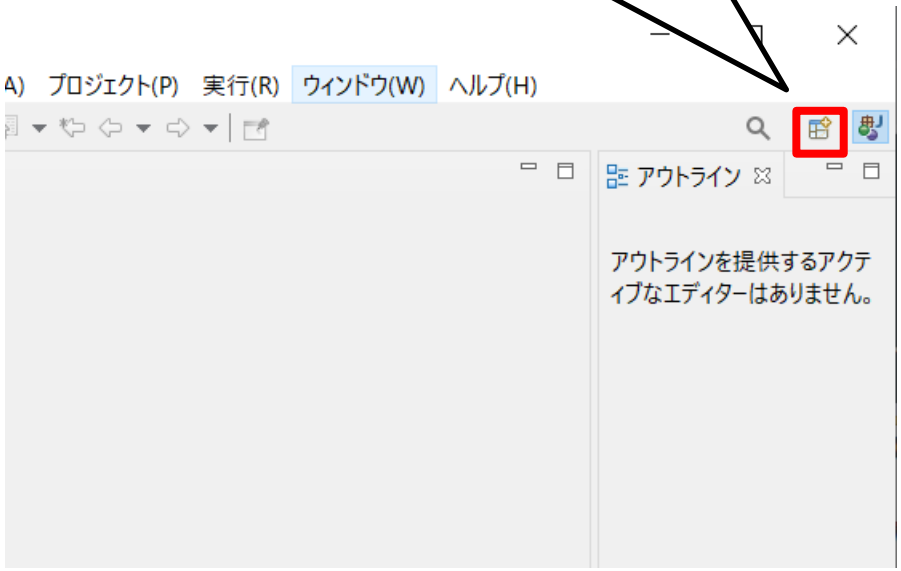
RT System Editor

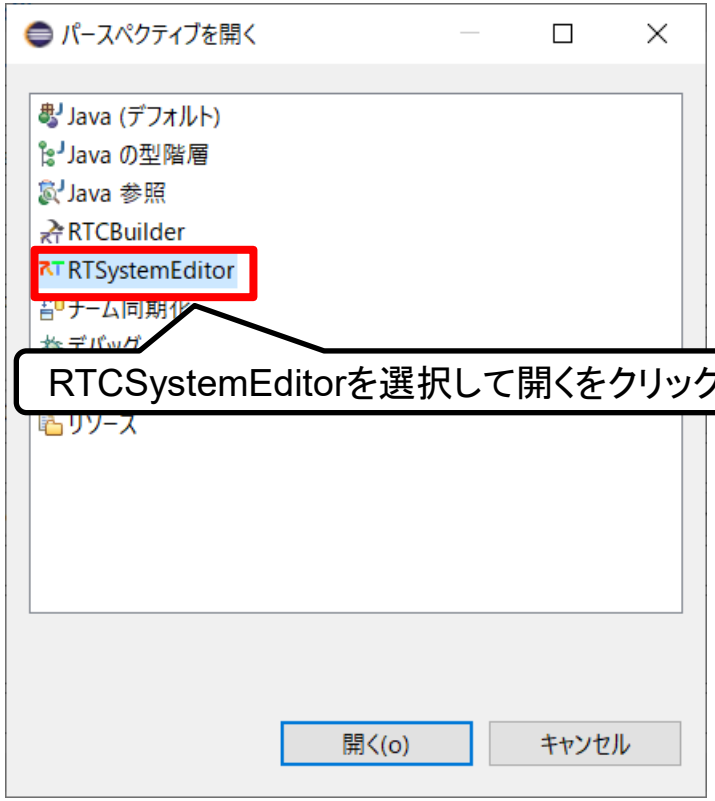
- RTCをGUIで操作するためのツール
  - データポート、サービスポートの接続
  - アクティブ化、非アクティブ化、リセット、終了
  - コンフィギュレーションパラメータの操作
  - 実行コンテキストの操作
    - 実行周期変更
    - 実行コンテキストの関連付け
  - 複合化
  - マネージャからRTCを起動
  - 作成したRTシステムの保存、復元



RT System Editorの起動

右上の「パースペクティブを開く」ボタンをクリック



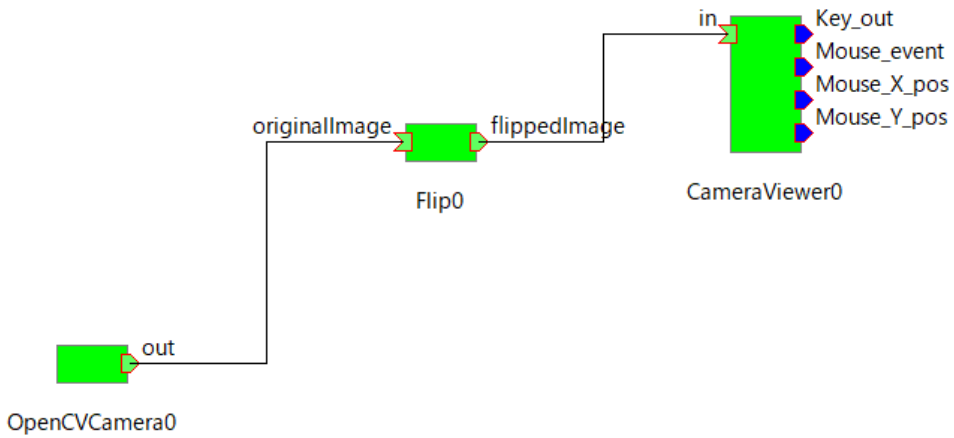


RTCSystemEditorを選択して開くをクリック

RT System Editorの画面構成

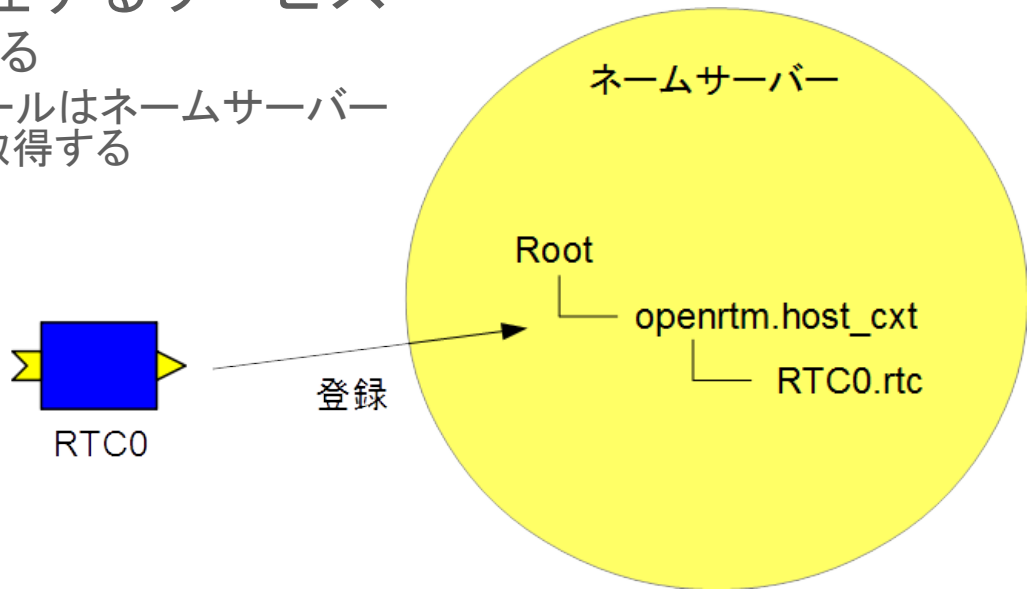
Flipコンポーネントの動作確認

- OpenCVCamera、CameraViewerコンポーネントと接続してカメラ画像を反転して表示するRTシステムを作成する
  - ネームサーバーを起動する
  - OpenCVCamera、CameraViewerコンポーネントを起動する
    - Windows
      - 「C++_OpenCV-Examples」を検索してエクスプローラ起動
      - 「OpenCVCameraComp.bat」と「CameraViewerComp.bat」を実行
    - Ubuntu
      - 以下のコマンド実行
        - » $ /usr/local/share/openrtm-1.2/components/c++/opencv-rtcs/CameraViewerComp
        - » $ /usr/local/share/openrtm-1.2/components/c++/opencv-rtcs/OpenCVCameraComp
  - Flipコンポーネント起動
  - OpenCVCamera、CameraViewer、Flipコンポーネントを接続して「All Activate」を行う

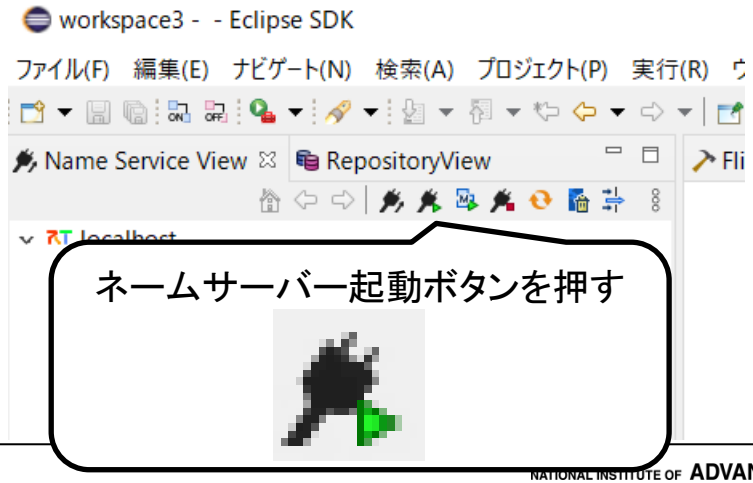


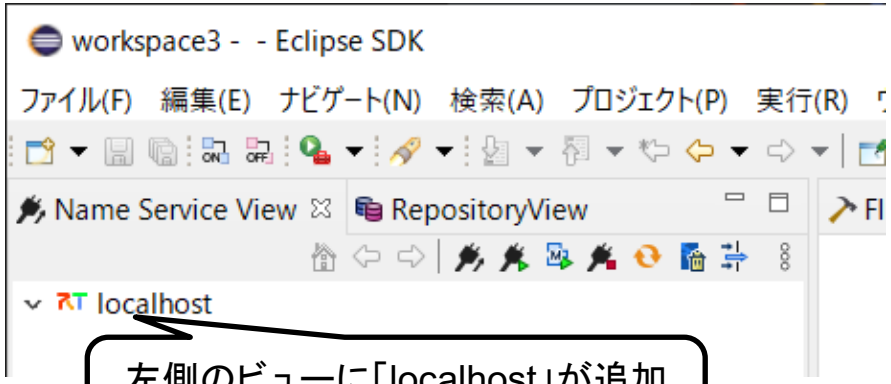
ネームサーバーの起動

- オブジェクトを名前で管理するサービス
  - RTCを一意の名前で登録する
    - RT System Editor等のツールはネームサーバーから名前でRTCの参照を取得する



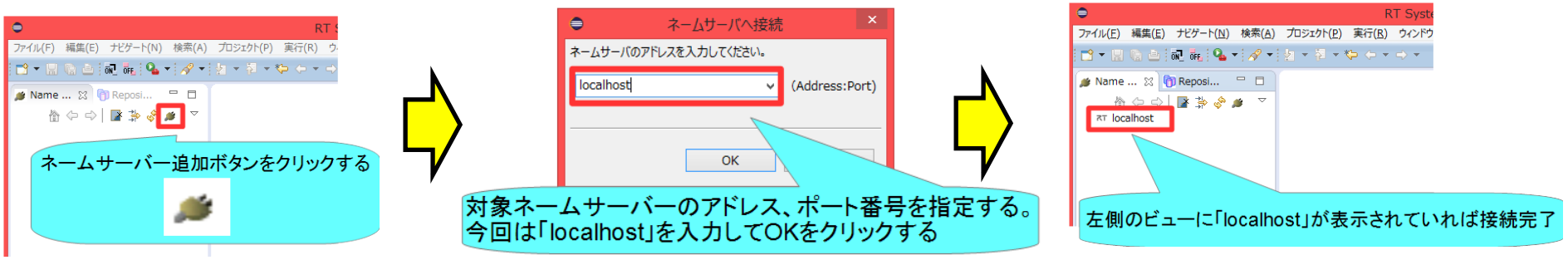
- 起動する手順





ネームサーバーの起動

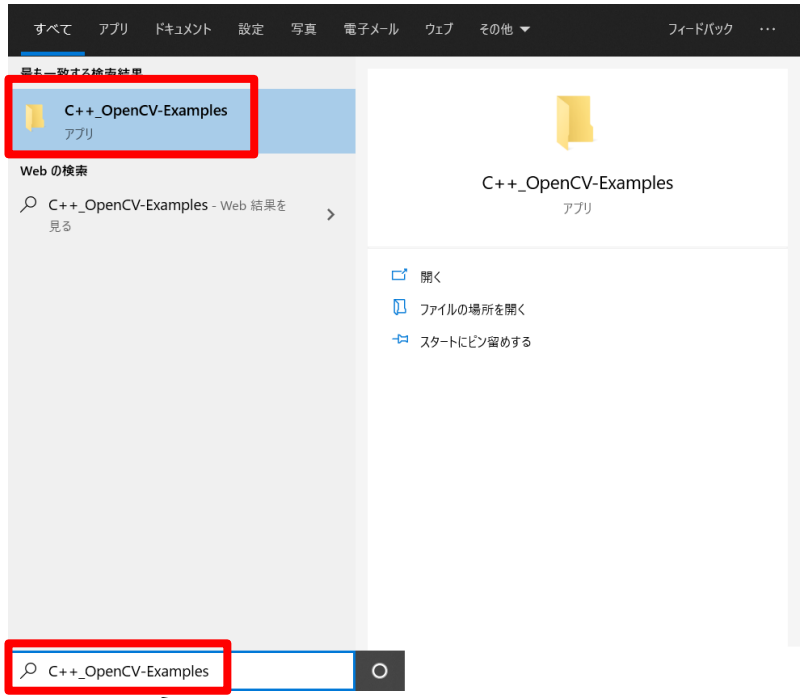
- OpenRTM-aist 1.1.2以前の手順
  - Windows 7
    - 「スタート」→「すべてのプログラム」→「OpenRTM-aist 1.2.0」→「Tools」→「Start Naming Service」
  - Windows 8.1
    - 「スタート」→「アプリビュー(右下矢印)」→「OpenRTM-aist 1.2.0」→「Start Naming Service」
  - Windows 10
    - 左下の「ここに入力して検索」にStart Naming Serviceと入力して起動
  - Ubuntu
    - $ rtm-naming



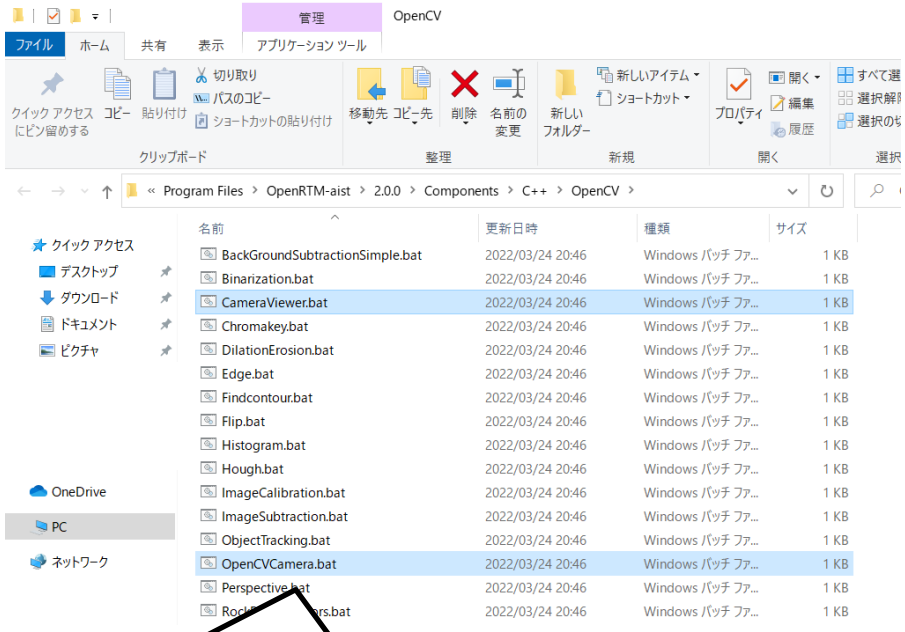
Flipコンポーネントの動作確認

- OpenCVCamera、CameraViewerコンポーネントと接続してカメラ画像を反転して表示するRTシステムを作成する
  - ネームサーバーを起動する
  - OpenCVCamera、CameraViewerコンポーネントを起動する
    - Windows
      - 「C++_OpenCV-Examples」を検索してエクスプローラ起動
      - 「OpenCVCameraComp.bat」と「CameraViewerComp.bat」を実行
    - Ubuntu
      - 以下のコマンド実行
        - » $ /usr/local/share/openrtm-1.2/components/c++/opencv-rtcs/CameraViewerComp
        - » $ /usr/local/share/openrtm-1.2/components/c++/opencv-rtcs/OpenCVCameraComp
  - Flipコンポーネント起動
    - Windows
      - **build¥src**フォルダの**Release**(もしくは**Debug**)フォルダ内にFlipComp.exeが生成されているためこれを起動する
    - Ubuntu
      - **build/src**フォルダにFlipCompが生成されているためこれを起動する
  - OpenCVCamera、CameraViewer、Flipコンポーネントを接続して「All Activate」を行う

Flipコンポーネントの動作確認 (Windows 10)

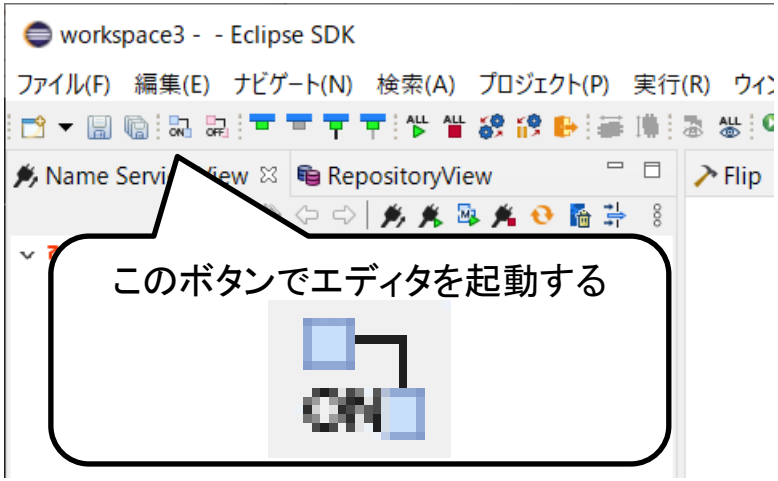


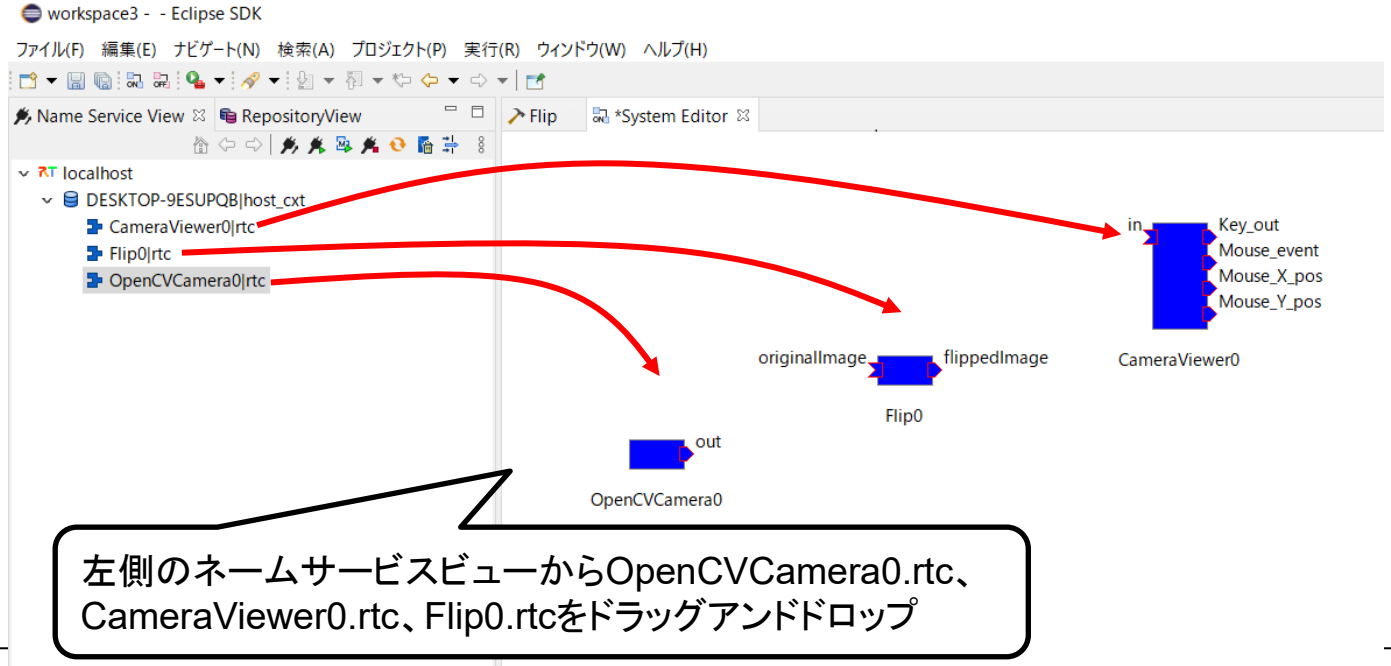
画面左下の「ここに入力して検索」に「C++_OpenCV-Examples」と入力して検索する



「OpenCVCamera.bat」と「CameraViewer.bat」をダブルクリックして起動する

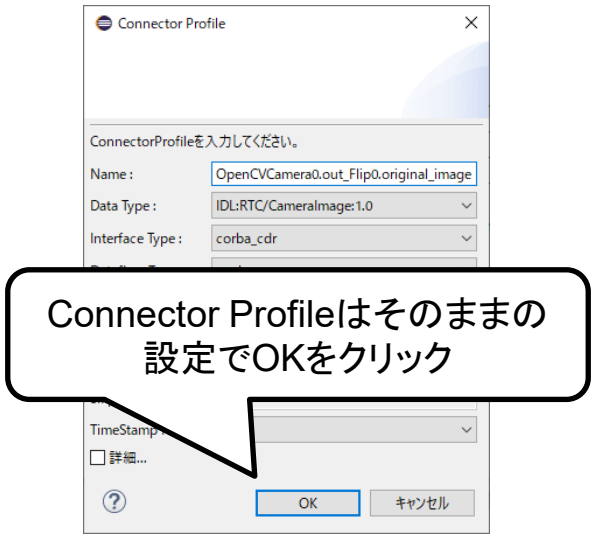
データポートの接続


workspace3 - - Eclipse SDK
ファイル(F) 編集(E) ナビゲート(N) 検索(A) プロジェクト(P) 実行(R)
RepositoryView
Flip
このボタンでエディタを起動する
ON

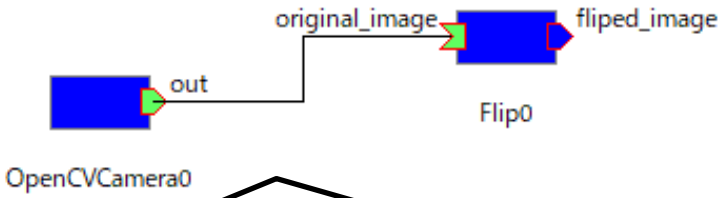

workspace3 - - Eclipse SDK
ファイル(F) 編集(E) ナビゲート(N) 検索(A) プロジェクト(P) 実行(R) ウィンドウ(W) ヘルプ(H)
Name Service View
RepositoryView
Flip
*System Editor
localhost
DESKTOP-9ESUPQB|host_cxt
CameraViewer0|rtc
Flip0|rtc
OpenCVCamera0|rtc
in
Key_out
Mouse_event
Mouse_X_pos
Mouse_Y_pos
CameraViewer0
originalImage
flippedImage
Flip0
out
OpenCVCamera0
左側のネームサービスビューからOpenCVCamera0.rtc、
CameraViewer0.rtc、Flip0.rtcをドラッグアンドドロップ

データポートの接続

OpenCVCamera0の「out」を選択して、Flip0の「original_image」にドラッグアンドドロップ

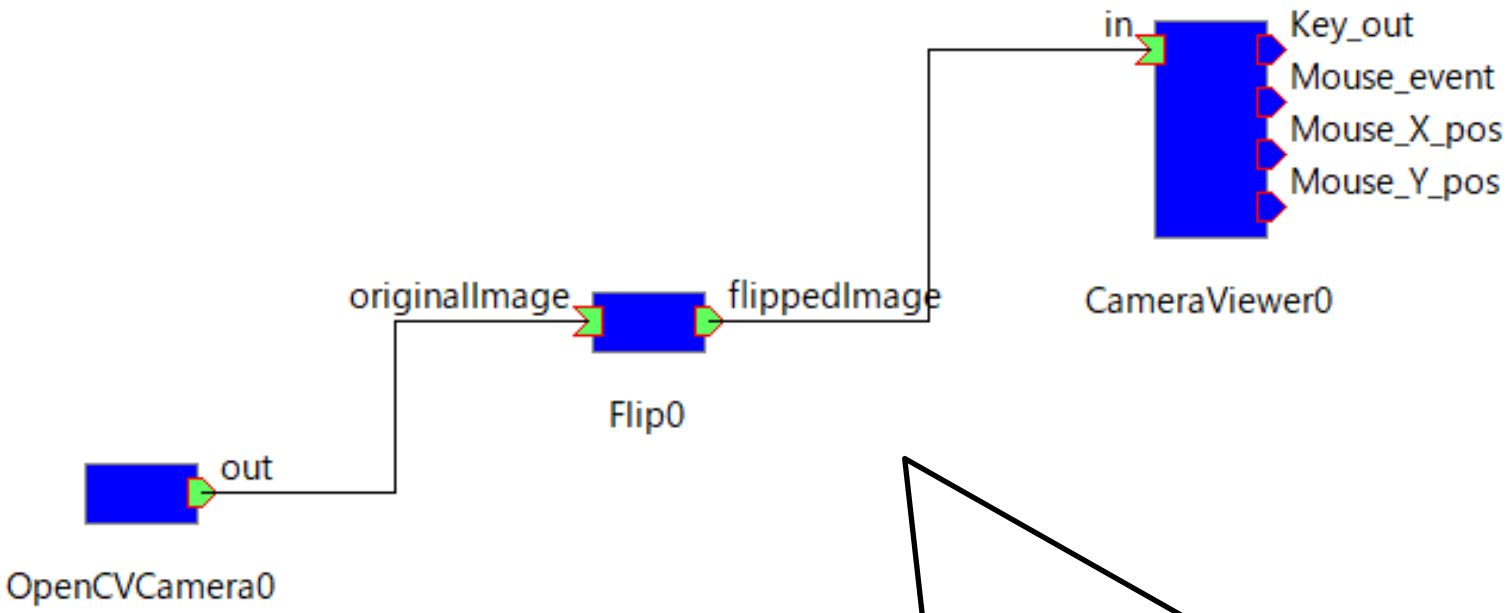


Connector Profileはそのままの設定でOKをクリック

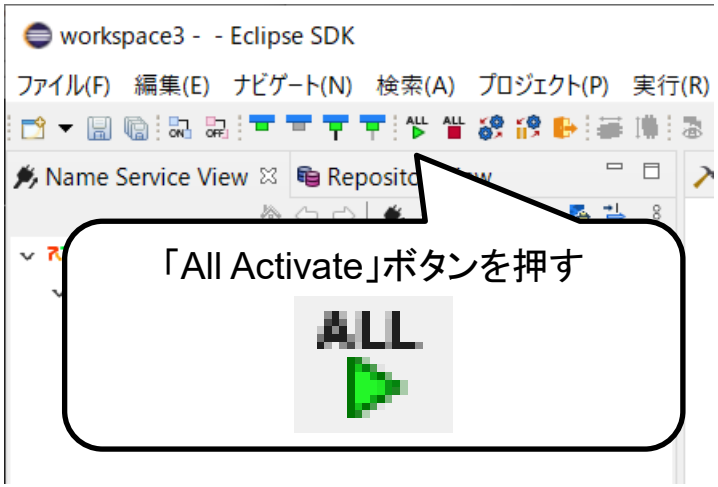


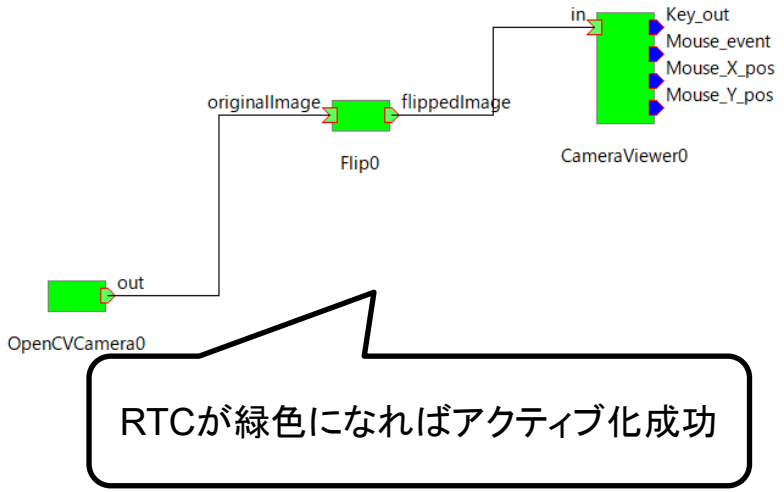
1. ポートの間に線が表示される
2. InPort、OutPortが緑色で表示される

データポートの接続


in
Key_out
Mouse_event
Mouse_X_pos
Mouse_Y_pos
CameraViewer0
originalImage
flippedImage
Flip0
out
OpenCVCamera0
Flip0の「flippedImage」とCameraViewer0と「in」を接続する

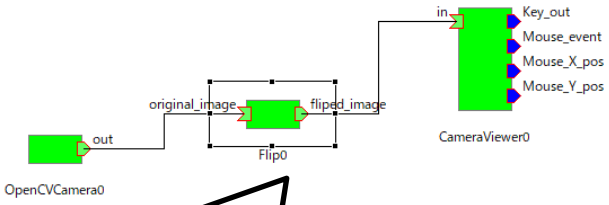
アクティブ化


workspace3 - - Eclipse SDK
ファイル(F) 編集(E) ナビゲート(N) 検索(A) プロジェクト(P) 実行(R)
Name Service View
「All Activate」ボタンを押す
ALL

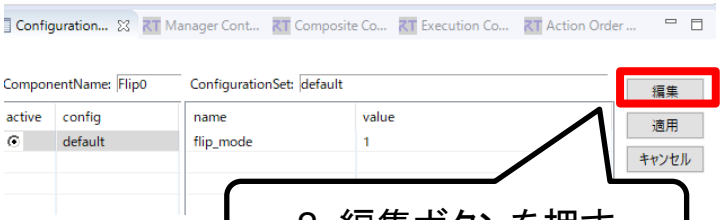

in
Key_out
Mouse_event
Mouse_X_pos
Mouse_Y_pos
originalImage
flippedImage
Flip0
CameraViewer0
out
OpenCVCamera0
RTCが緑色になればアクティブ化成功

コンフィギュレーションパラメータの操作

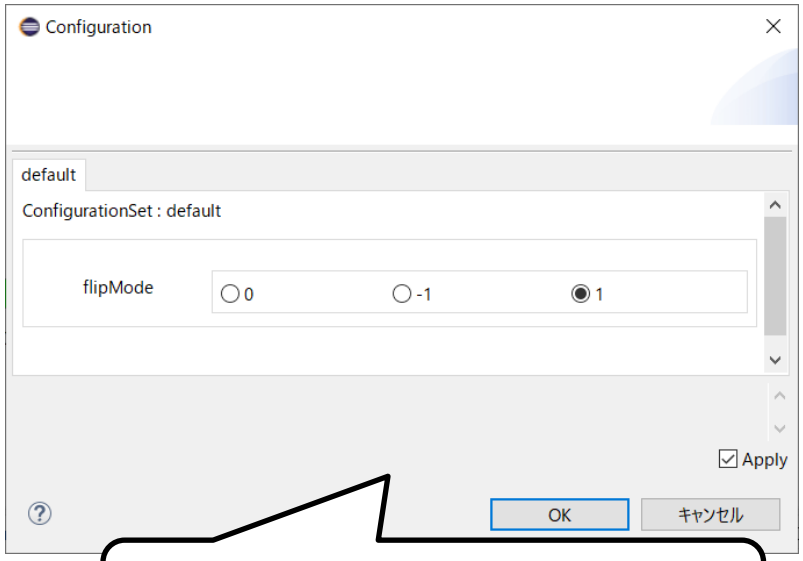
- コンフィギュレーションパラメータをRT System Editorから操作する



1. Flip0をクリックする



2. 編集ボタンを押す

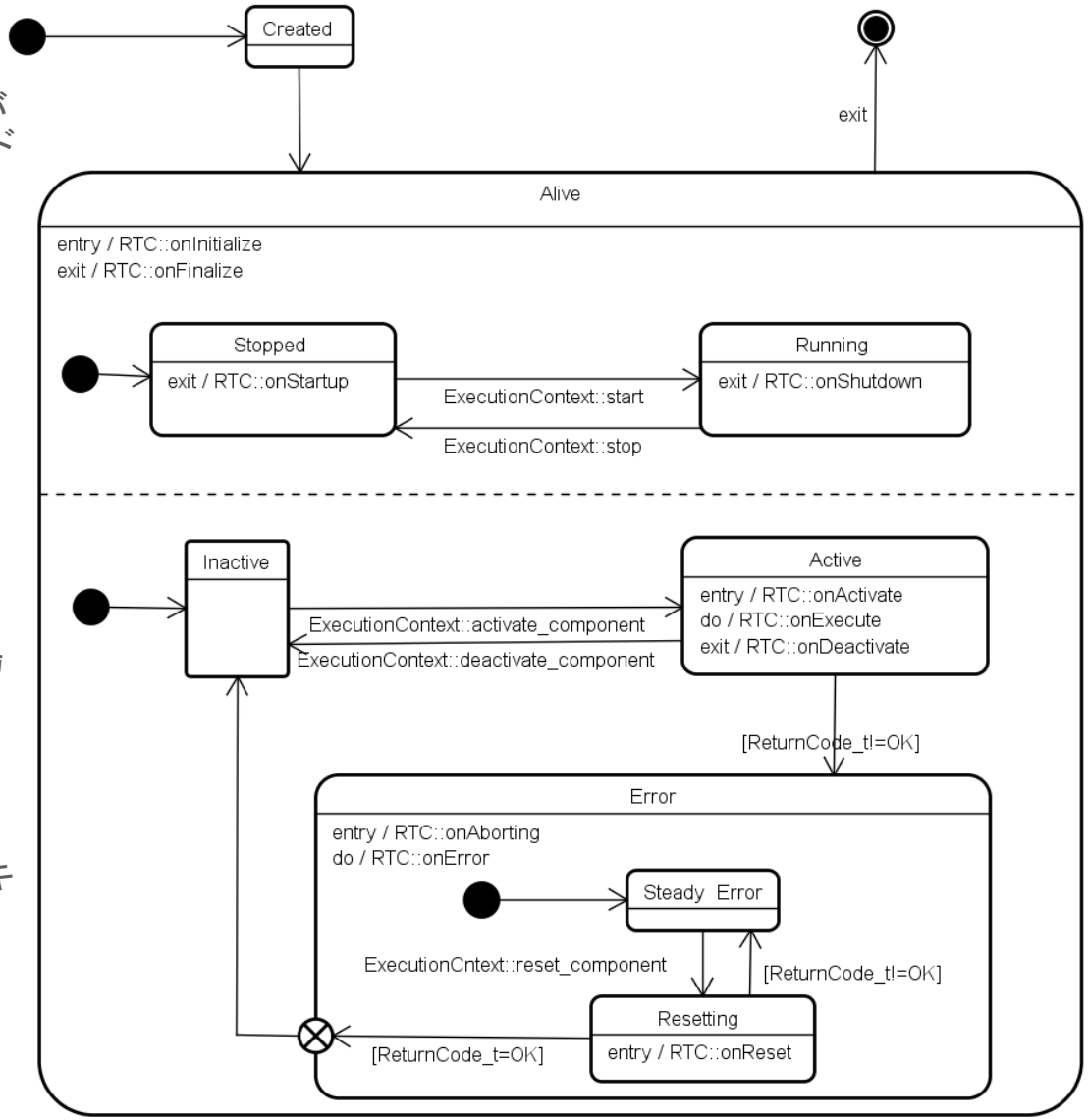


3. ラジオボタンを操作する

- 以下の動作ができるか確認
  - カメラ画像を表示するビュワーが表示されるか？
  - flipModeを変更することで画像の反転方向を切り替えられるか？

RTコンポーネントの状態遷移

- RTCには以下の状態が存在する
  - Created
    - 生成状態
    - 実行コンテキストを生成し、start()が呼ばれて実行コンテキストのスレッドが実行中(Runnning)状態になる
    - 自動的にInactive状態に遷移する
  - Inactive
    - 非活性状態
    - activate_componentメソッドを呼び出すと活性状態に遷移する
    - RT System Editor上での表示は青
  - Active
    - 活性状態
    - onExecuteコールバックが実行コンテキストにより実行される
    - リターンコードがRTC_OK以外の場合はエラー状態に遷移する
    - RT System Editor上での表示は緑
  - Error
    - エラー状態
    - onErrorコールバックが実行コンテキストにより実行される
    - reset_componentメソッドを呼び出すと非活性状態に遷移する
    - RT System Editor上での表示は赤
  - 終了状態

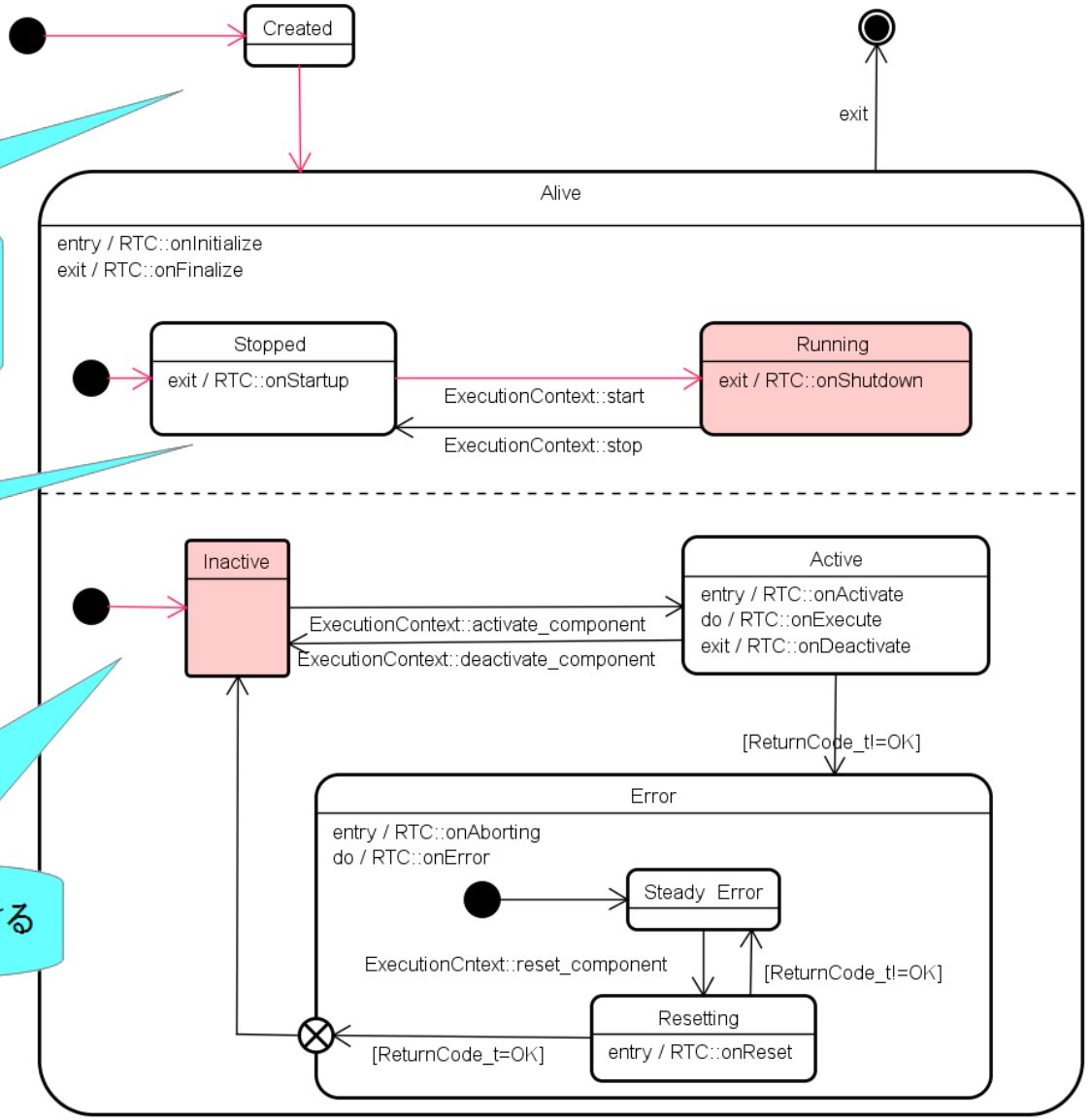


RTコンポーネントの状態遷移(生成直後)

最初にCreated状態に遷移して
実行コンテキストの生成等を行う
この時にonInitializeコールバックを実行する

startメソッドにより実行コンテキストが
Runnning状態に遷移する
このときonStartupコールバックが
呼び出される

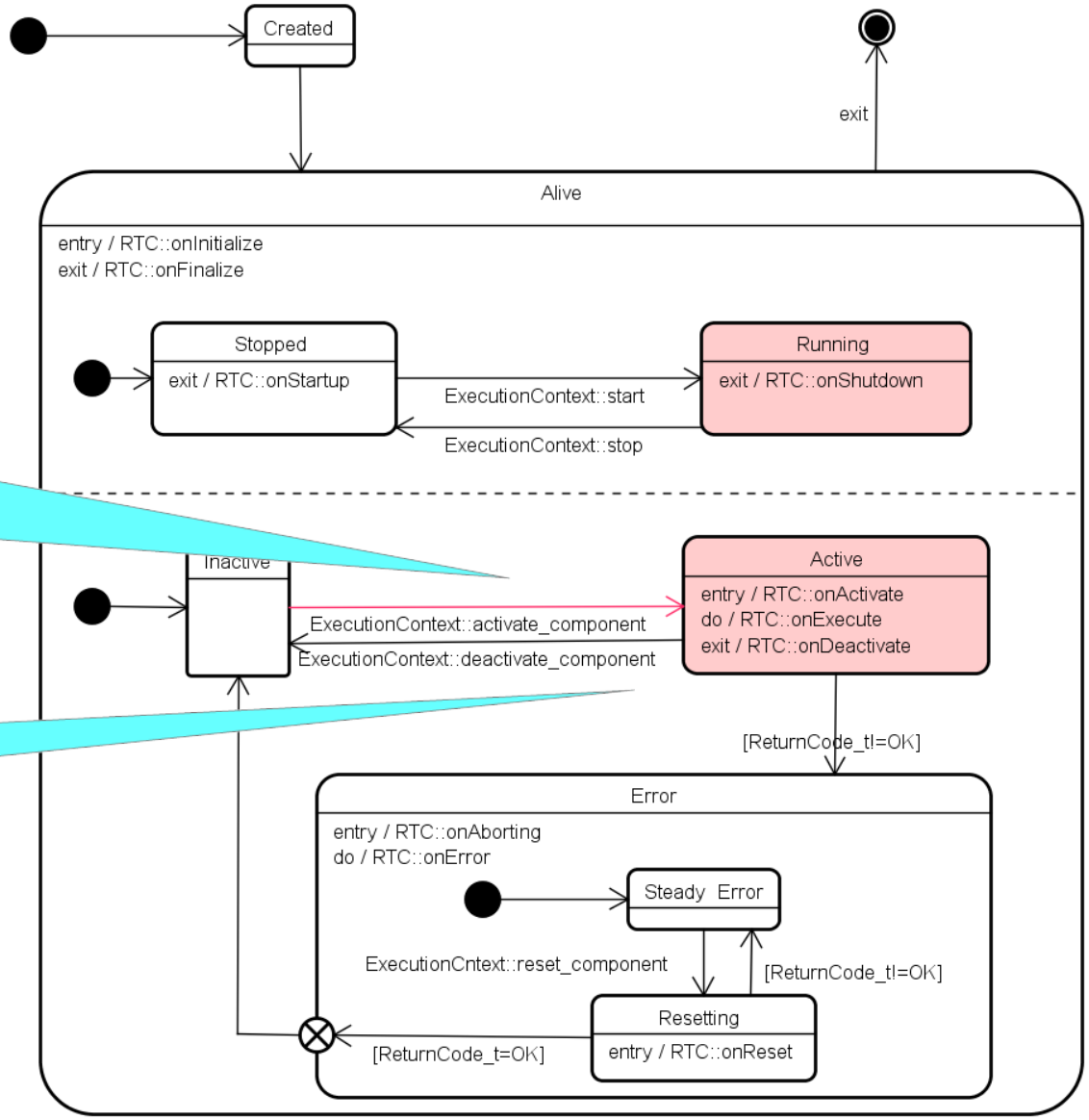
Created状態の次にInactive状態に遷移する



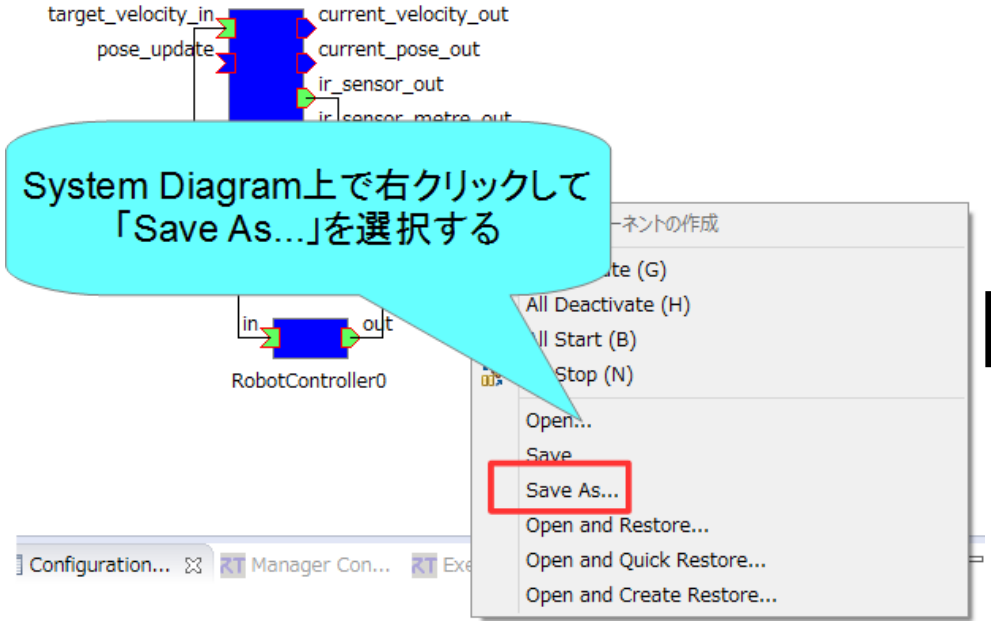
RTコンポーネントの状態遷移(アクティブ化)

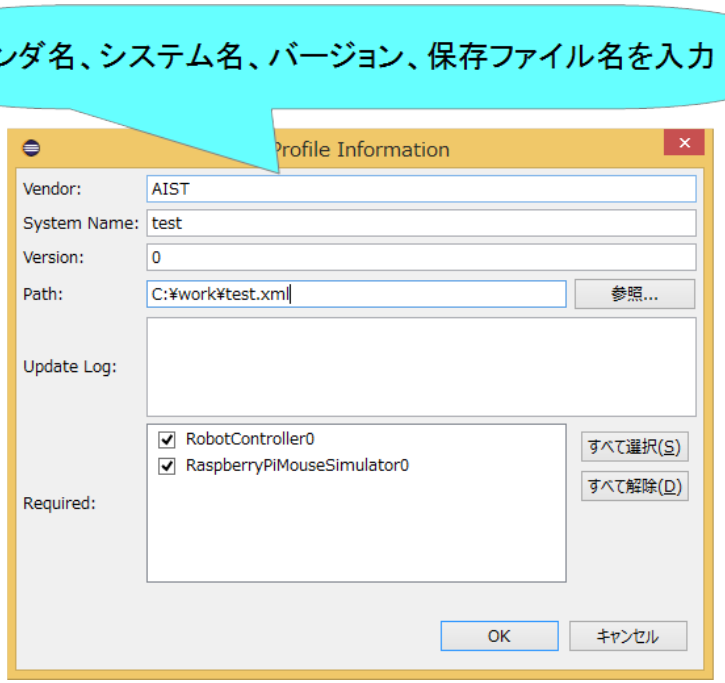
RTシステムエディタの操作によりRTコンポーネントのアクティブ化を行うとactivate_componentメソッドが呼び出される。
activate_componentメソッドによりコンポーネントがActive状態に遷移する。
この時onActivatedコールバックが実行される

周期実行の実行コンテキストの場合、
onExecuteコールバックが周期的に呼び出される。

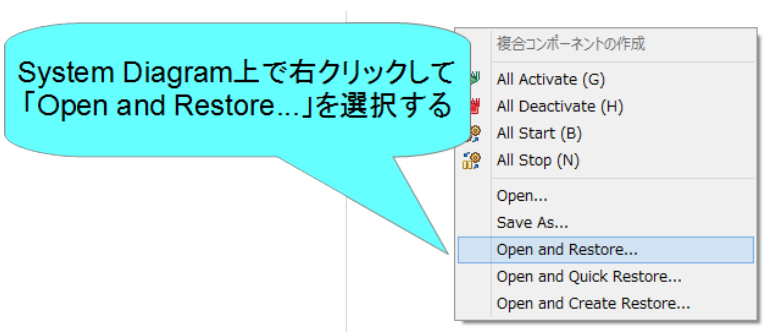


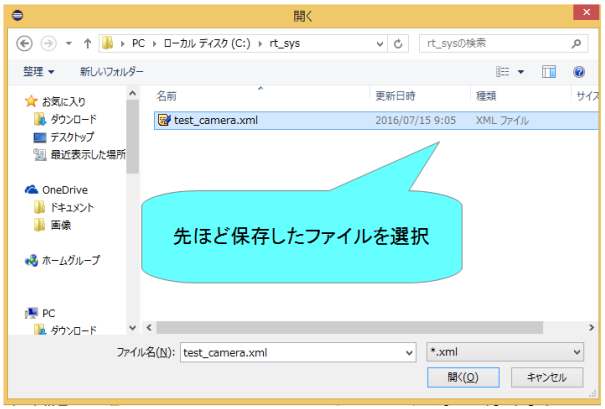
システムの保存

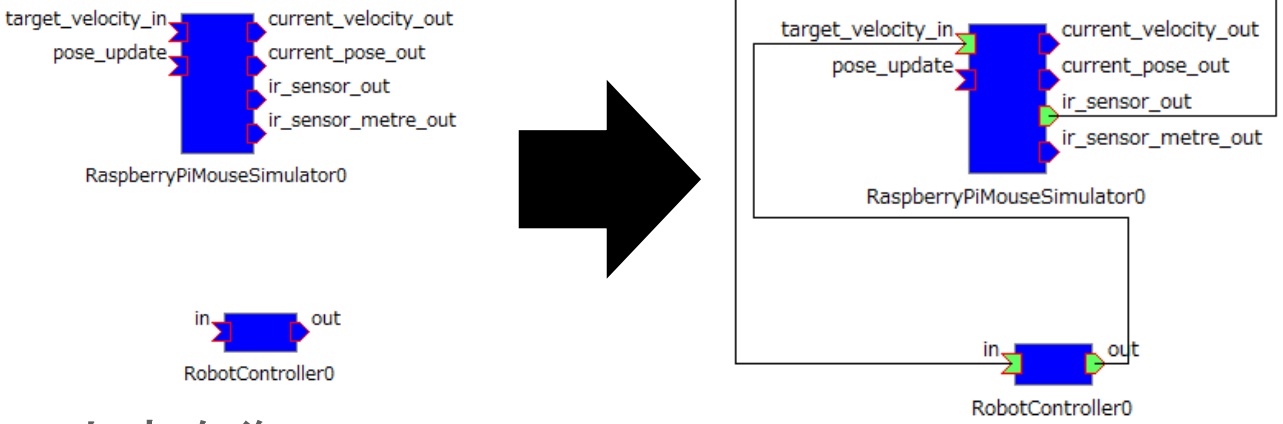

target_velocity_in
pose_update
current_velocity_out
current_pose_out
ir_sensor_out
System Diagram上で右クリックして
「Save As...」を選択する
in
out
RobotController0
All Deactivate (H)
Open...
Save
Save As...
Open and Restore...
Open and Quick Restore...
Open and Create Restore...
Configuration...
Manager Con...


ベンダ名、システム名、バージョン、保存ファイル名を入力
Profile Information
Vendor:
AIST
System Name:
test
Version:
0
Path:
C:¥work¥test.xml
参照...
Update Log:
RobotController0
RaspberryPiMouseSimulator0
すべて選択(S)
すべて解除(D)
Required:
OK
キャンセル

システムの復元





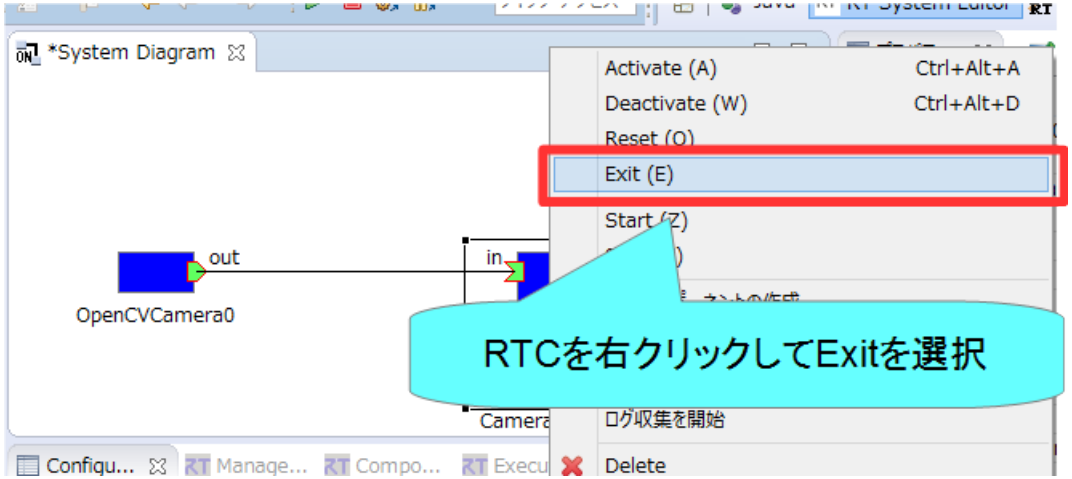


- 以下の内容を復元
  - ポート間の接続
  - コンフィギュレーション
  - 「Open and Create Restore」を選択した場合はマネージャからコンポーネント起動

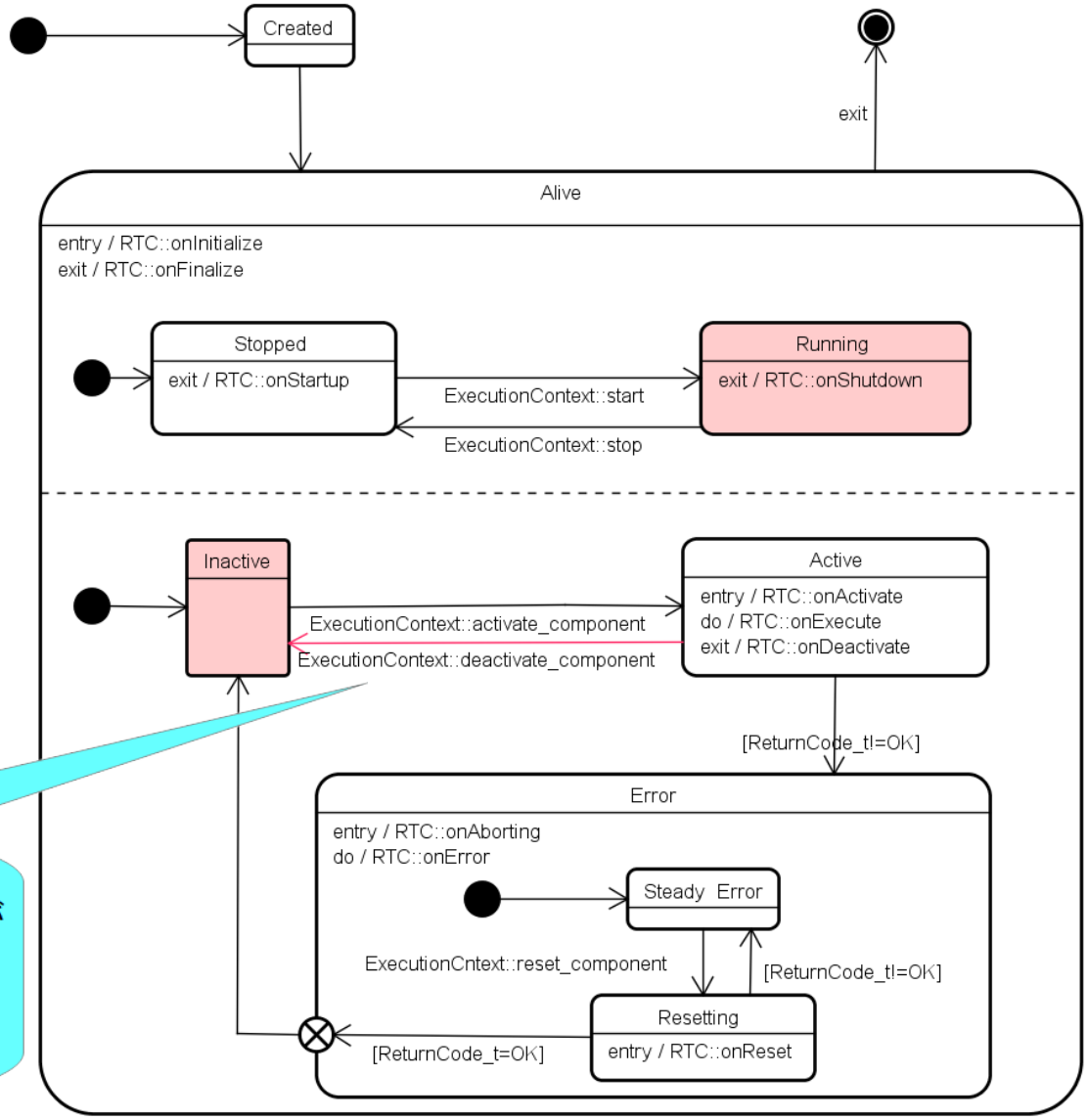
非アクティブ化、終了

- 非アクティブ化

- 終了



RTコンポーネントの状態遷移(非アクティブ化)



RTシステムエディタの操作によりRTコンポーネントの非アクティブ化を行うとdeactivate_componentメソッドが呼び出される。
deactivate_componentメソッドによりコンポーネントがInactive状態に遷移する。
この時onDeactivatedコールバックが実行される