

RTコンポーネント作成入門

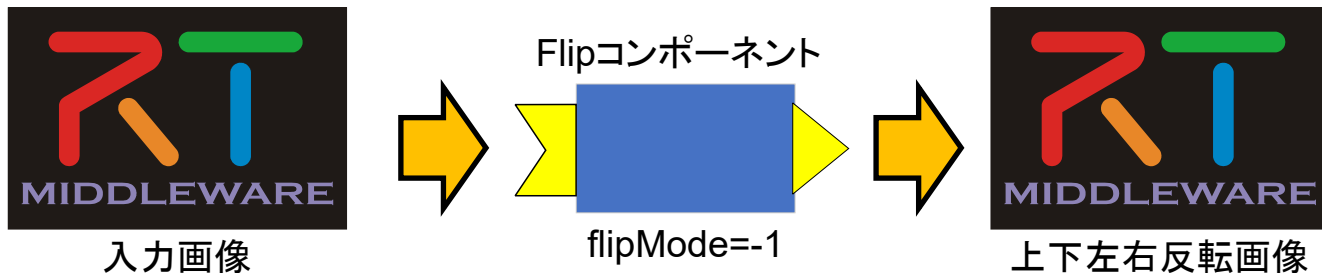
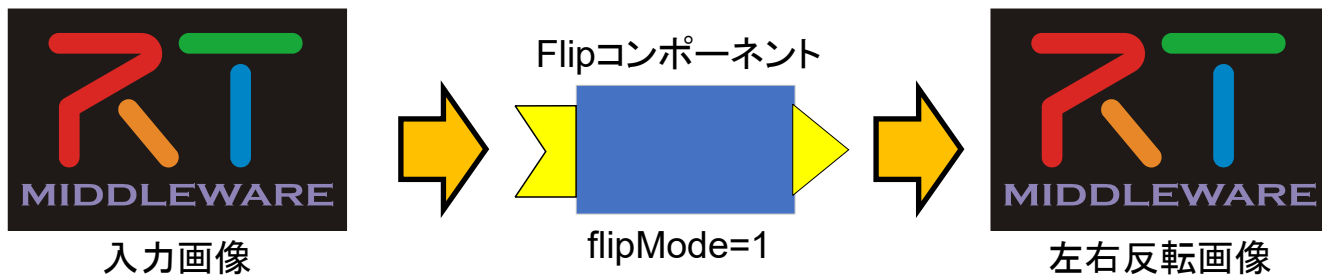
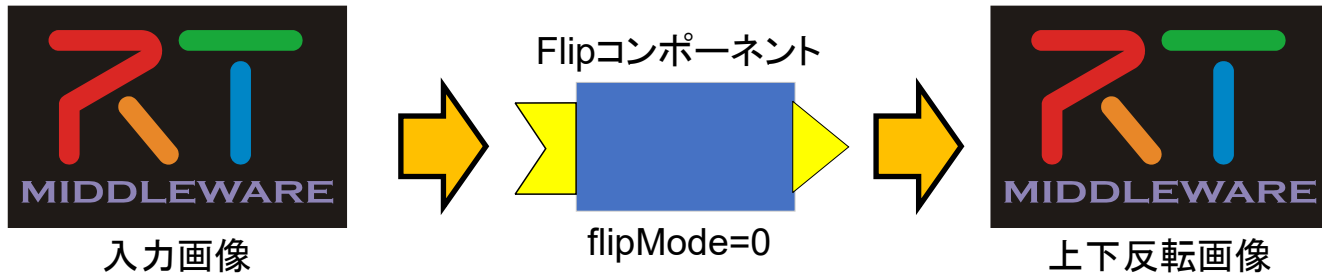
宮本 信彦

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

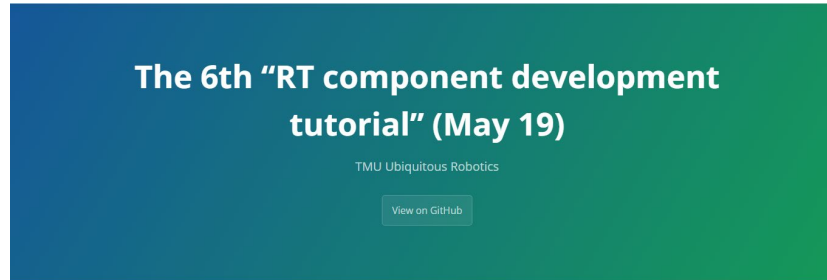
- 入力画像を反転して出力するFlipコンポーネントを作成する。



実習内容

- 資料

- <https://sealbreeder.github.io/TMU-Ubiquitous-Robotics/240508>



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>



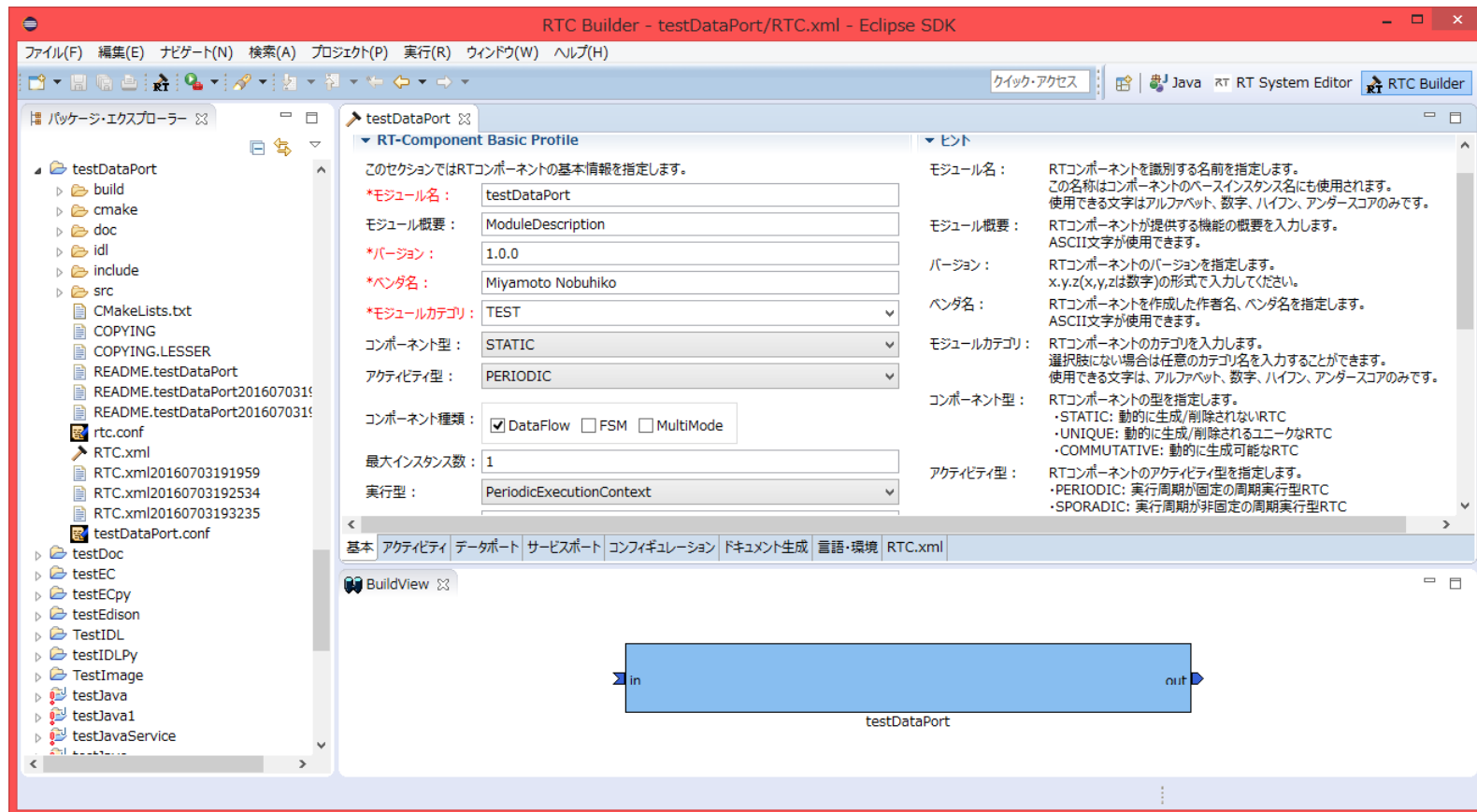
全体の手順

- 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

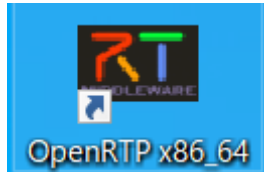
RTC Builder

- 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



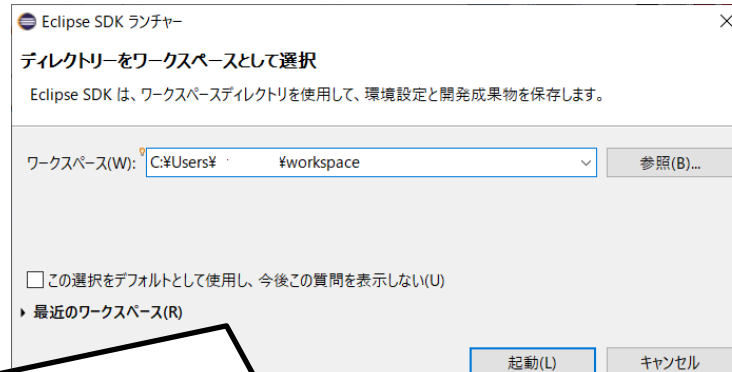
RTC Builderの起動

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



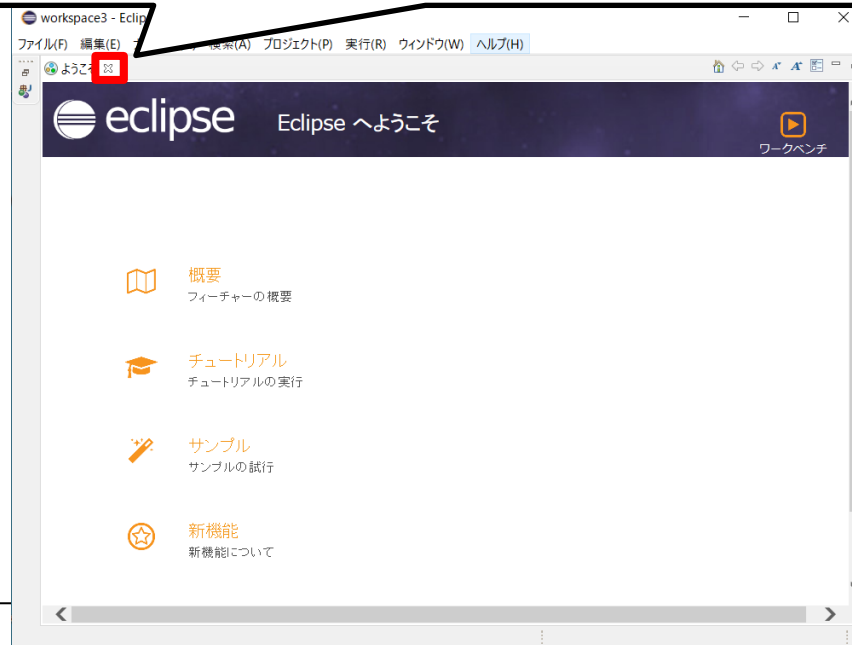
- Ubuntu
 - 以下のコマンドを入力
 - \$ openrtp2

RTC Builderの起動



ワークスペースに適切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

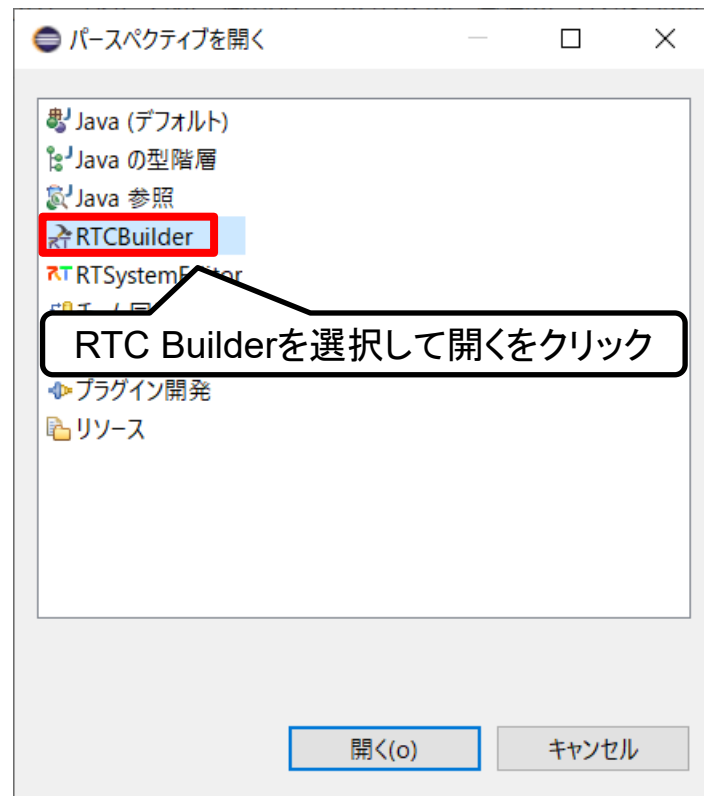
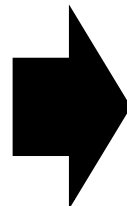
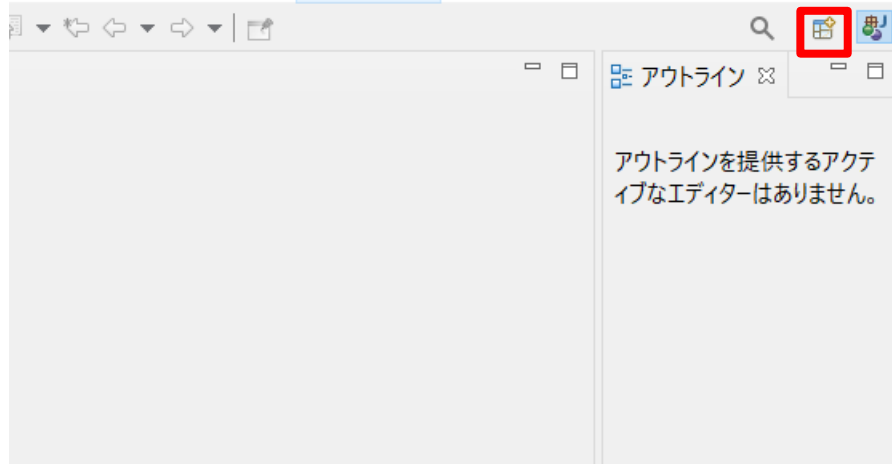


RTC Builderの起動

右上の「パースペクティブを開く」ボタンをクリック

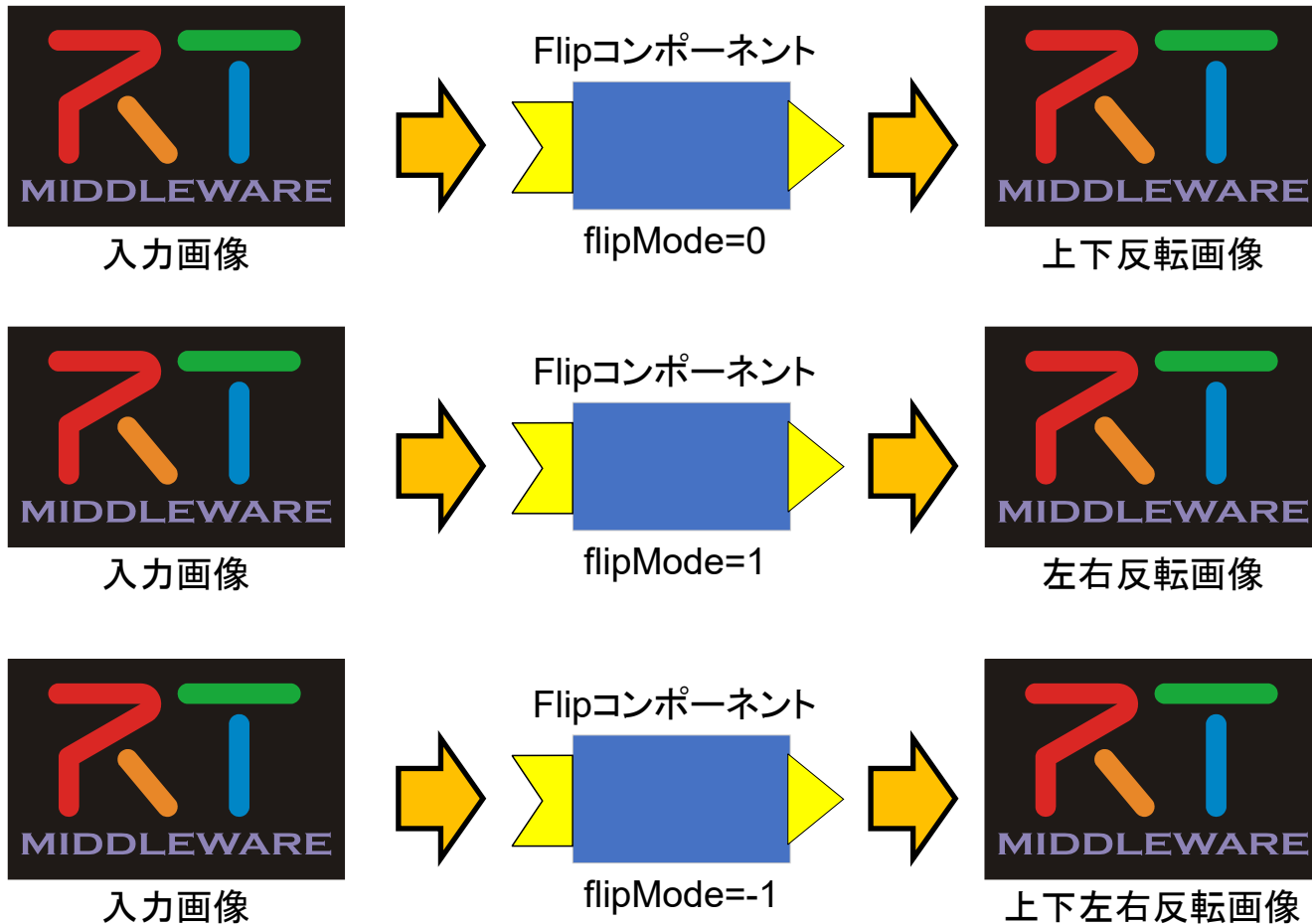


A) プロジェクト(P) 実行(R) ウィンドウ(W) ヘルプ(H)

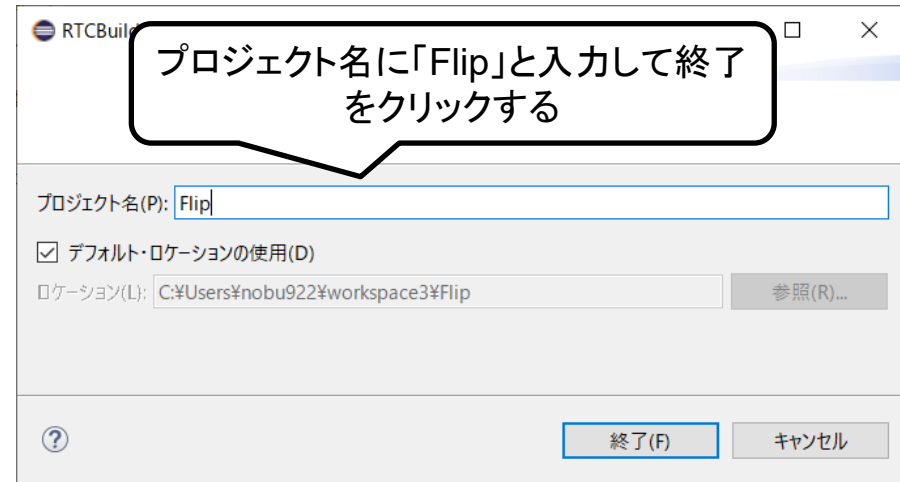
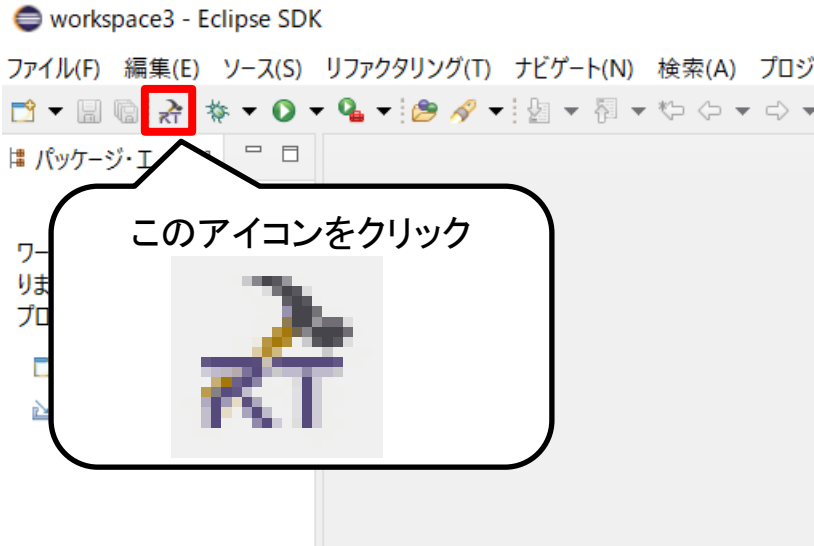


プロジェクト作成

- Flipコンポーネントのスケルトンコードを作成する。
 - CameralImage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

基本プロフィールの入力

- 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

RTCプロフィールエディタ
ここに各項目を入力する

ヒント

Flip 88

基本

▼ RT-Component Basic Profile

このセクションではRTコンポーネントの基本情報を指定します。

*モジュール名: ModuleName

モジュール概要: ModuleDescription

*バージョン: 1.0.0

*ベンダ名: Miyamoto Nobuhiko

*モジュールカテゴリ: TEST

コンポーネント型: STATIC

アクティビティ型: PERIODIC

コンポーネント種類: ☒ DataFlow ☐ FSM ☐ MultiMode

最大インスタンス数: 1

▼ ヒント

モジュール名: RTコンポーネントを識別する名前を指定します。この名称はコンポーネントのベースインスタンス名にも使用されます。使用できる文字はアルファベット、数字、ハイフン、アンダースコアのみです。

モジュール概要: RTコンポーネントが提供する機能の概要を入力します。ASCII文字が使用できます。

バージョン: RTコンポーネントのバージョンを指定します。x.y.z(x,y,zは数字)の形式で入力してください。

ベンダ名: RTコンポーネントを作成した作者名、ベンダ名を指定します。ASCII文字が使用できます。

モジュールカテゴリ: RTコンポーネントのカテゴリを入力します。選択されていない場合は任意のカテゴリ名を入力することができます。使用できる文字は、アルファベット、数字、ハイフン、アンダースコアのみです。

コンポーネント型: RTコンポーネントの型を指定します。
・STATIC: 動的に生成/削除されないRTC
・UNIQUE: 動的に生成/削除されるユニークなRTC
・COMMUTATIVE: 動的に生成可能なRTC

アクティビティ型: RTコンポーネントのアクティビティ型を指定します。

基本 | アクティビティ | データポート | サービスポート | コンフィギュレーション | ドキュメント生成 | 言語・環境 | RTC.xml

「基本」タブを選択

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

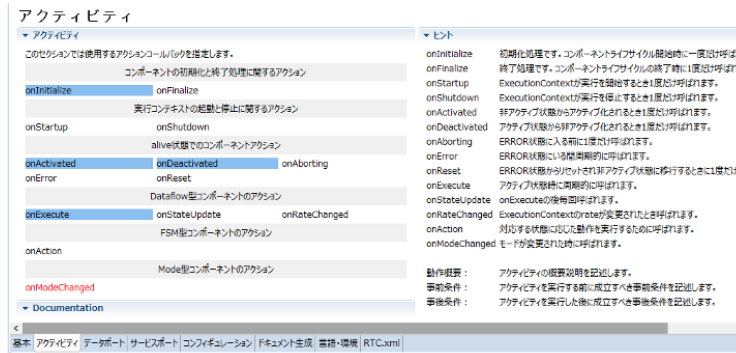
▼ RT-Component Basic Profile

このセクションではRTコンポーネントの基本情報を指定します。

*コンポーネント名 :	Flip
概要 :	Flip image component
*バージョン :	1.0.0
*ベンダ名 :	TMU
*カテゴリ :	ImageProcessingg ▼
コンポーネント型 :	STATIC ▼
アクティビティ型 :	PERIODIC ▼
最大インスタンス数 :	1
実行型 :	PeriodicExecutionContext ▼
実行周期 :	1000.0
概要 :	<div></div> ▲ ▼
RTC Type :	

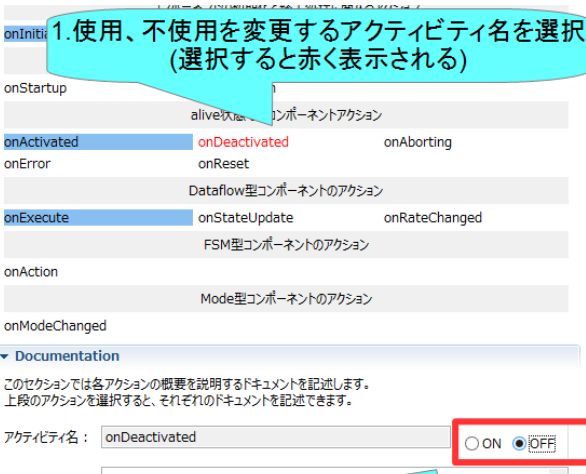
アクティビティの設定

- 使用するアクティビティを設定する

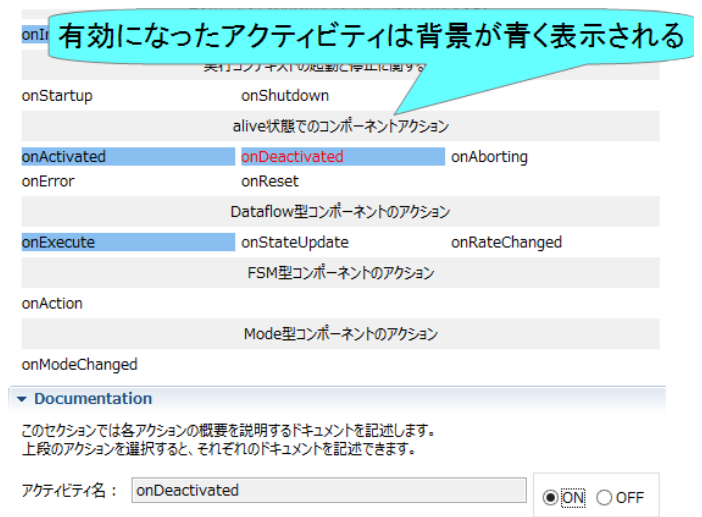


「アクティビティ」タブを選択

- 指定アクティビティを有効にする手順



2. アクティビティ名の選択後、ON・OFFを選択する

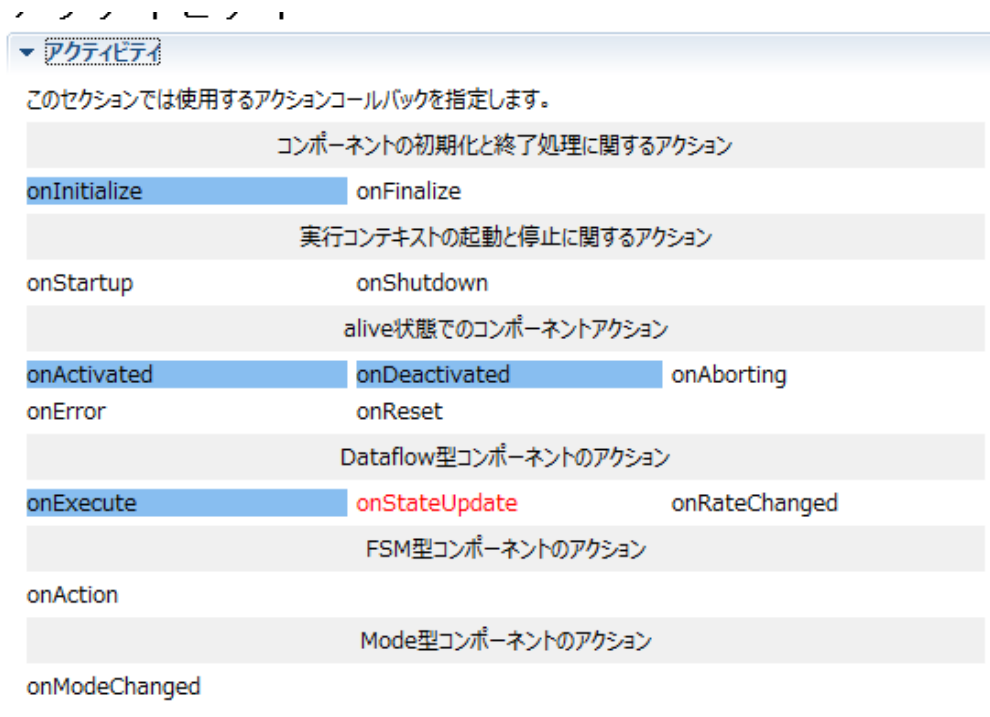


アクティビティの設定

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



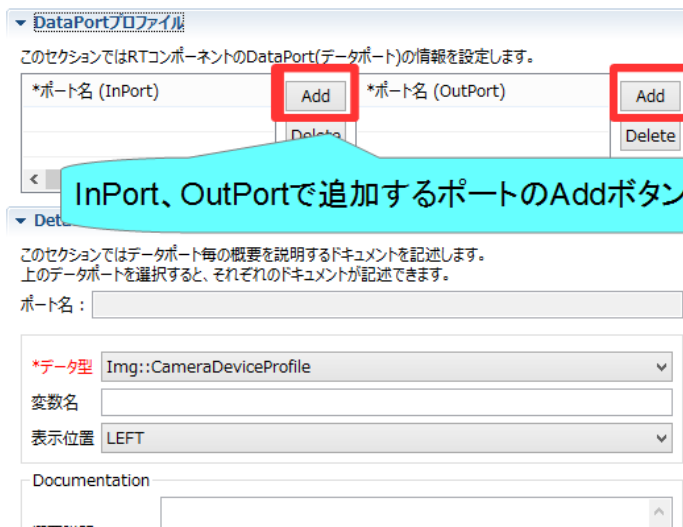
データポートの設定

- InPort、OutPortの追加、設定を行う

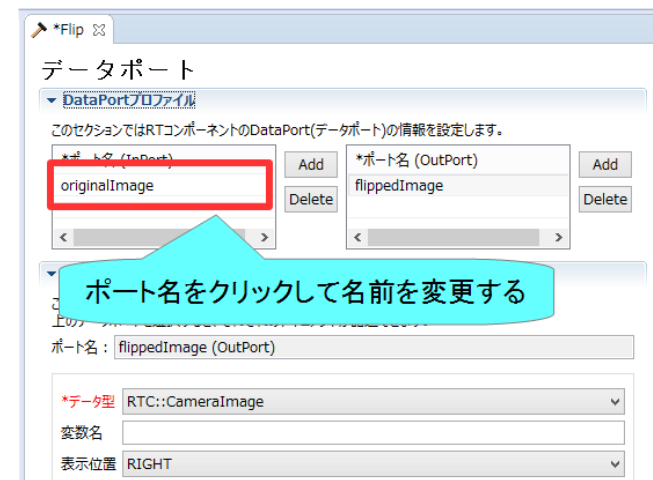


「データポート」タブを選択

- データポートを追加する手順



InPort、OutPortで追加するポートのAddボタンをクリック



ポート名をクリックして名前を変更する

各項目を設定する

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

The screenshot shows the configuration interface for data ports in the RT Middleware. At the top, there are two tables for InPort and OutPort. The InPort table has one entry 'in' and the OutPort table has one entry 'out'. Below these tables is a 'Detail' section. The 'Detail' section has a dropdown menu for 'ポート名' (Port Name) set to 'out (OutPort)'. Below this, there is a list of data types for the selected port. The list includes 'RTC::TimedVelocity2D', 'RTC::TimedULongSeq', 'RTC::TimedUShort', 'RTC::TimedUShortSeq', 'RTC::TimedVector2D', 'RTC::TimedVector3D', 'RTC::TimedVelocity2D', and 'RTC::TimedVelocity3D'. A red box highlights the dropdown arrow next to the first item, 'RTC::TimedVelocity2D'. A blue arrow points from a text box to this dropdown arrow.

*ポート名 (InPort)	Add	Delete
in		

*ポート名 (OutPort)	Add	Delete
out		

▼ Detail

このセクションではデータポート毎の概要を説明するドキュメントを記述します。
上のデータポートを選択すると、それぞれのドキュメントが記述できます。

ポート名: out (OutPort)

*データ型: RTC::TimedVelocity2D

IDLファイル:

変数名:

表示位置:

基本 アクティブ

Reload

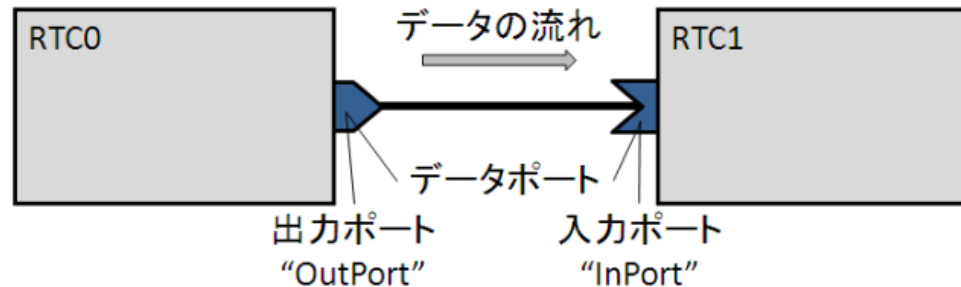
Browse...

環境 RTC.xml

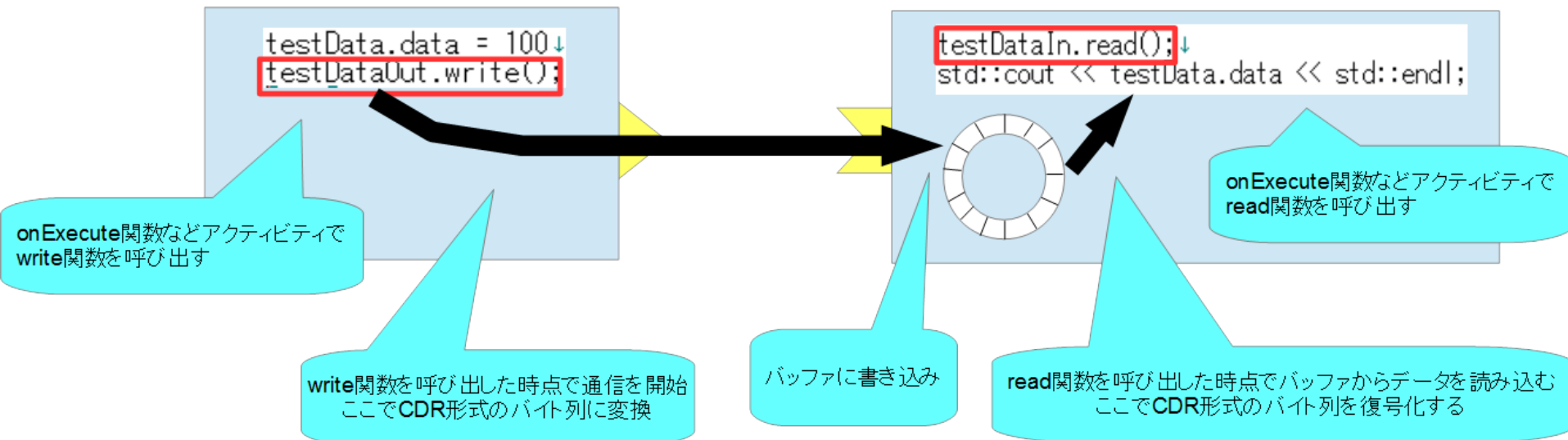
データ型はドロップダウン
リストから選択する

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

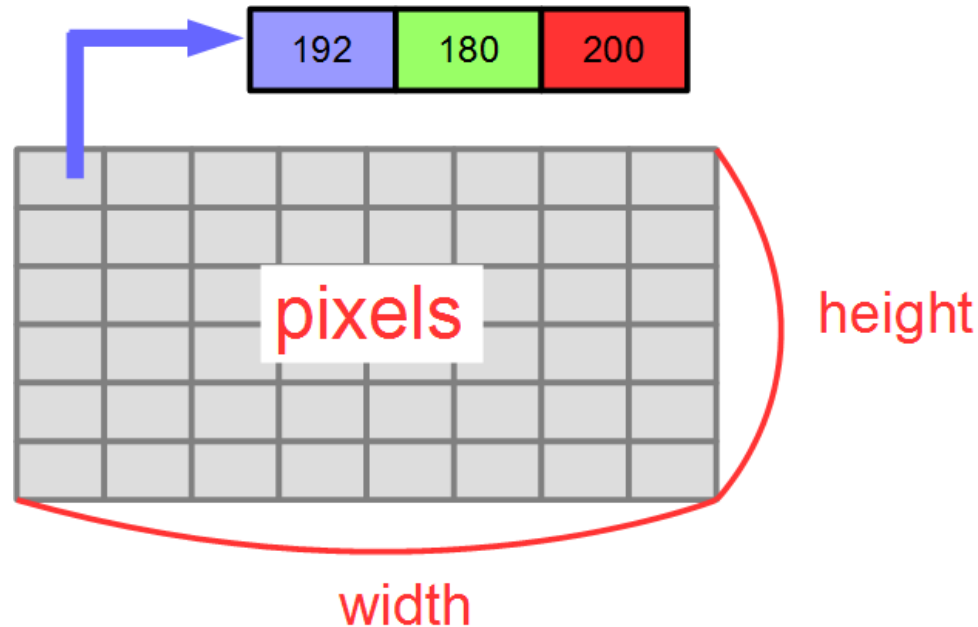


- 以下の例はデータフロー型がpush、サブスクリプション型がflush、インターフェース型がcorba_cdrの場合



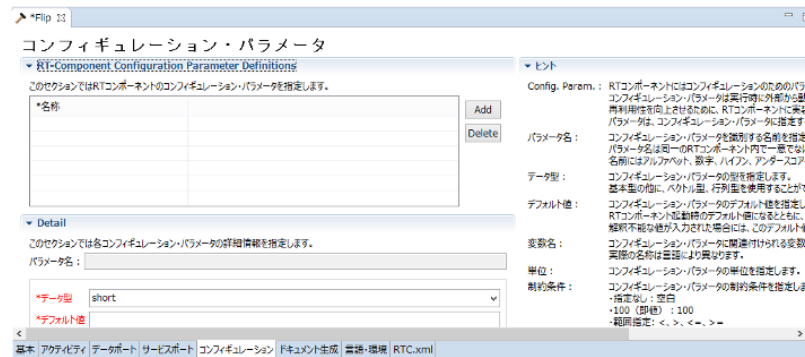
RTC::Camera型について

- 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



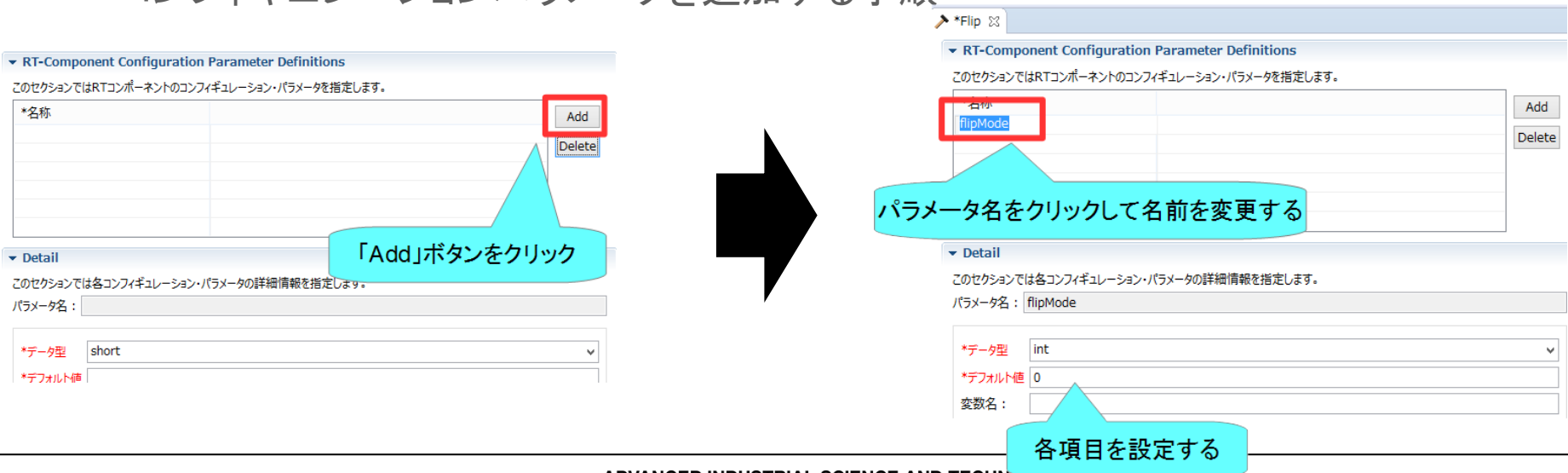
コンフィギュレーションの設定

- 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



「Add」ボタンをクリック

パラメータ名をクリックして名前を変更する

各項目を設定する

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

コンフィギュレーション・パラメータ

RT-Component Configuration Parameter Definitions

このセクションではRTコンポーネントのコンフィギュレーション・パラメータを指定します。

*名称		Add
flipMode		Delete

Detail

このセクションでは各コンフィギュレーション・パラメータの詳細情報を指定します。

パラメータ名: flipMode

*データ型	int
*デフォルト値	0
変数名:	
単位:	
制約条件:	(0, -1, 1)
Widget:	radio

GUI(ラジオボタン)にて反転方向の操作ができるようにする

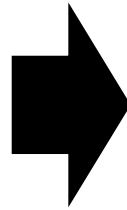
flipMode

<input type="radio"/> 0	<input type="radio"/> -1	<input checked="" type="radio"/> 1
-------------------------	--------------------------	------------------------------------

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

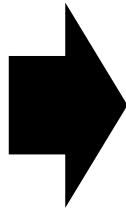
- Widget: text



- 制約条件: $0 \leq x \leq 100$
- Widget: spin
- Step: 10

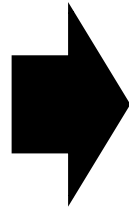


- 制約条件: $0 \leq x \leq 100$
- Widget: slider
- Step: 10

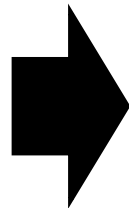


コンフィギュレーションパラメータの制約、Widgetの設定

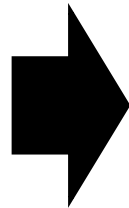
- 制約条件: (0,1,2,3)
- Widget: radio

☐ 0 ☐ 1 ☒ 2
☐ 3

- 制約条件: (0,1,2,3)
- Widget: checkbox

☒ 0 ☐ 1 ☒ 2
☐ 3

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



0
1
2
3

>
<

0
1

^
v

ドキュメントの設定

- 各種ドキュメント情報を設定

RobotController

ドキュメント生成

▼ コンポーネント概要

概要説明：

入出力：

アルゴリズムなど：

▼ その他

作成者・連絡先：

ライセンス、使用条件：

▼ ヒント

コンポーネント概要：コンポーネントに関する概要説明を記述します。
その他：コンポーネントに関する付加的な情報を記述します。
作成者・連絡先：name <mail address> の書式で入力します。
名前[name]はローマ字表記で入力します。
メールアドレスは<>記号で括弧する必要があります。

基本 | アクティビティ | データポート | サービスポート | コンフィギュレーション | **ドキュメント生成** | 言語・環境 | RTC.xml

「ドキュメント生成」タブを選択

- 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

- 実装する言語，動作環境に関する情報を設定

*Flip

実行周期 : 1000.0

概要 :

RTC Type :

▼ 言語

このセクションでは使用する言語を指定します

☒ C++

☐ Java

☐ Lua

☐ Python

言語を選択する
今回は「C++」を選択

▼ コード生成

コードの生成を行います。

☐ コード生成 ☐ Choreonoid用コードの生成

▼ プロファイル情報のインポート・エクスポート

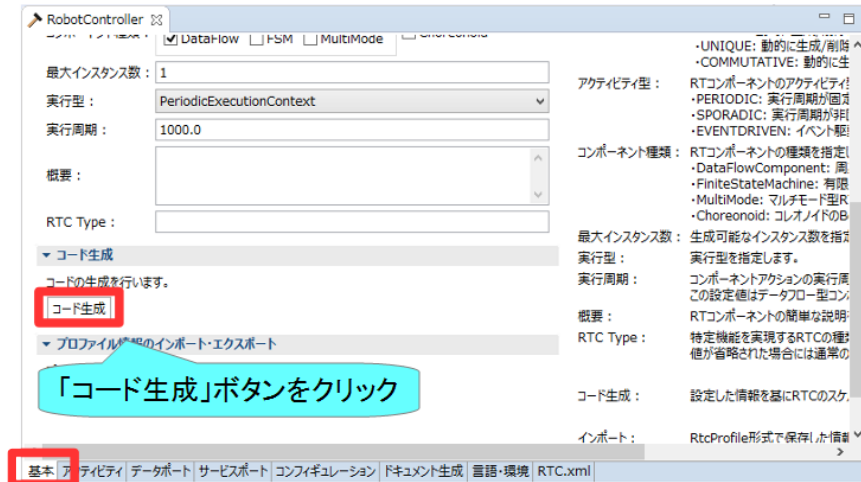
「基本」タブを選択

インポート エクスポート

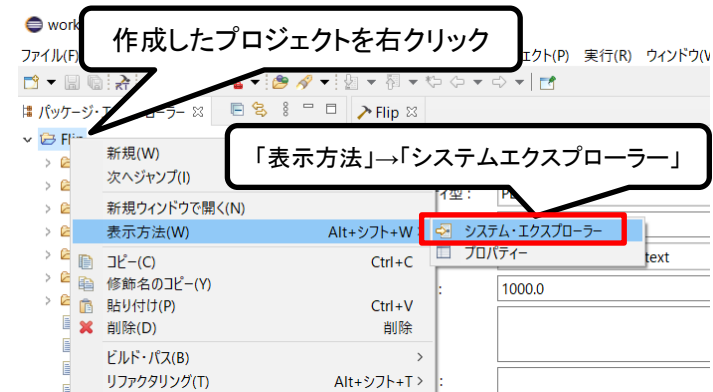
基本 アクティビティ FSM データポート サービスポート コンフィギュレーション ドキュメント生成 RTC.xml

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

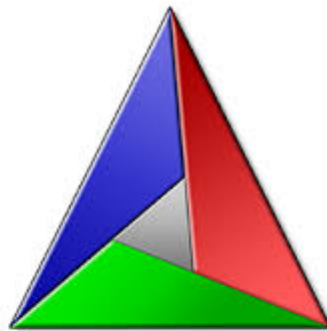
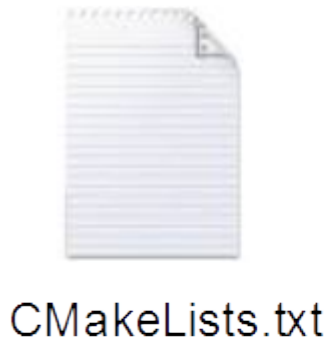
ソースコードの編集、RTCのビルド

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



Visual Studio
(ソリューションファイル、
プロジェクトファイル等)



CMake



Makefile

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)
```

4行目あたりに追加

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

target_link_libraries に\${OpenCV_LIBS}を追加

```
add_dependencies(${PROJECT_NAME} ALL_IDL_TGT)
```

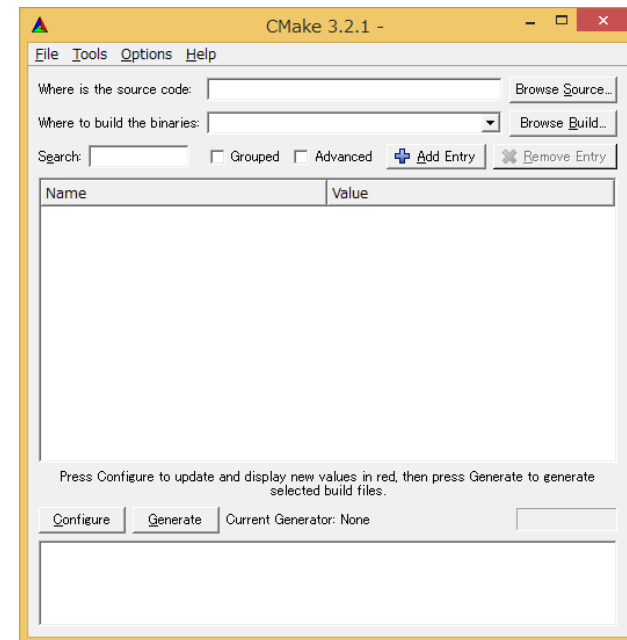
```
target_link_libraries(${PROJECT_NAME} ${OPENRTM_LIBRARIES} ${OpenCV_LIBS})
```

```
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

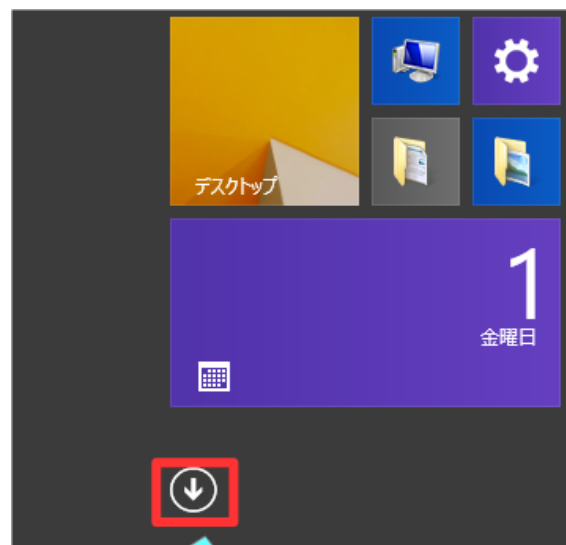
- Windows 8.1

デスクトップ



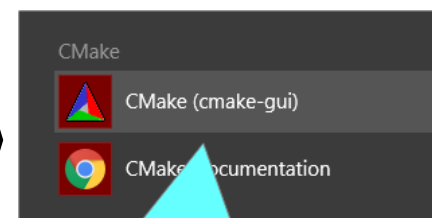
画面左下のアイコンをクリック

スタート画面



画面左下の矢印をクリック

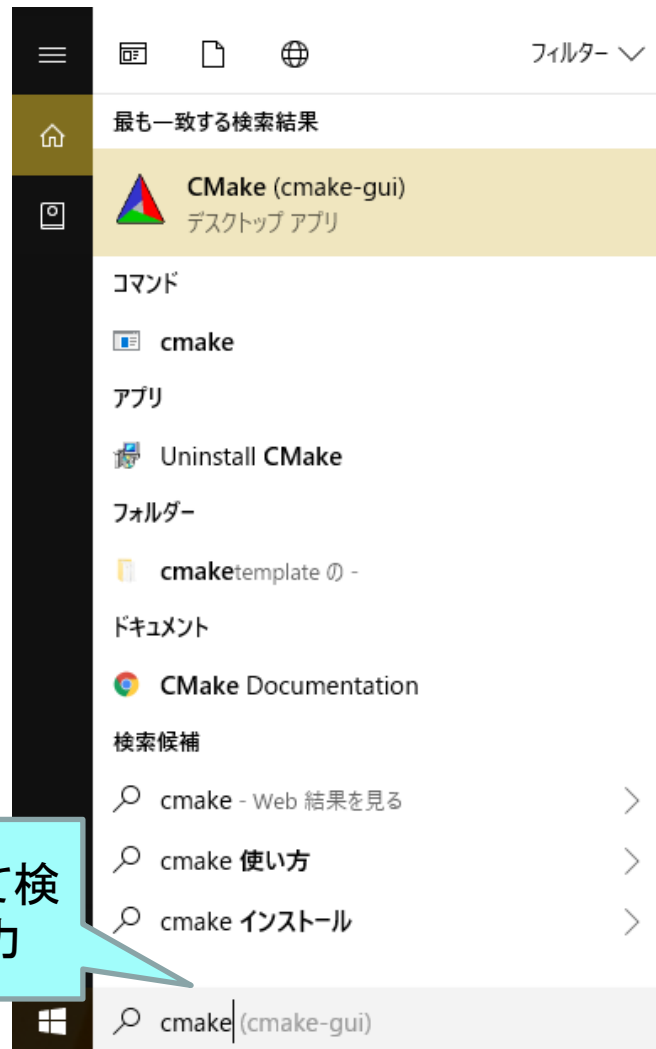
アプリビュー



CMake(cmake-gui)をクリック

cmake-guiの起動

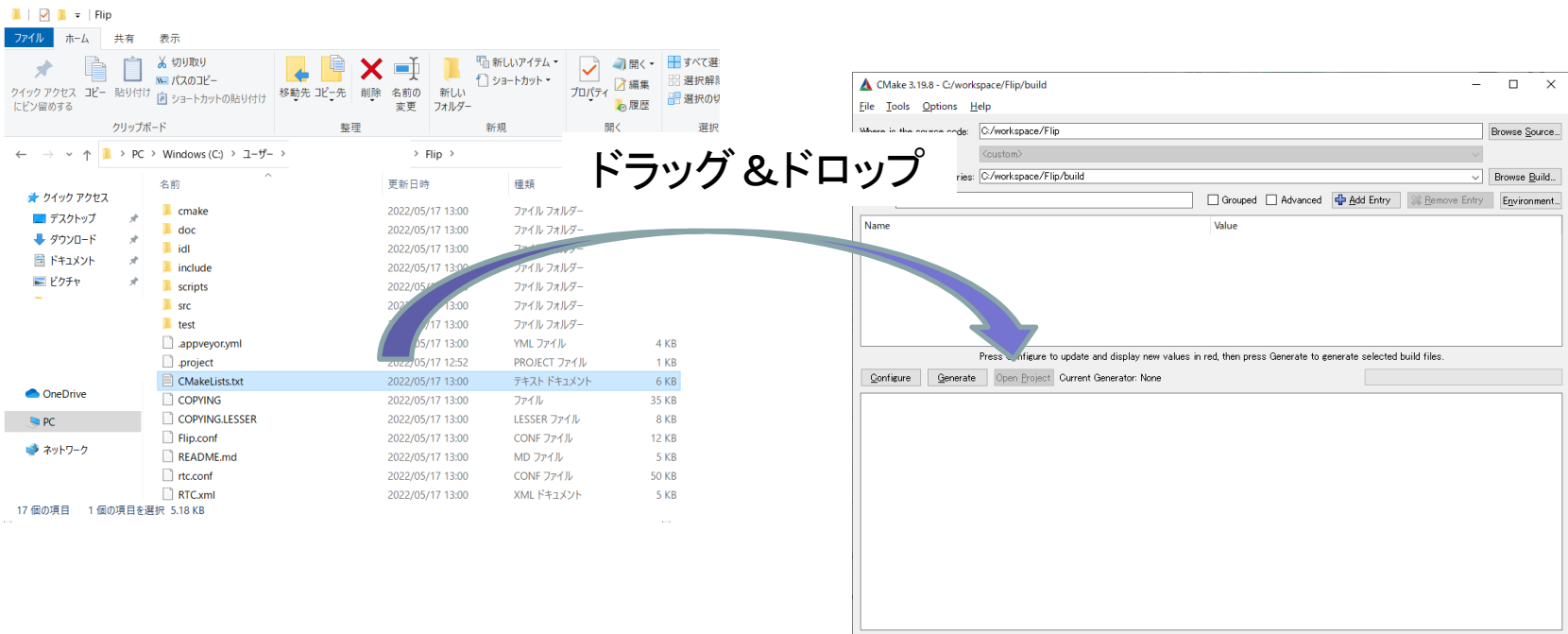
- Windows 10、11



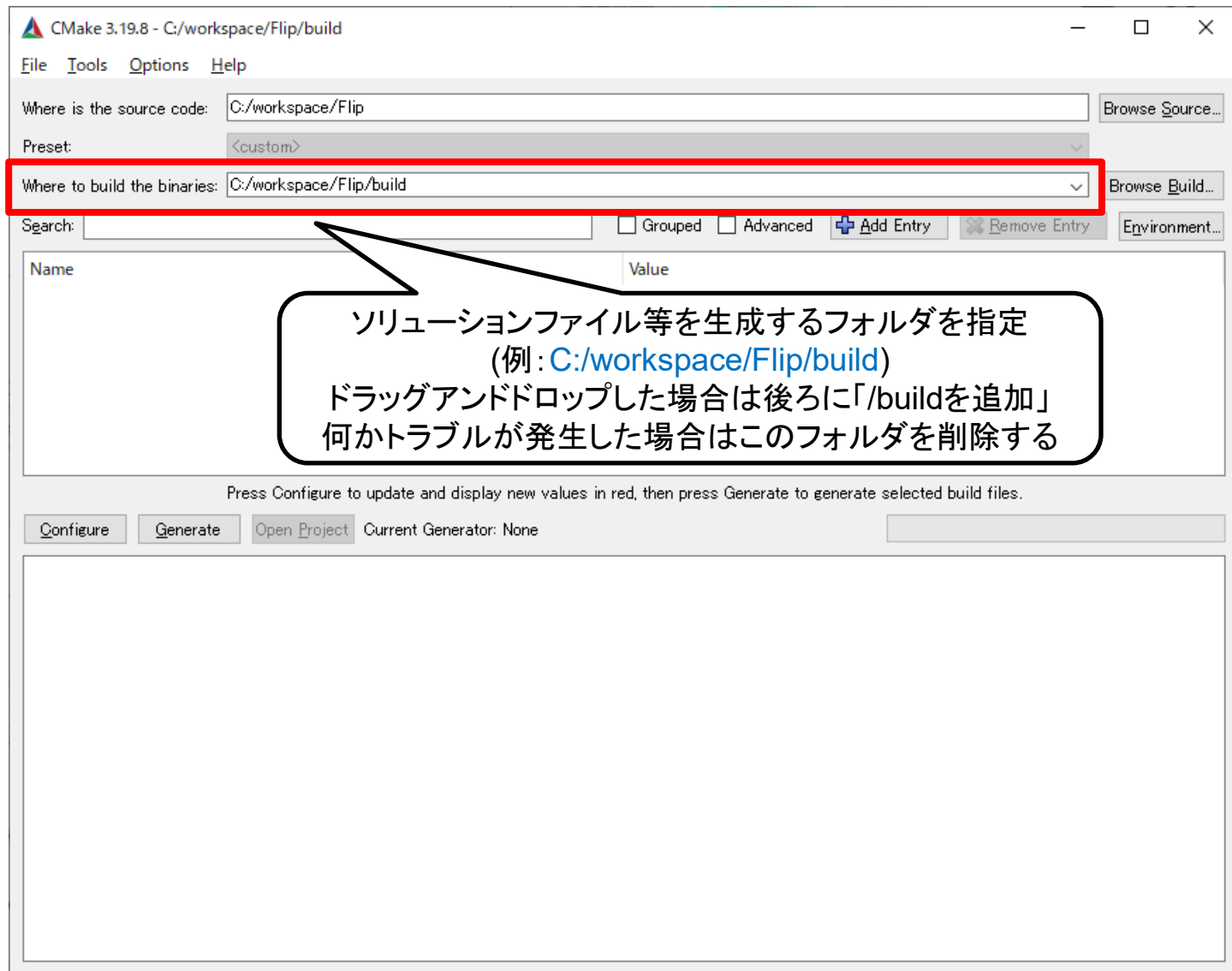
左下の「ここに入力して検索」に「cmake」と入力

ビルドに必要なファイルの生成

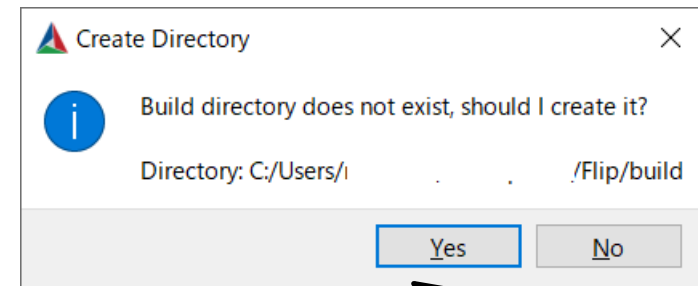
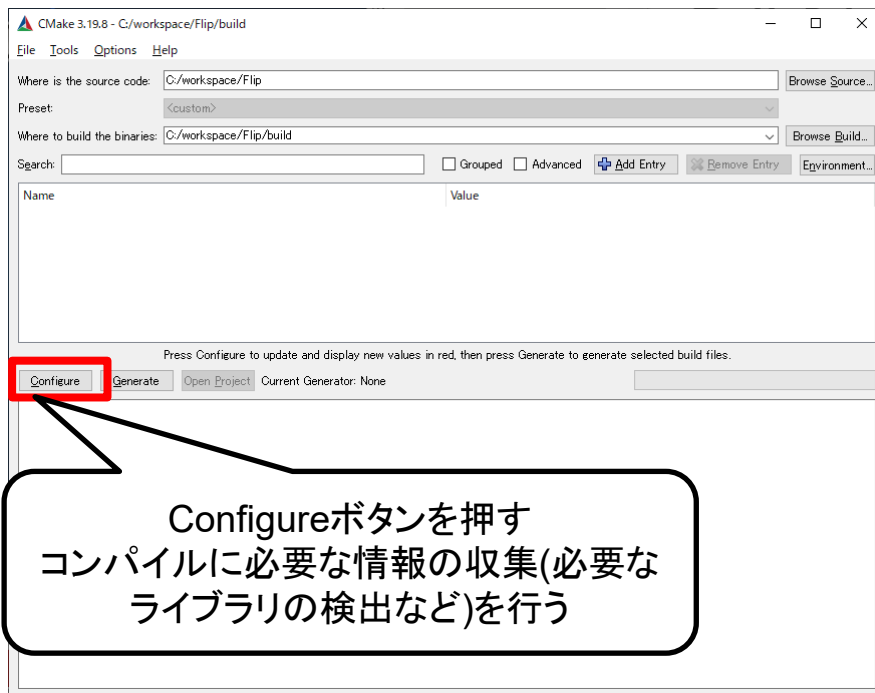
- **CMakeLists.txt**をcmake-guiにドラッグアンドドロップ
 - CMakeLists.txtはRTC Builderで生成したプロジェクトのフォルダ
 - (例: C:¥workspace¥Flip)
 - ※先ほど編集したsrc/CMakeLists.txtではなく、一つ上のディレクトリのファイル



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



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



Specify the generator for this project

Visual Studio 16 2019

Optional platform for generator(if empty, generator uses: x64)

Optional toolset to use (argument to -T)

- ☒ Use default native compilers
- ☐ Specify native compilers
- ☐ Specify toolchain file for cross-compiling
- ☐ Specify options for cross-compiling

Finish

Cancel

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

Specify the generator for this project

Visual Studio 12 2013

Optional toolset to use (-T parameter)

☒ Use default native compilers

☐ Specify native compilers

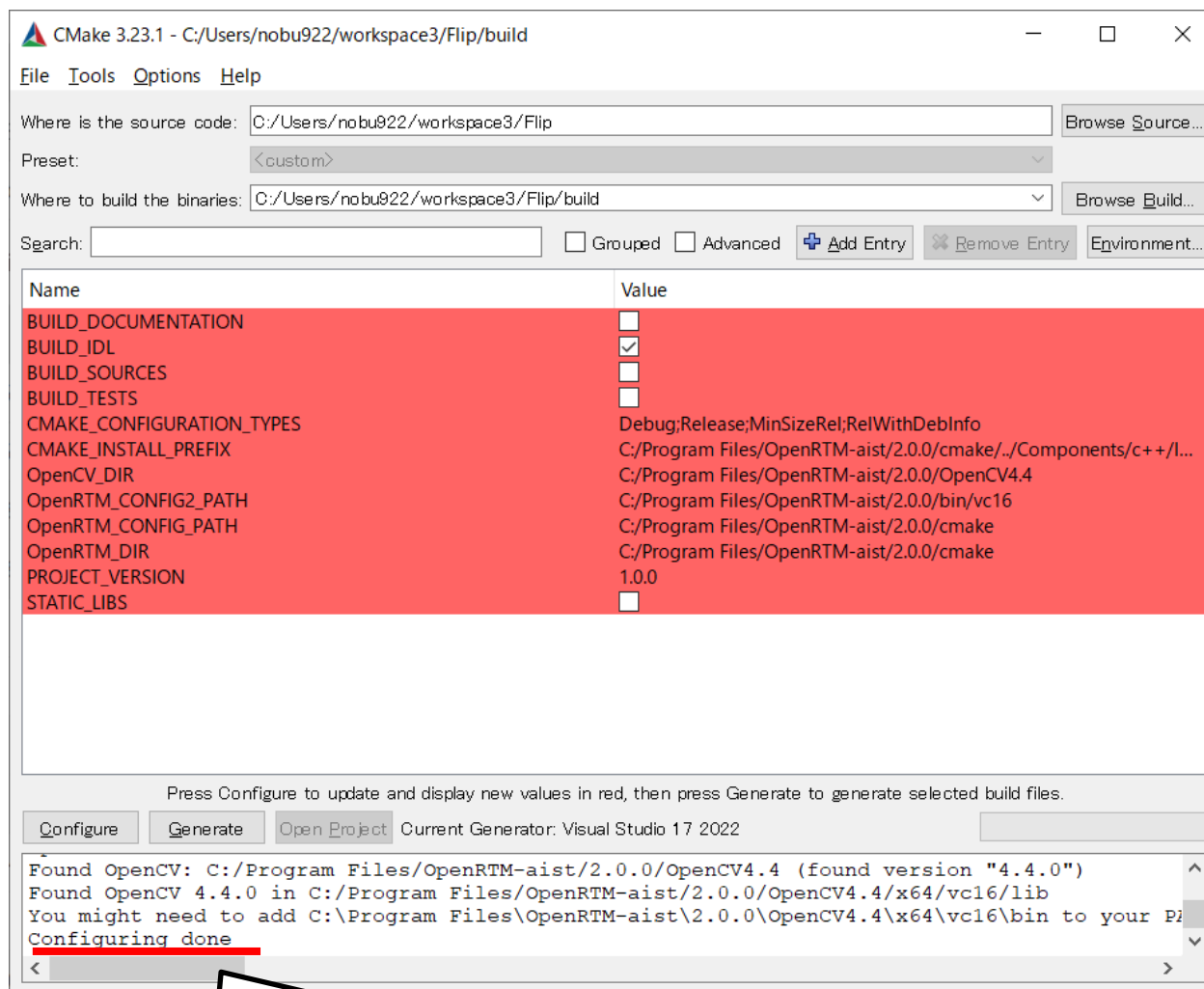
☐ Specify toolchain file for cross-compiling

☐ Specify options for cross-compiling

Finish Cancel

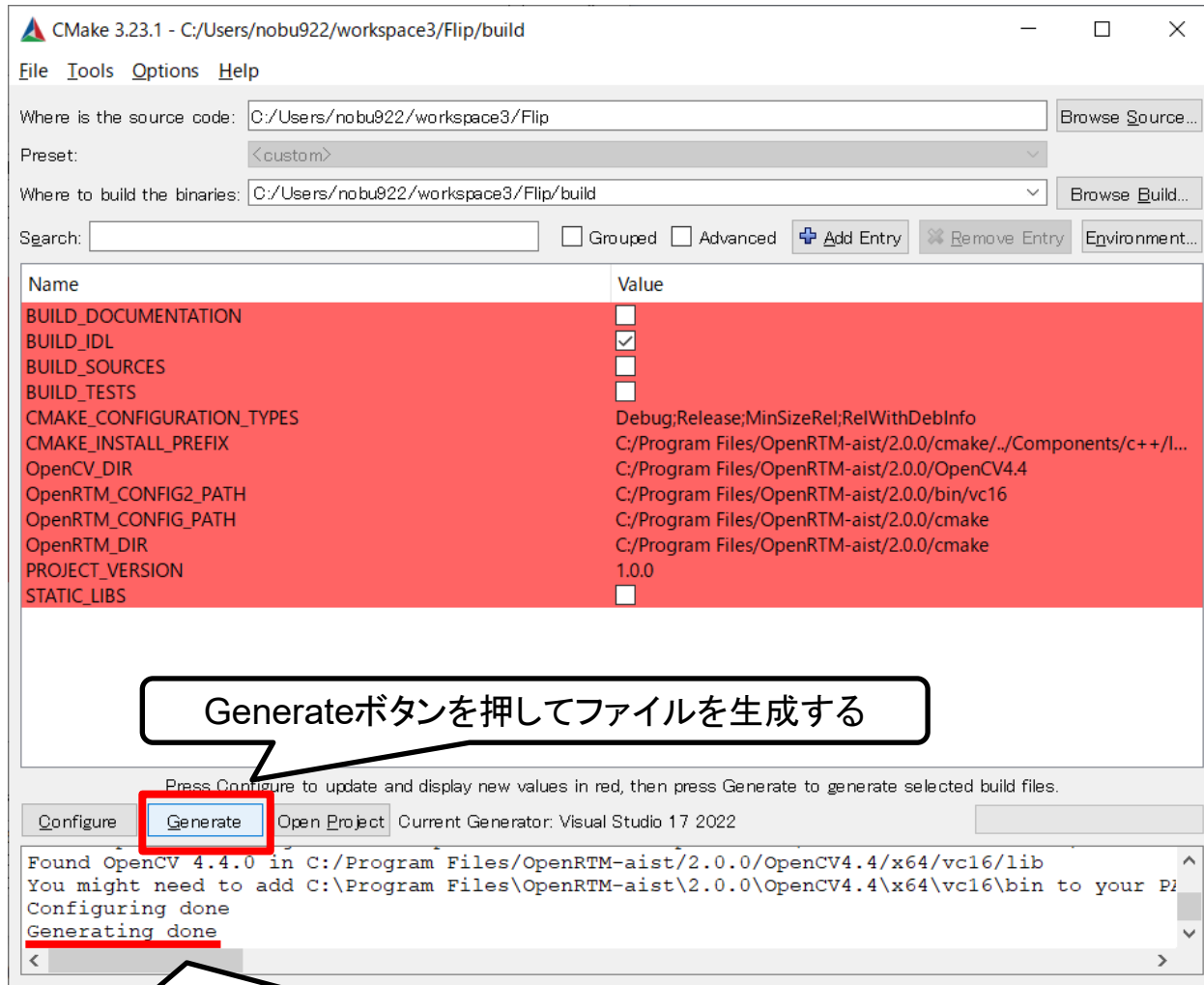
設定後、Finishボタンを押す

ビルドに必要なファイルの生成



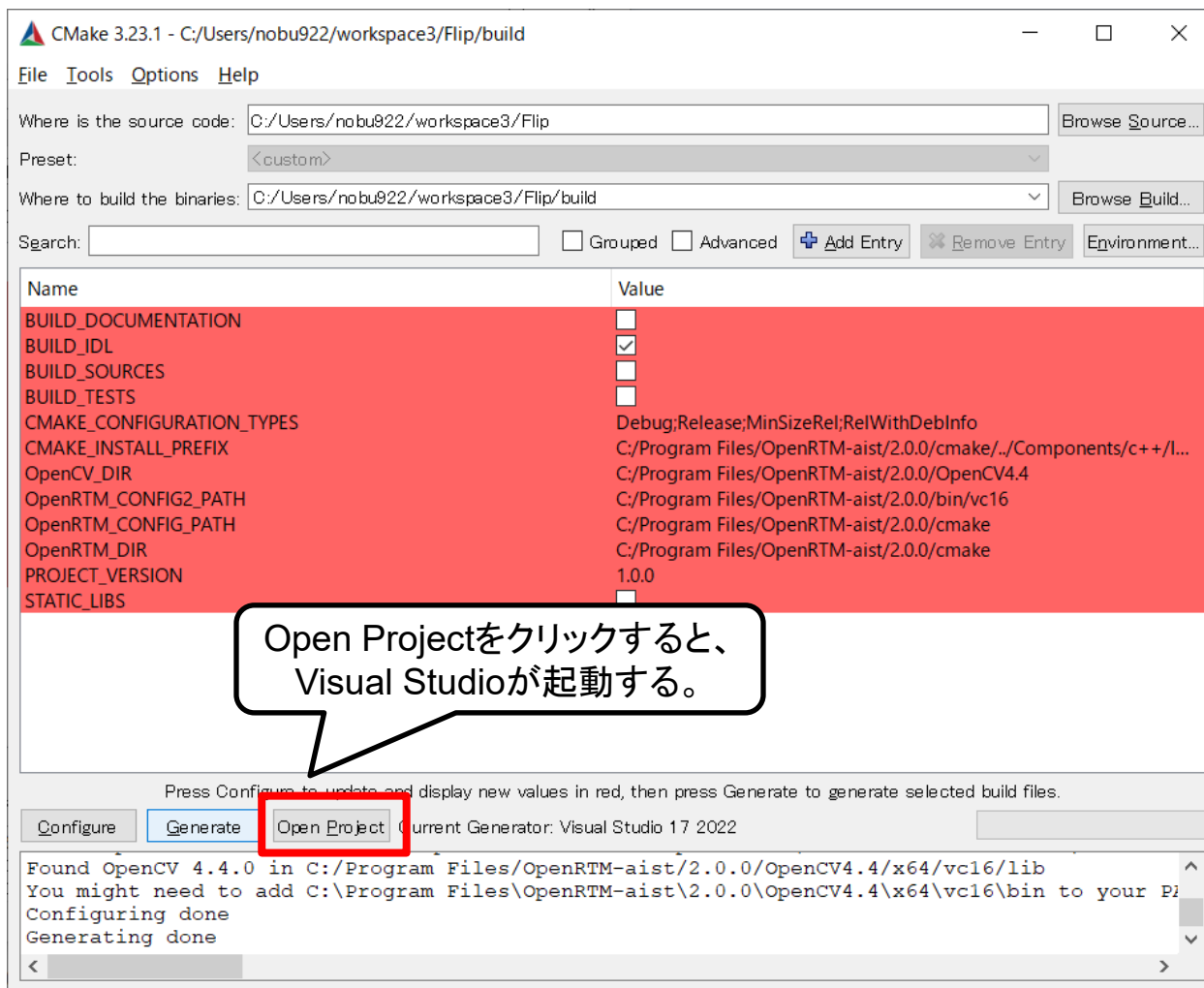
「Configure done」が表示されていれば成功

ビルドに必要なファイルの生成



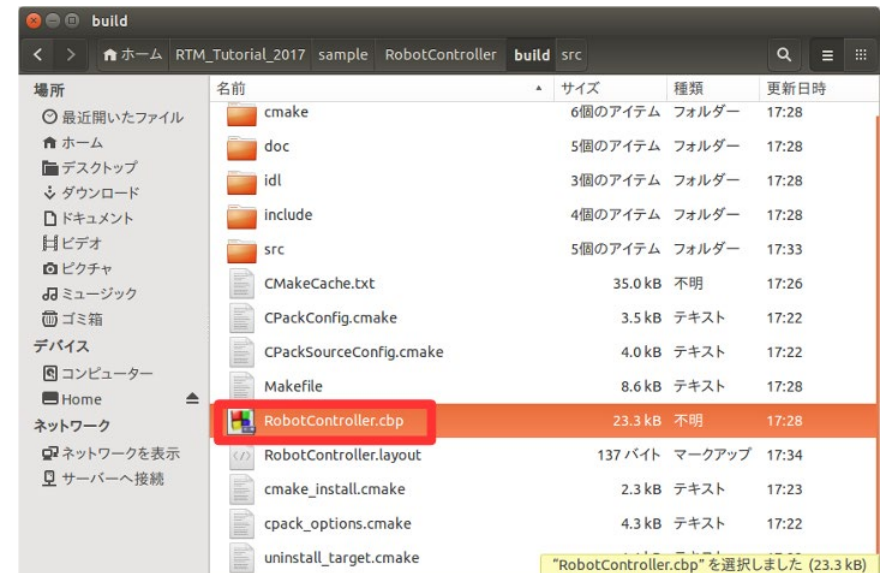
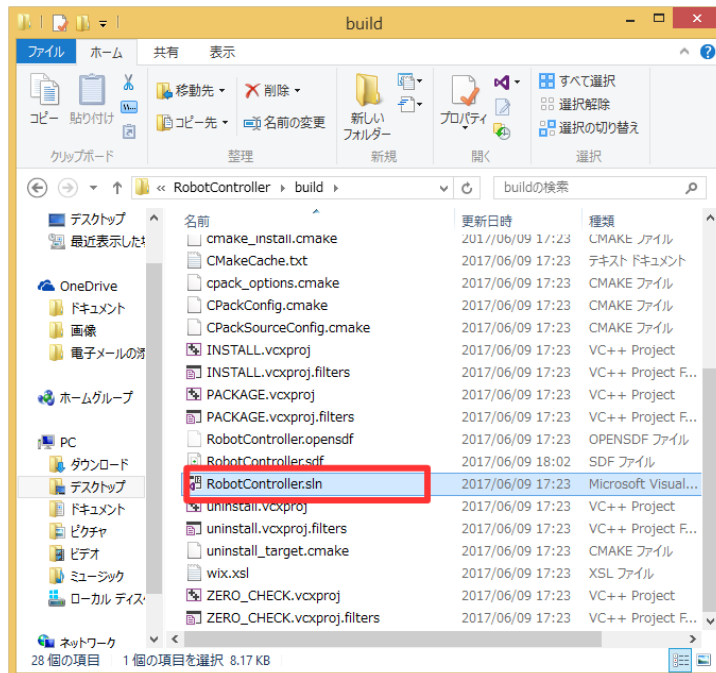
「Generating done」が表示されていれば成功

ソースコードの編集



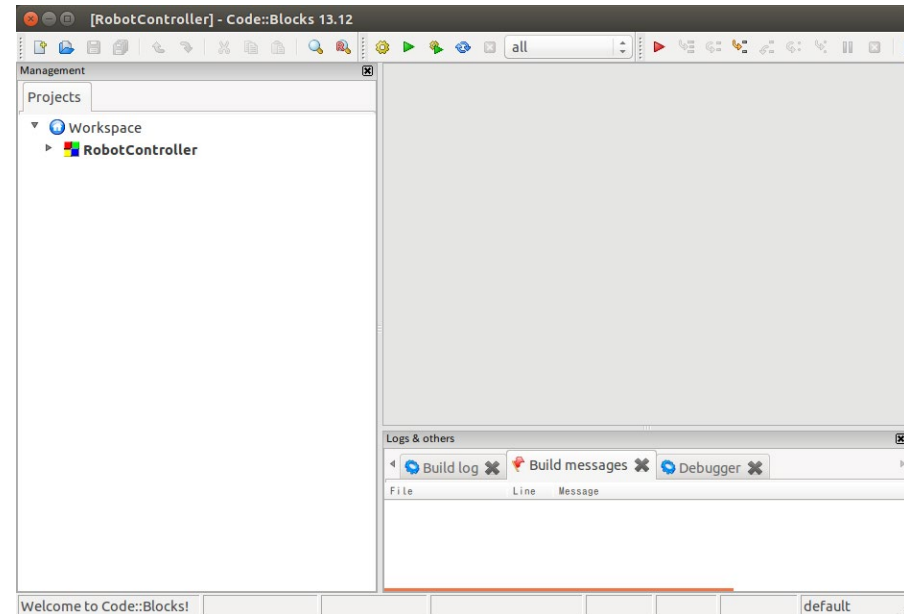
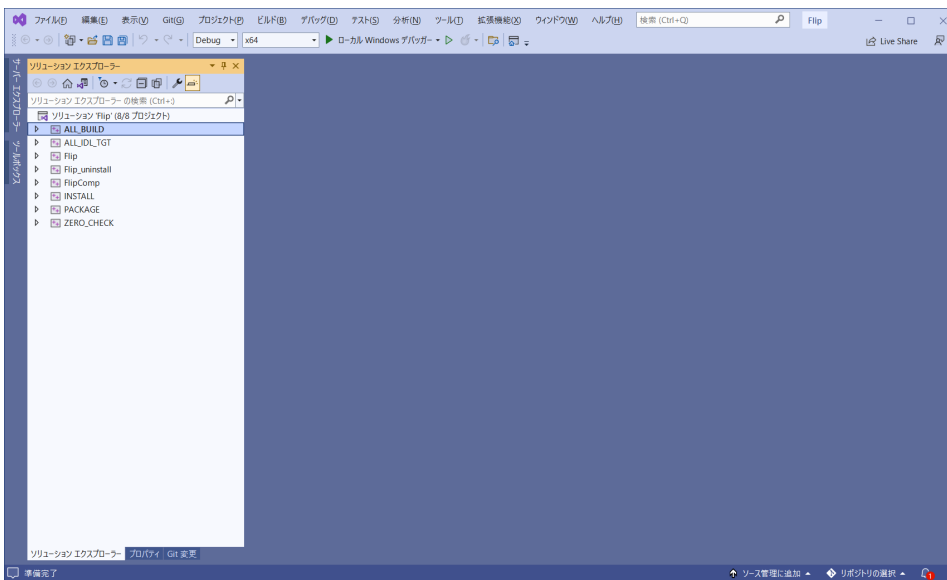
ソースコードの編集

- 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

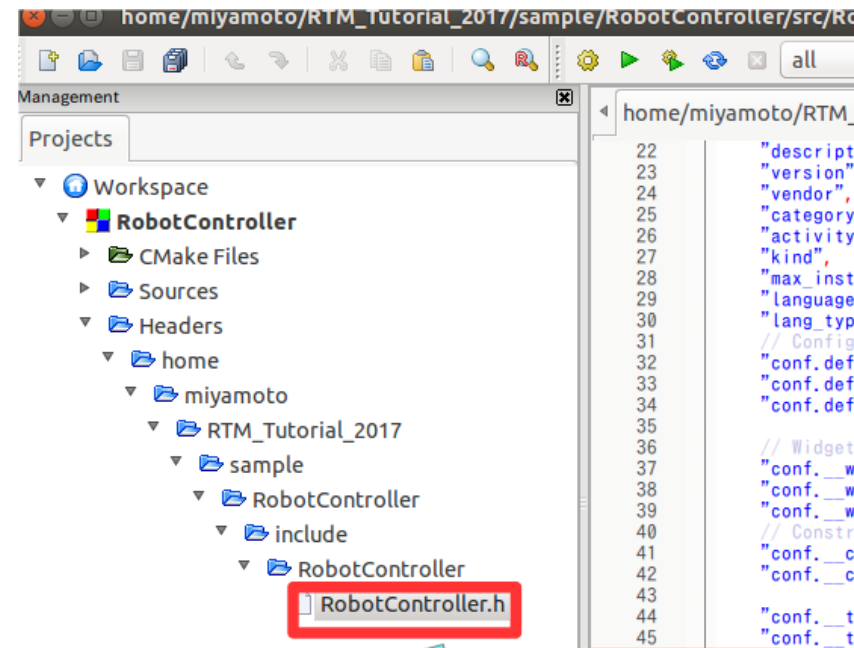
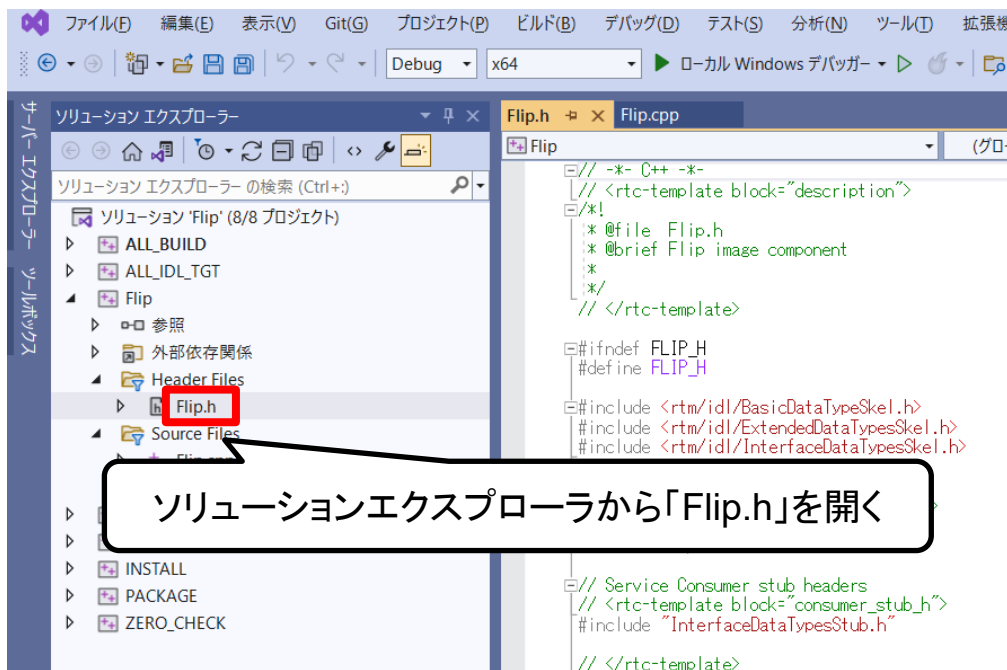


ソースコードの編集

- 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>
```

35行目あたりに追加

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

```
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;
```

271行目あたりに追加

```
//変換後画像を格納する変数
```

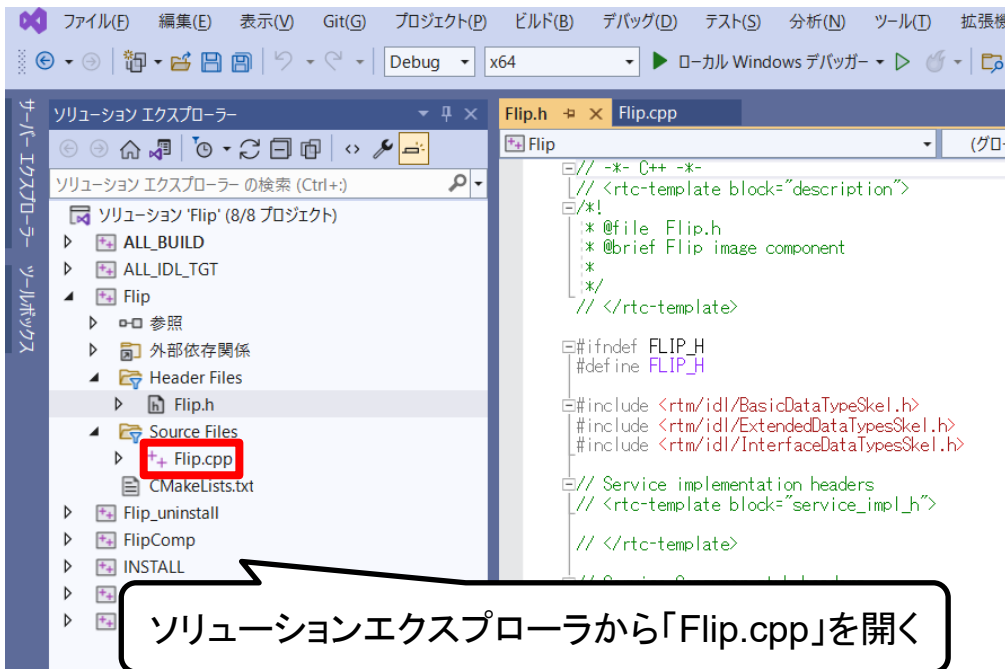
```
cv::Mat m_flipImageBuff;
```

274行目あたりに追加

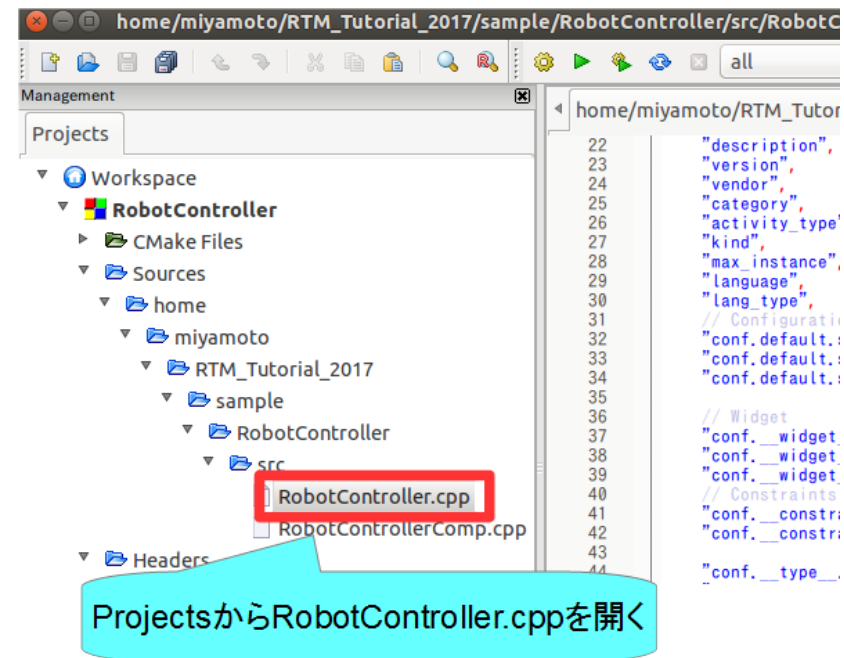
ソースコードの編集

- 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

```
if (m_originalImageIn.isNew()) {  
    // InPortデータの読み込み  
    m_originalImageIn.read(&m_originalImage);  
  
    // InPortとOutPortのサイズを一致させる  
    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());  
}
```

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

補足: ameraImage型はpixelsに画像データ、width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);
```

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

```
// 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。
```

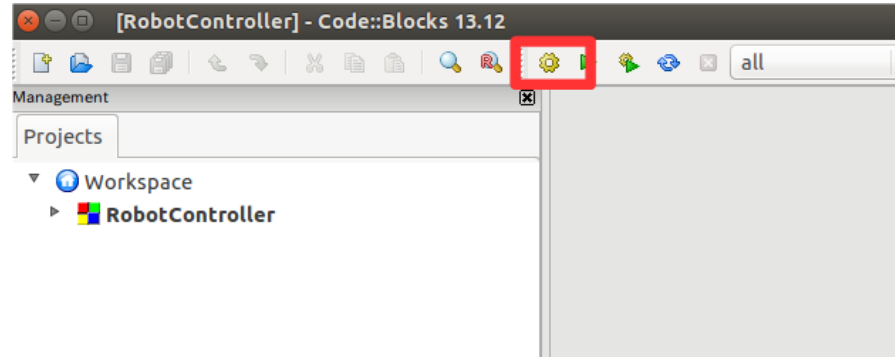
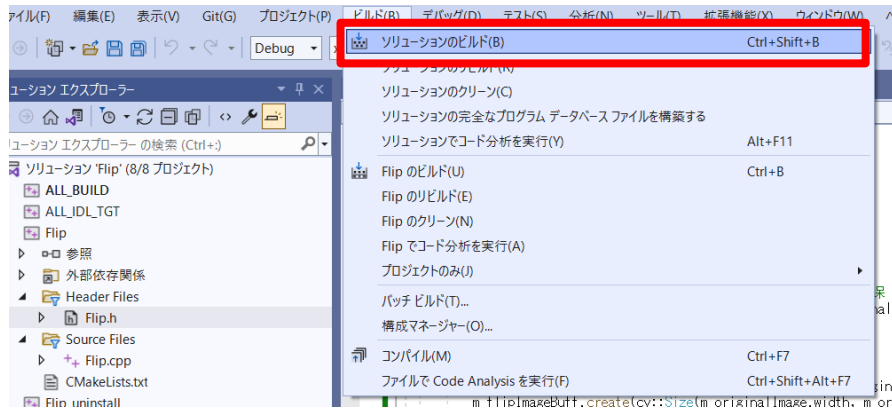
```
m_flippedImageOut.write();
```

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

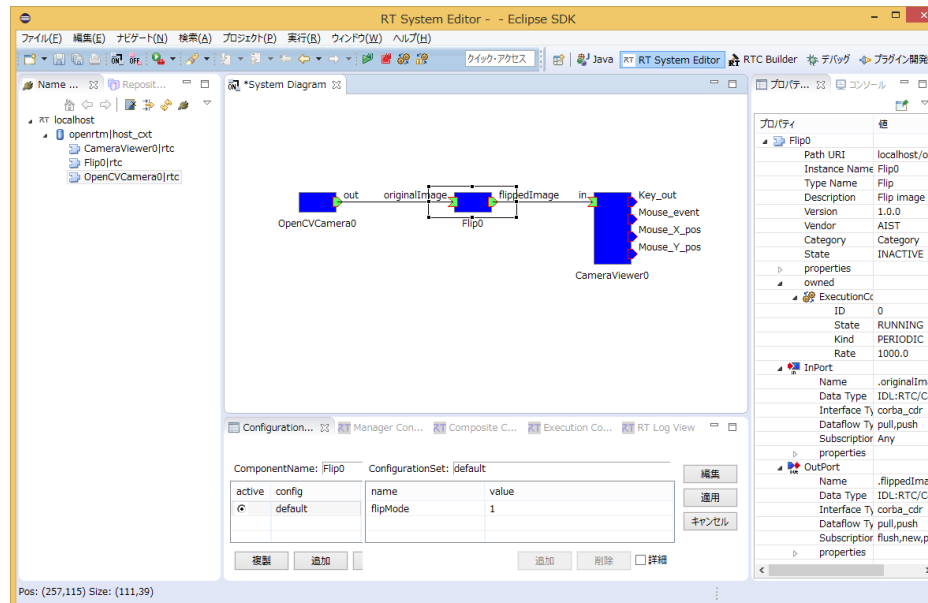


- 成功した場合、実行ファイルが生成される
 - Windows
 - build\src**フォルダの**Release**(もしくは**Debug**)フォルダ内にFlipComp.exeが生成される
 - Ubuntu
 - build/src**フォルダにFlipCompが生成される

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

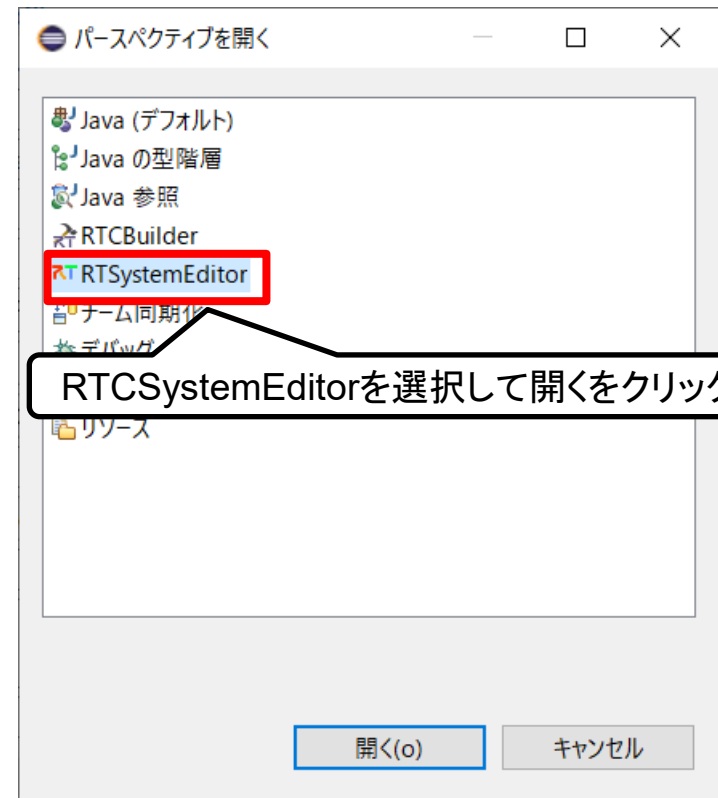
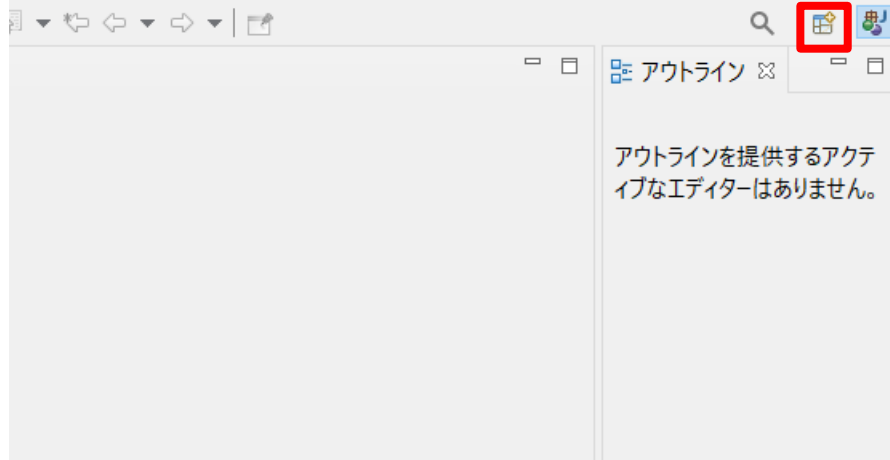


RT System Editorの起動

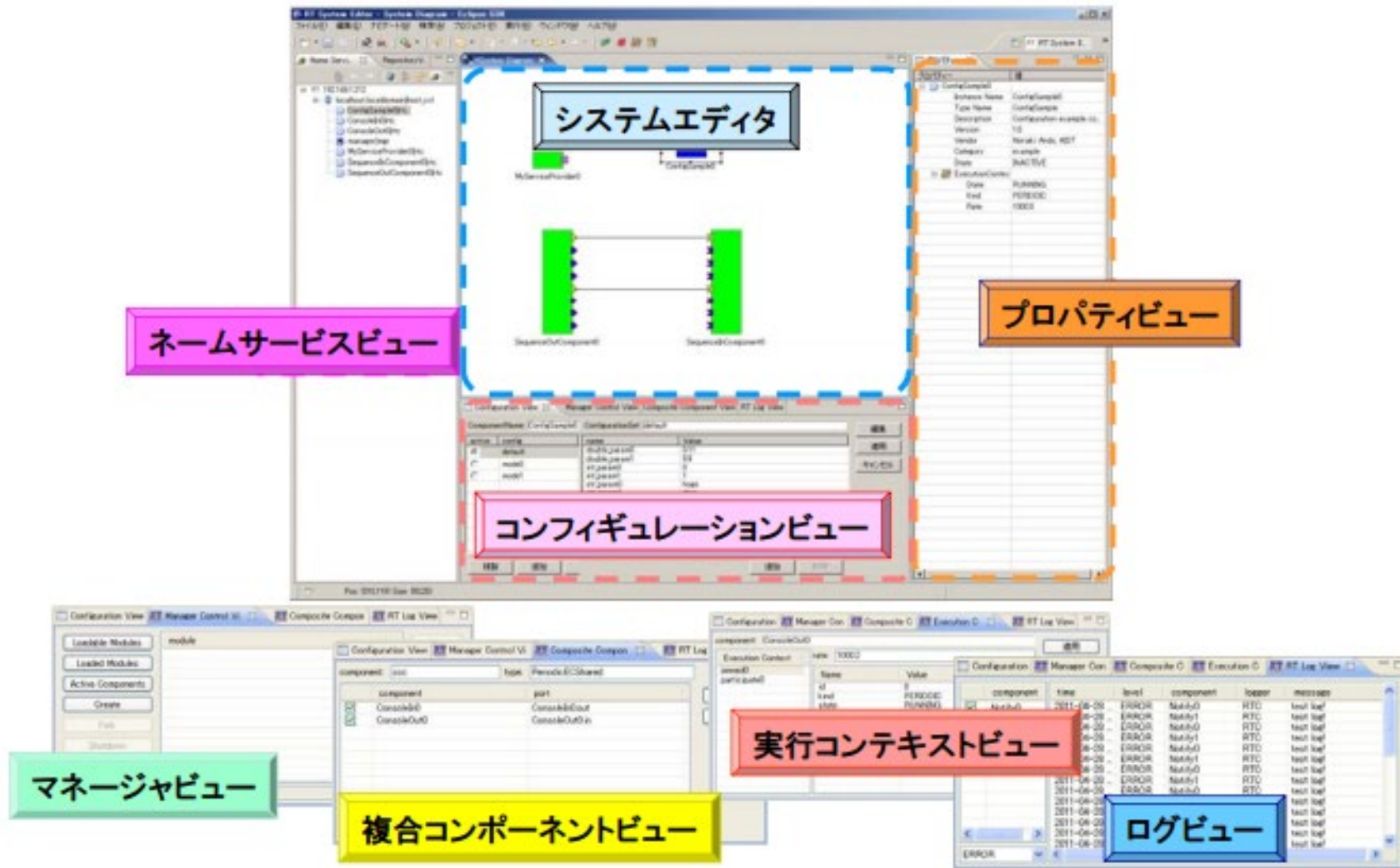
右上の「パースペクティブを開く」ボタンをクリック



A) プロジェクト(P) 実行(R) ウィンドウ(W) ヘルプ(H)

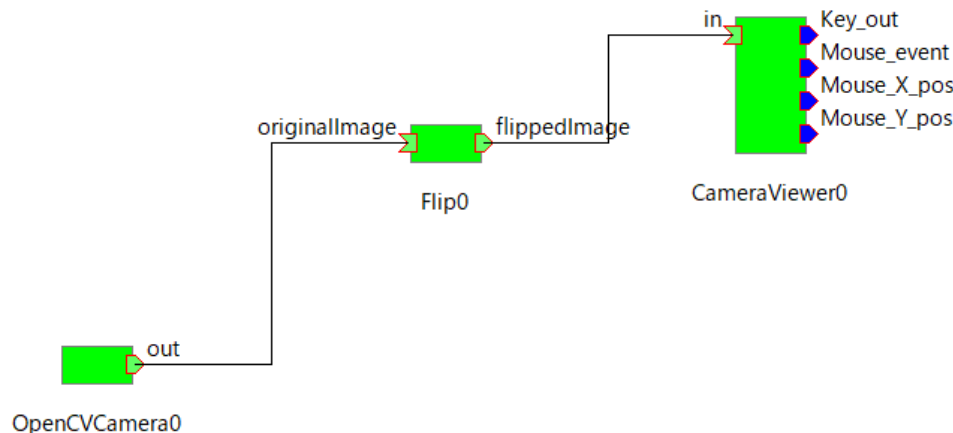


RT System Editorの画面構成



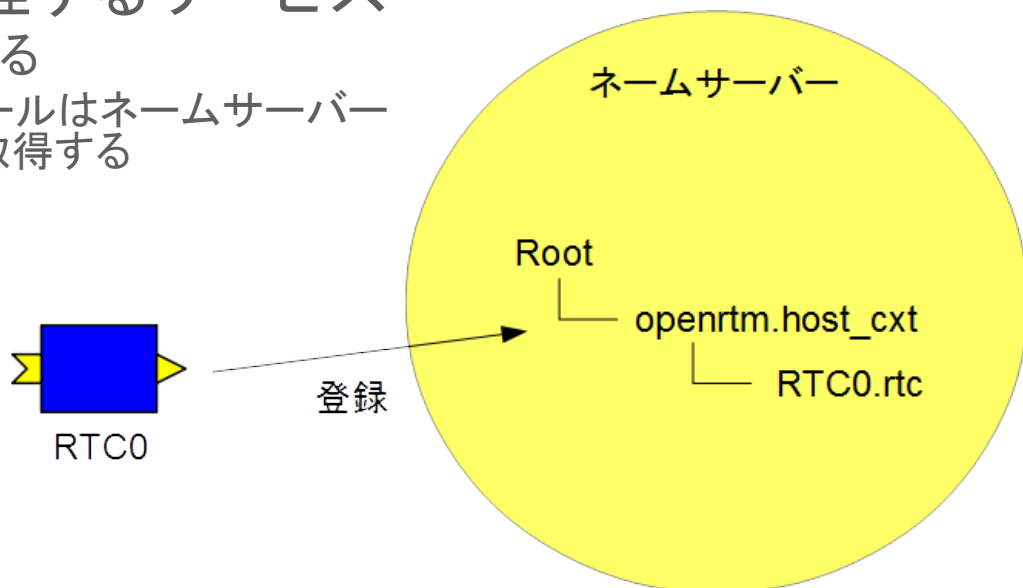
Flipコンポーネントの動作確認

- 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-2.0/components/c++/opencv-rtcs/CameraViewerComp
 - » \$ /usr/local/share/openrtm-2.0/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

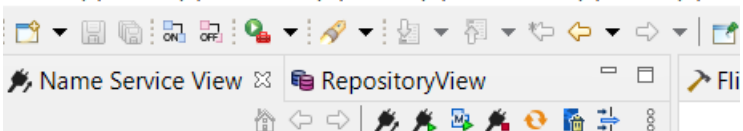
- オブジェクトを名前で管理するサービス
 - RTCを一意の名前で登録する
 - RT System Editor等のツールはネームサーバーから名前でRTCの参照を取得する



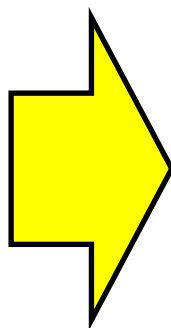
起動する手順

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

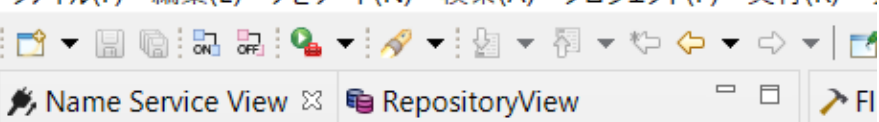


ネームサーバー起動ボタンを押す



workspace3 - - Eclipse SDK

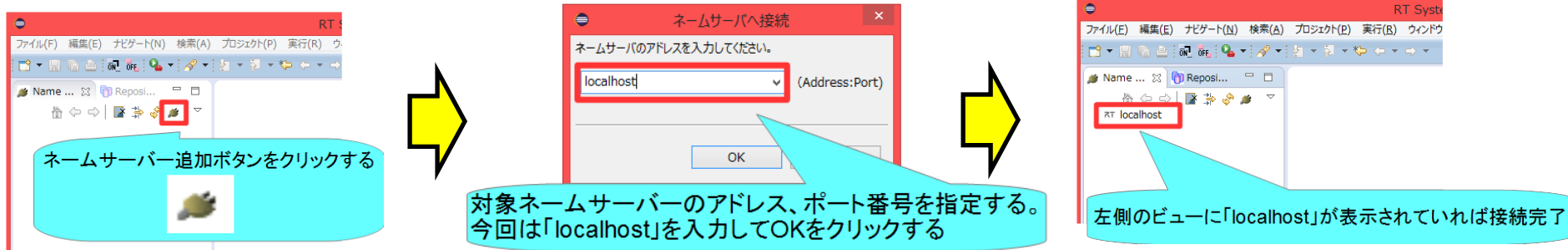
ファイル(F) 編集(E) ナビゲート(N) 検索(A) プロジェクト(P) 実行(R) フ



左側のビューに「localhost」が追加
されていたら接続成功

ネームサーバーの起動

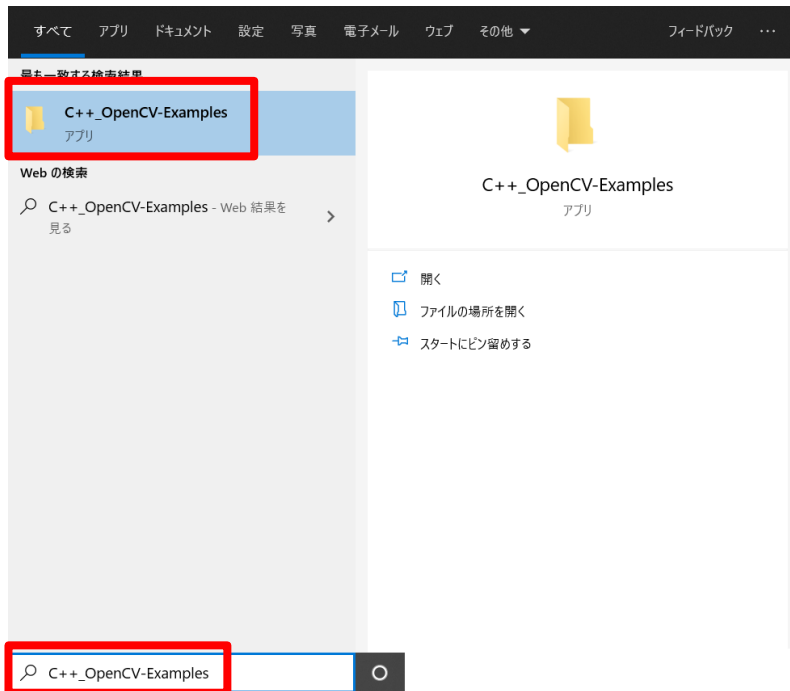
- OpenRTM-aist 1.1.2以前の手順
 - Windows 7
 - 「スタート」→「すべてのプログラム」→「OpenRTM-aist 1.1.2」→「Tools」→「Start Naming Service」
 - Windows 8.1
 - 「スタート」→「アプリビュー(右下矢印)」→「OpenRTM-aist 1.1.2」→「Start Naming Service」
 - Windows 10
 - 左下の「ここに入力して検索」にStart Naming Serviceと入力して起動
 - Ubuntu
 - \$ rtm-naming



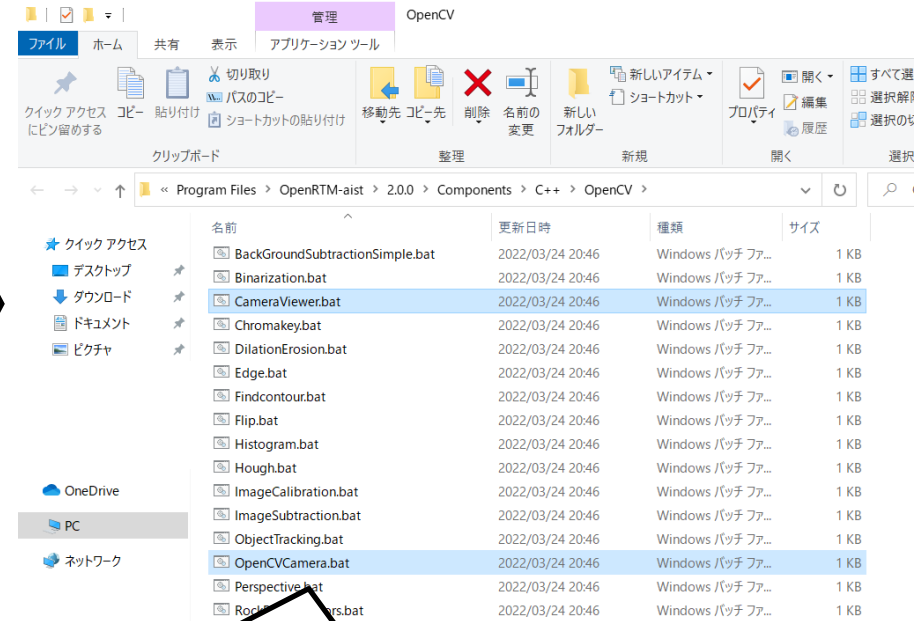
Flipコンポーネントの動作確認

- 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-2.0/components/c++/opencv-rtcs/CameraViewerComp
 - » \$ /usr/local/share/openrtm-2.0/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)

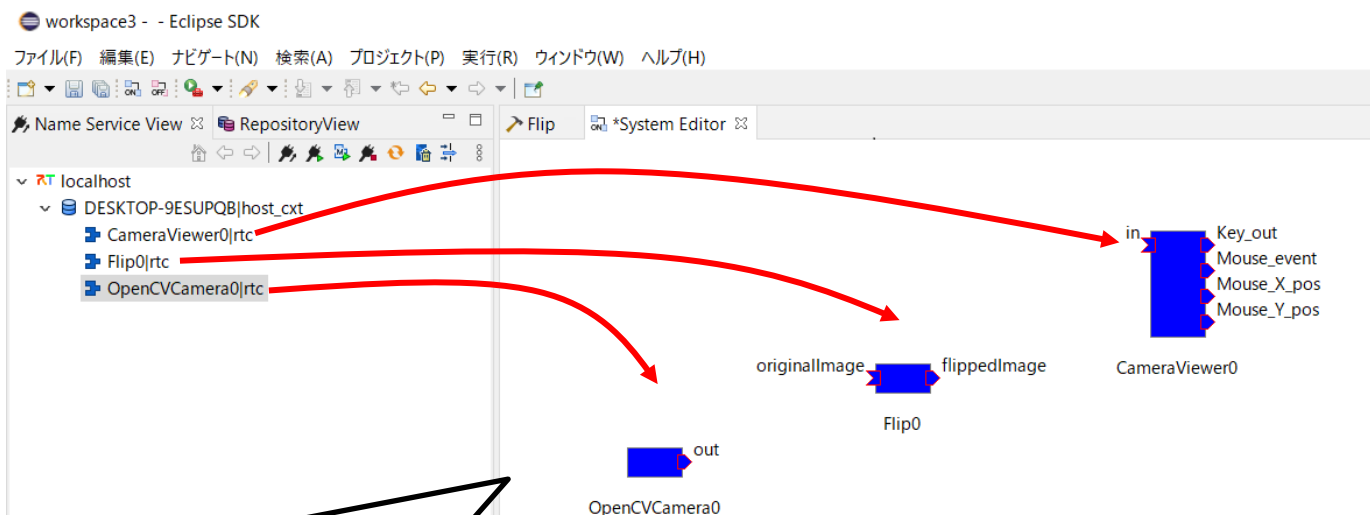
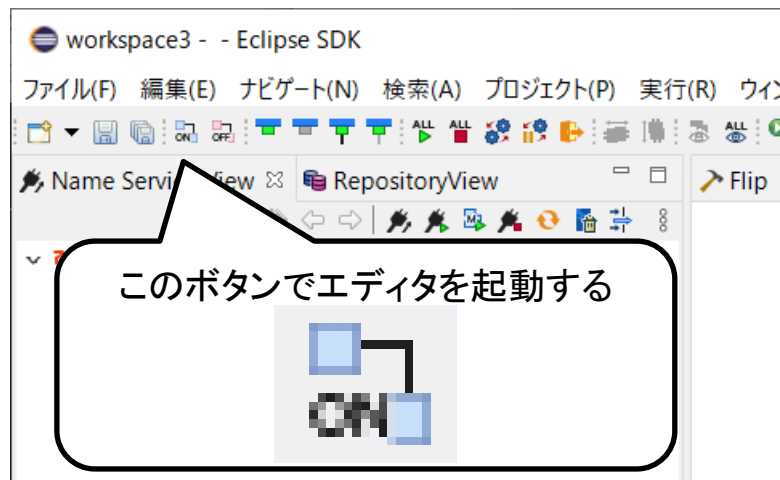


画面左下の「ここに入力して検索」に
「C++_OpenCV-Examples」と入力して検索する



「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



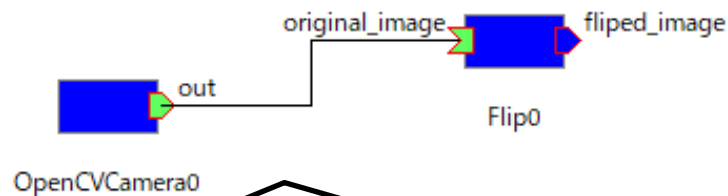
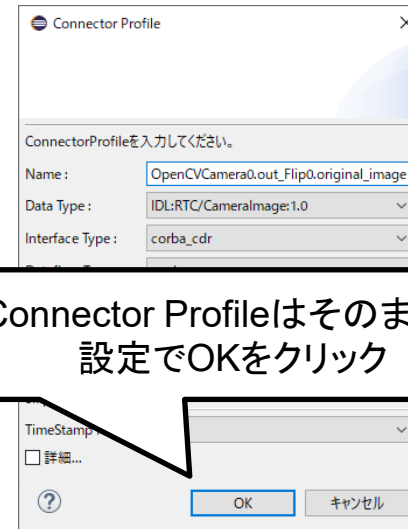
左側のネームサービスビューからOpenCVCamera0.rtc、CameraViewer0.rtc、Flip0.rtcをドラッグアンドドロップ

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

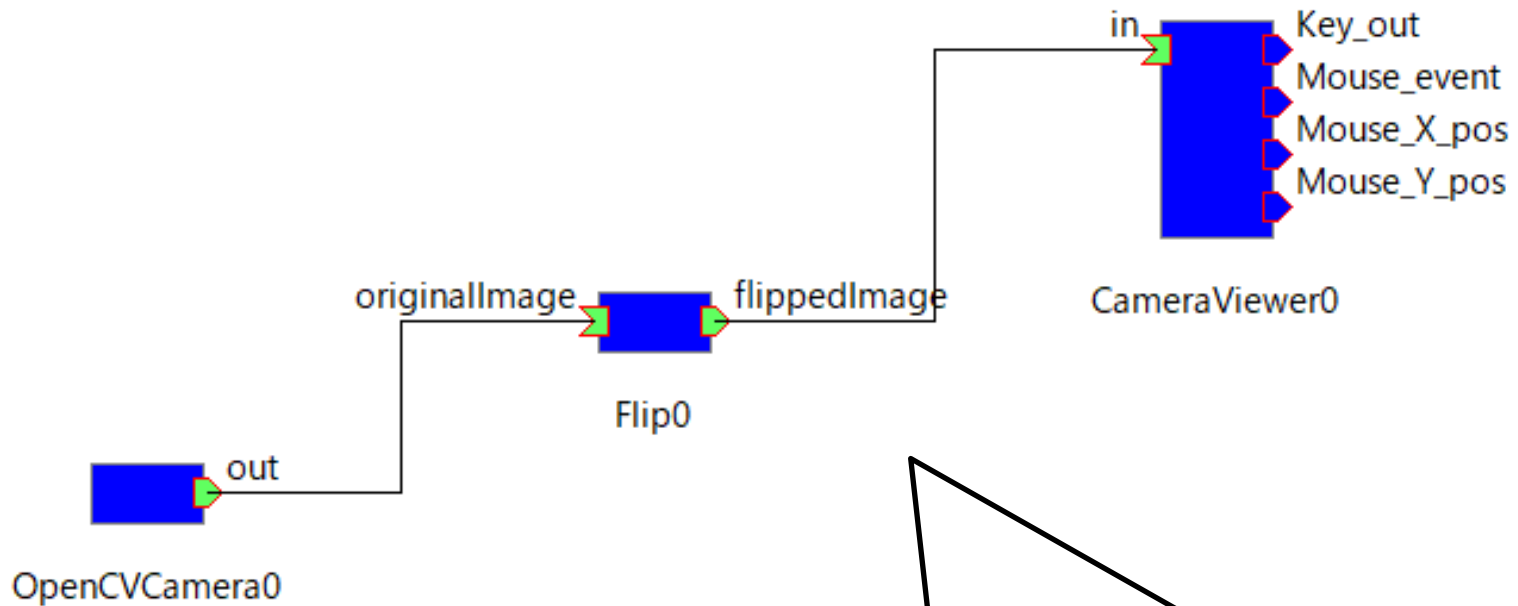


Connector Profileはそのままでの設定でOKをクリック



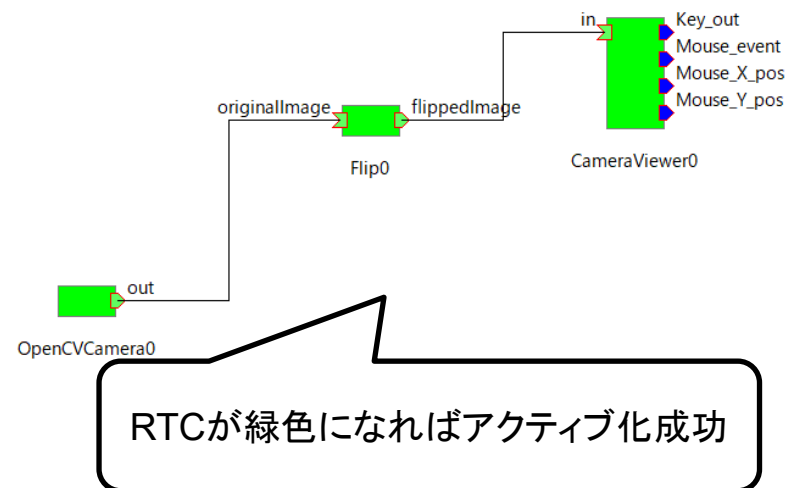
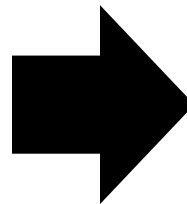
1. ポートの間に線が表示される
2. InPort、OutPortが緑色で表示される

データポートの接続



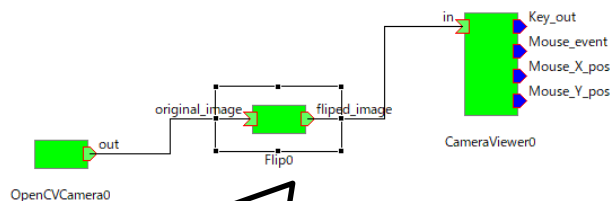
Flip0の「flippedImage」とCameraViewer0と「in」を接続する

アクティブ化

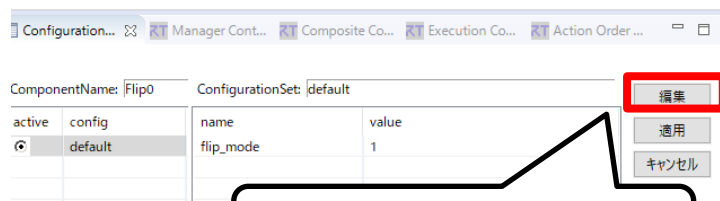


コンフィギュレーションパラメータの操作

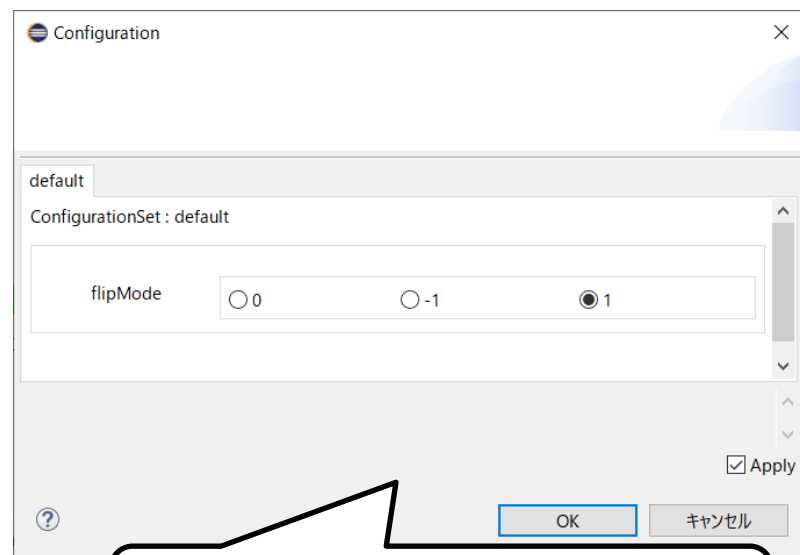
- コンフィギュレーションパラメータをRT System Editorから操作する



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ー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(Running)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

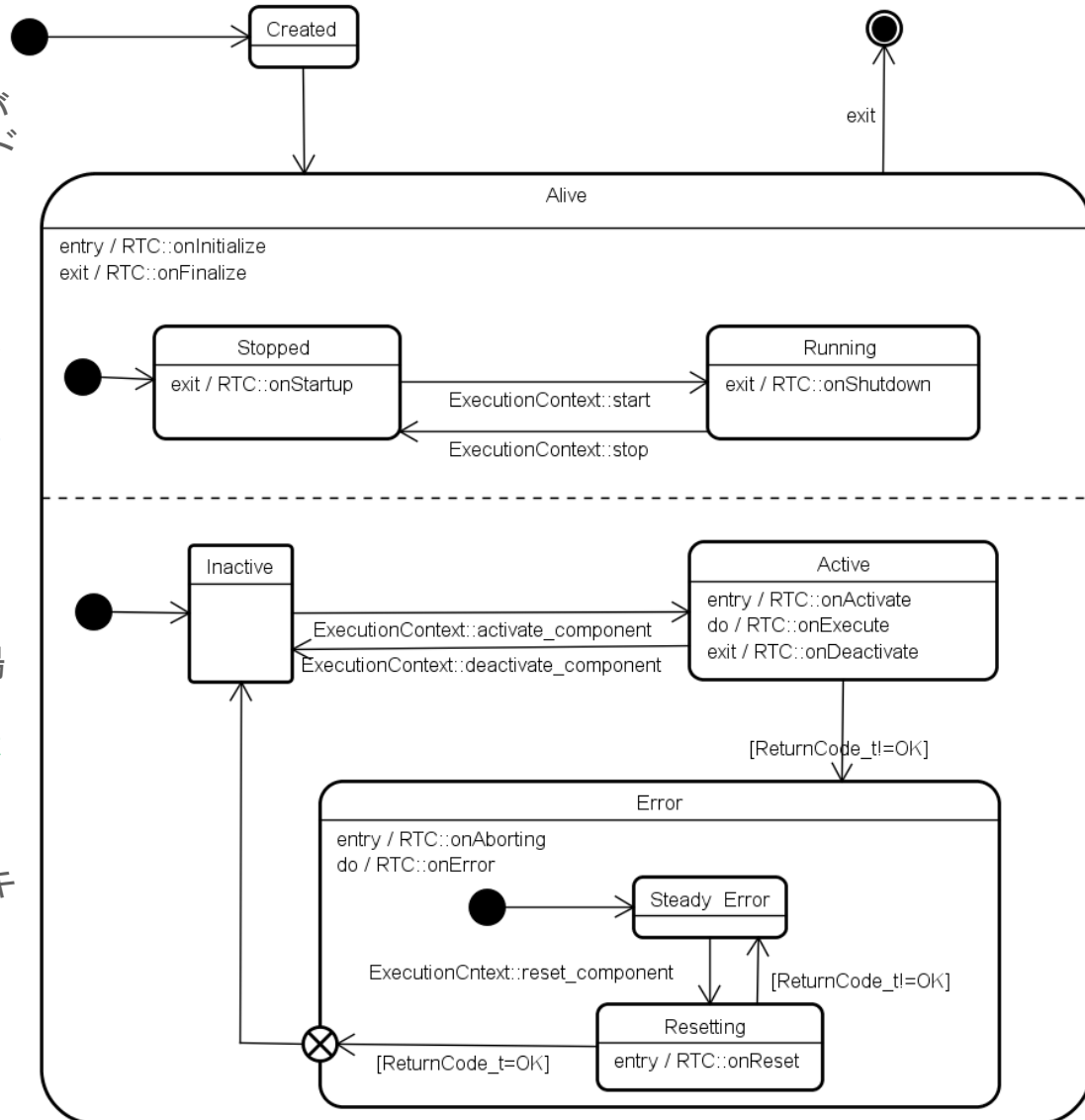
- Active

- 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

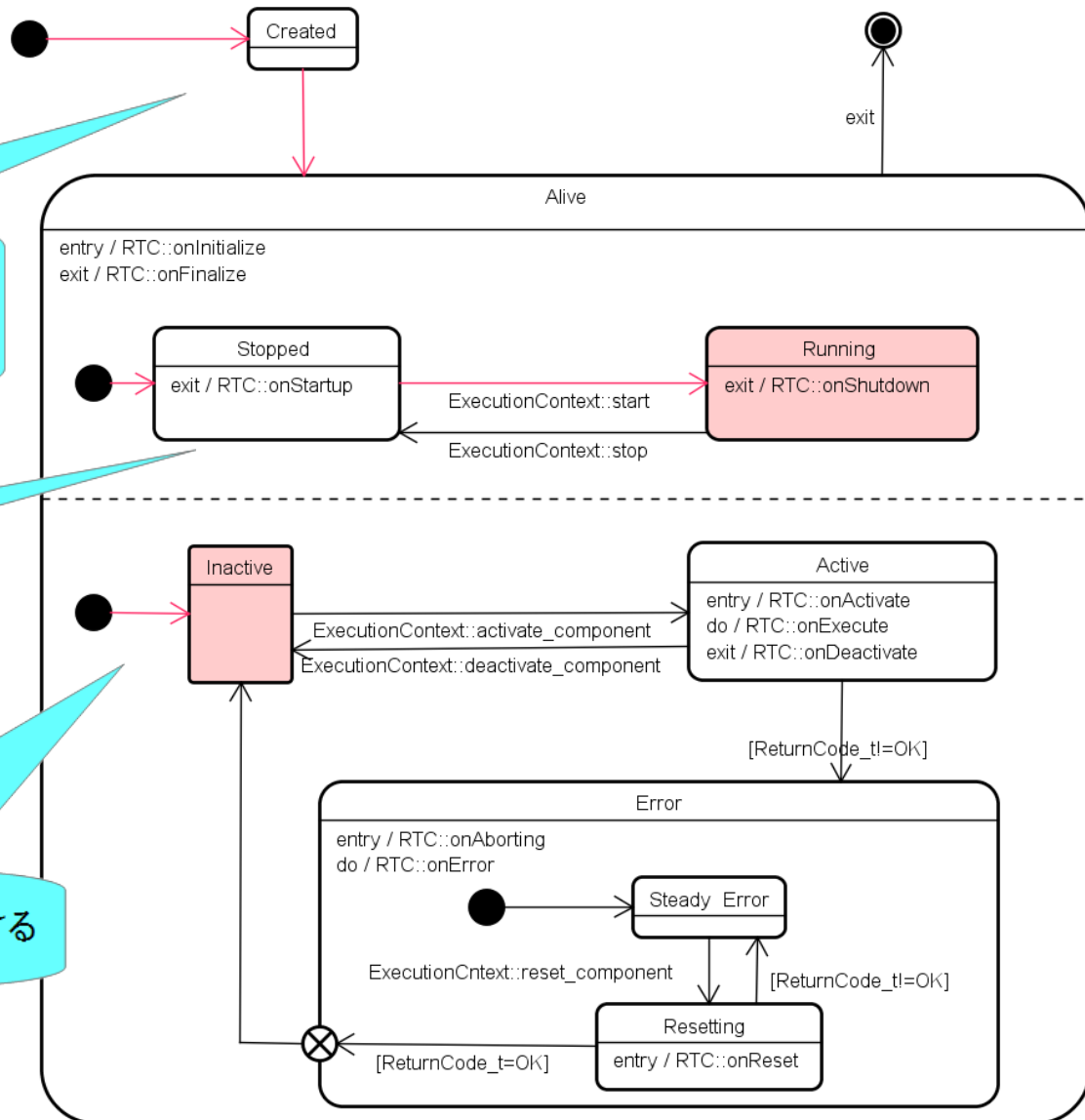


RTコンポーネントの状態遷移(生成直後)

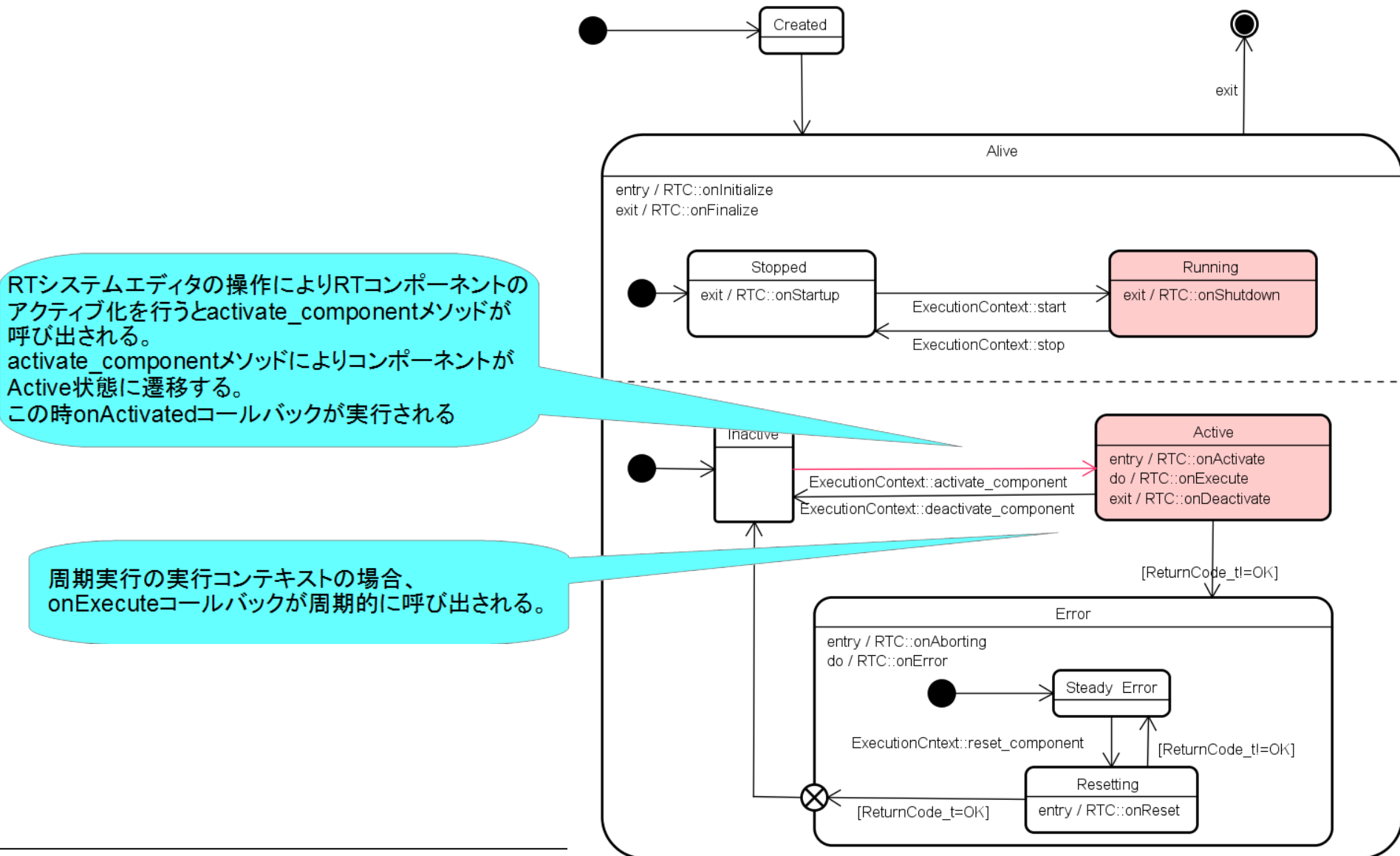
最初にCreated状態に遷移して
実行コンテキストの生成等を行う
この時にonInitializeコールバックを実行する

startメソッドにより実行コンテキストが
Running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

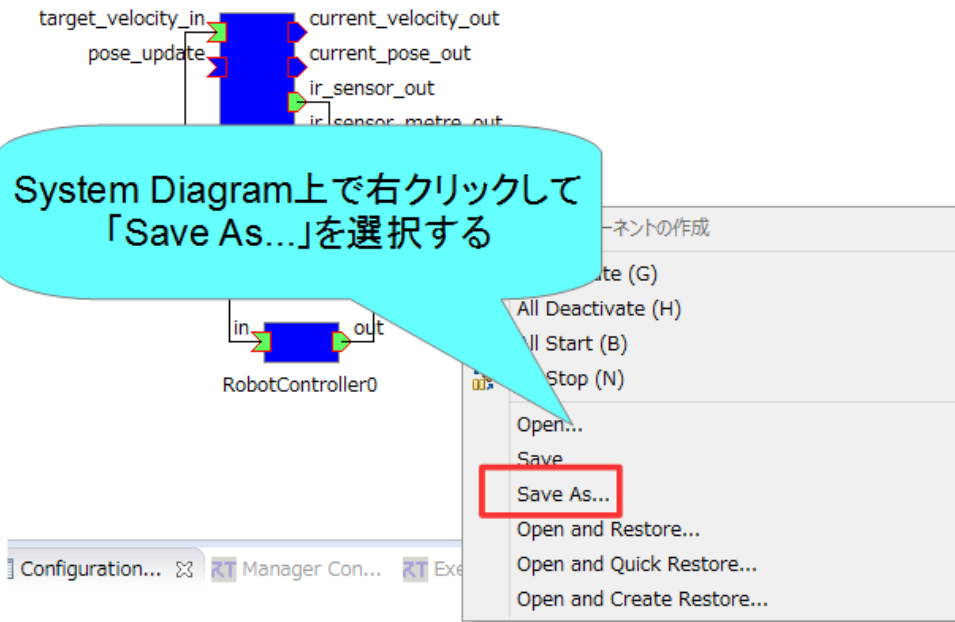
Created状態の次にInactive状態に遷移する



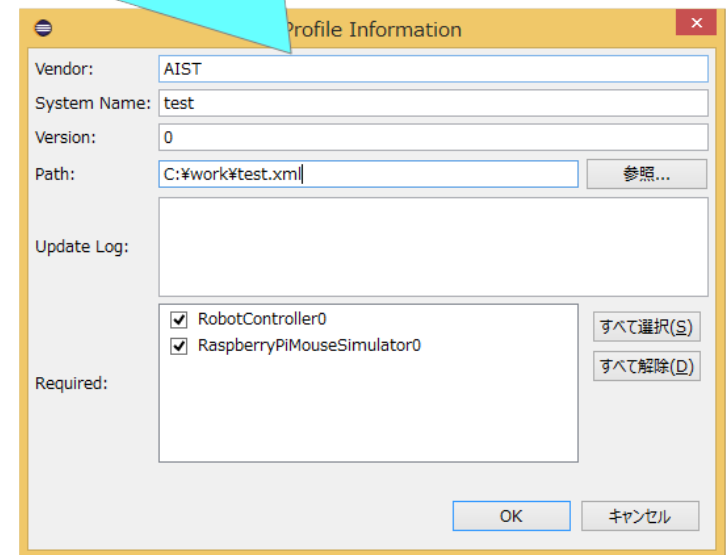
RTコンポーネントの状態遷移(アクティブ化)



システムの保存



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

Required:

すべて選択(S) すべて解除(D)

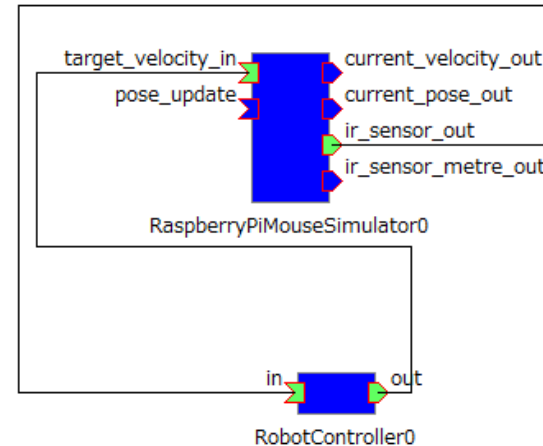
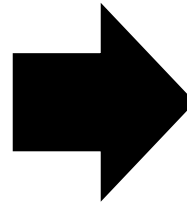
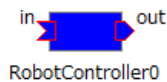
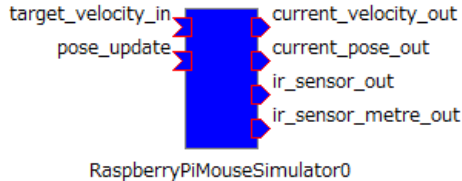
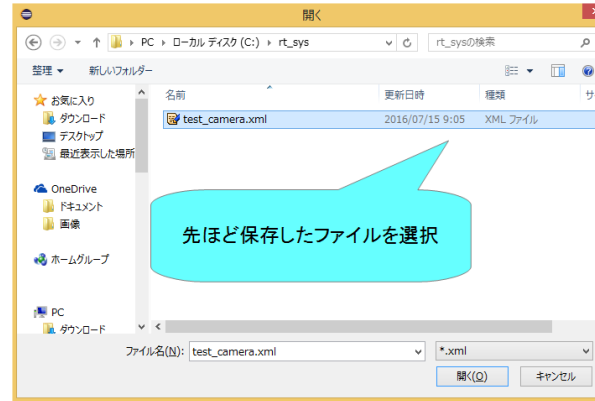
OK キャンセル

システムの復元

System Diagram上で右クリックして
「Open and Restore...」を選択する

複合コンポーネントの作成

All Activate (G)
All Deactivate (H)
All Start (B)
All Stop (N)
Open...
Save As...
Open and Restore...
Open and Quick Restore...
Open and Create Restore...



- 以下の内容を復元

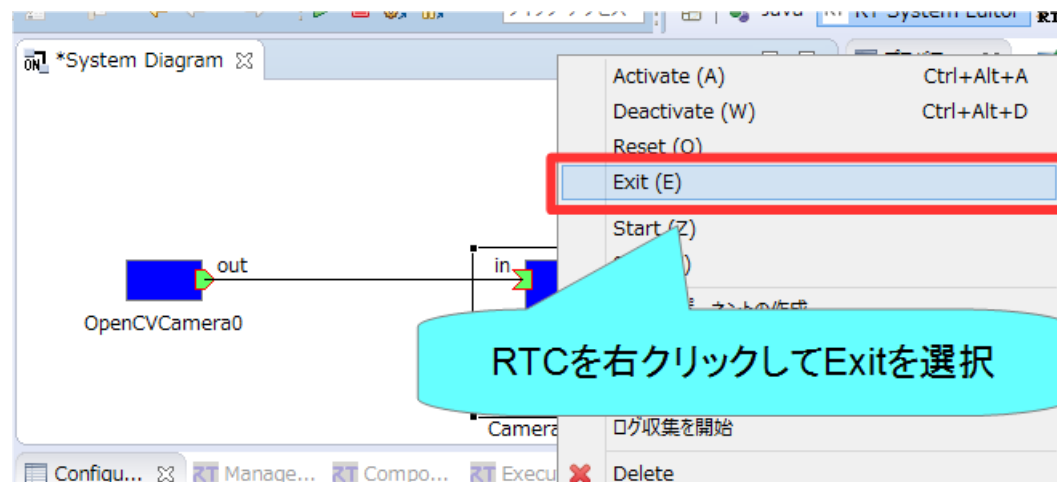
- ポート間の接続
- コンフィギュレーション
- 「Open and Create Restore」を選択した場合はマネージャからコンポーネント起動

非アクティブ化、終了

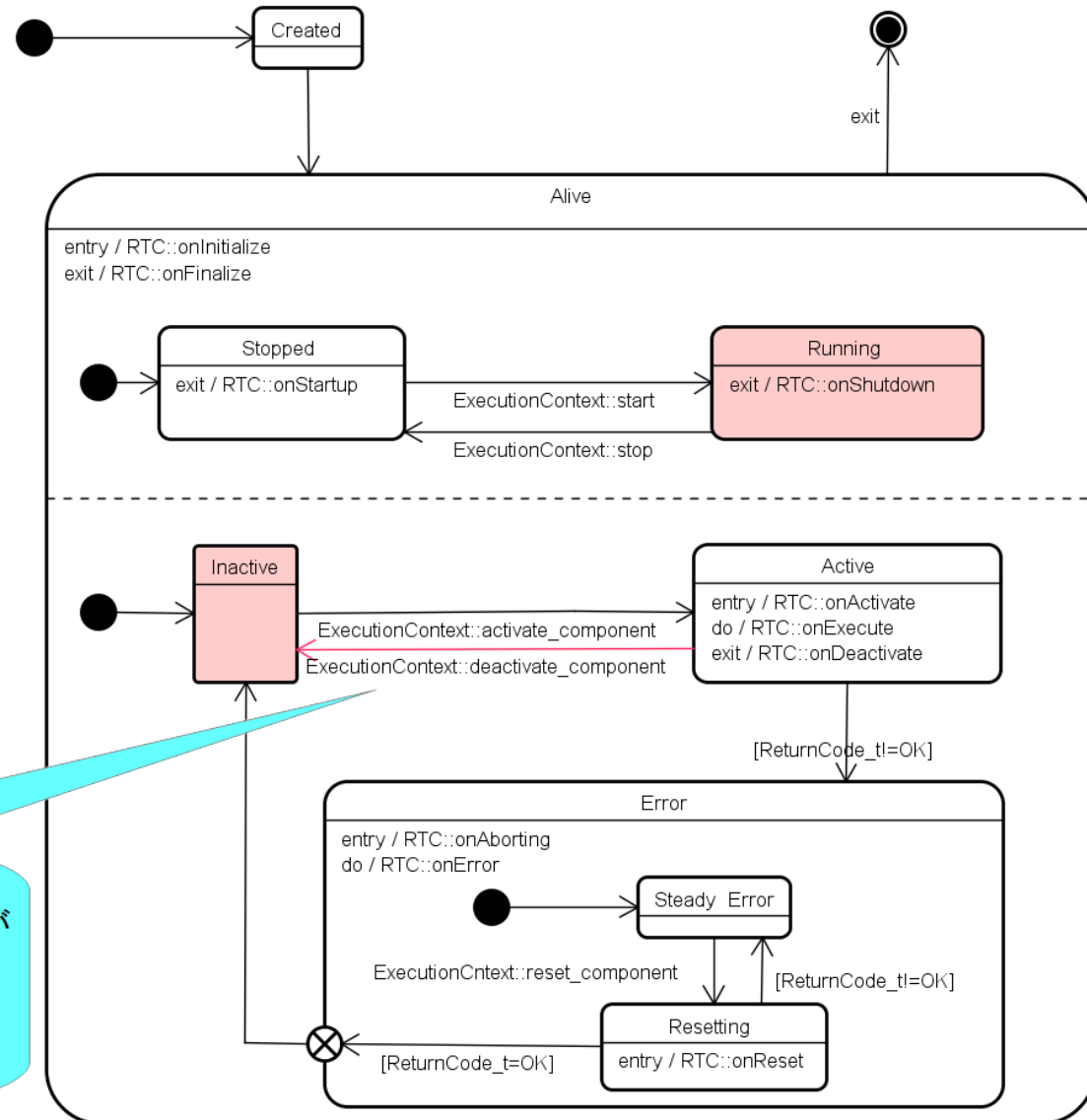
- 非アクティブ化



- 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