

2021年11月09日

高度ポリテクセンター・能力開発セミナー (E0771)

「RTミドルウェアによるロボットプログラミング技術」

3. RTCプログラミング演習



はじめに

- 本日の講習会のWebページ
– <https://bit.ly/3EI3peN>
- こちらに、必要な情報がまとまっていますので、ブックマークしてください。


OpenRTM-aist
MIDDLEWARE The power to connect

[ホーム](#)
[ダウンロード](#)
[ドキュメント](#)
[コミュニティ](#)
[研究開発](#)
[プロジェクト](#)
[ハードウェア](#)

ホーム » 高度ポリテクセンター「RTミドルウェアによるロボットプログラミング技術」(2020年11月10日～11日)

高度ポリテクセンター「RTミドルウェアによるロボットプログラミング技術」 (2020年11月10日～11日)

Facebookに登録して、友達や「いいね！」を見てください。

Table of contents

- 開催概要
- 日時・場所
- プログラム
 - RaspberryPiマウス
 - インストールするソフトウェア
- 資料
 - 1. コース概要
 - 2. プログラミングの基礎
 - 3. RTMによるプログラミング
 - 4. ロボットの運動学と制御の基礎
 - 5. 総合演習



開催概要

2020年11月10日(火)～11月11日(水)の日程で、高度ポリテクセンターの能力開発セミナーの一環として、「RTミドルウェアによるロボットプログラミング技術」講習会を開催いたします。

・コース案内: <https://www.apc.jeet.or.jp/zaishoku/2020/E0771.html>

日時・場所

・日時: 2019年11月13日(火), 14日(水), 10時00分～16時45分
 ・場所: 高度ポリテクセンター
 ・交通アクセス
 ・主催: 高度ポリテクセンター
 ・参加費: 20,000円

プログラム

11月10日(火)	
10:00 - 10:50	1. コース概要 (1) ロボットシステムプログラミングの現状 (2) ロボットOS・ミドルウェア (3) RTミドルウェア(RTM)を用いたロボット開発 資料: 201110-01.pdf

概要

1. RTCBuilderによるひな形コードの生成
2. プログラムの作成とコンパイル
3. システム構築とテスト

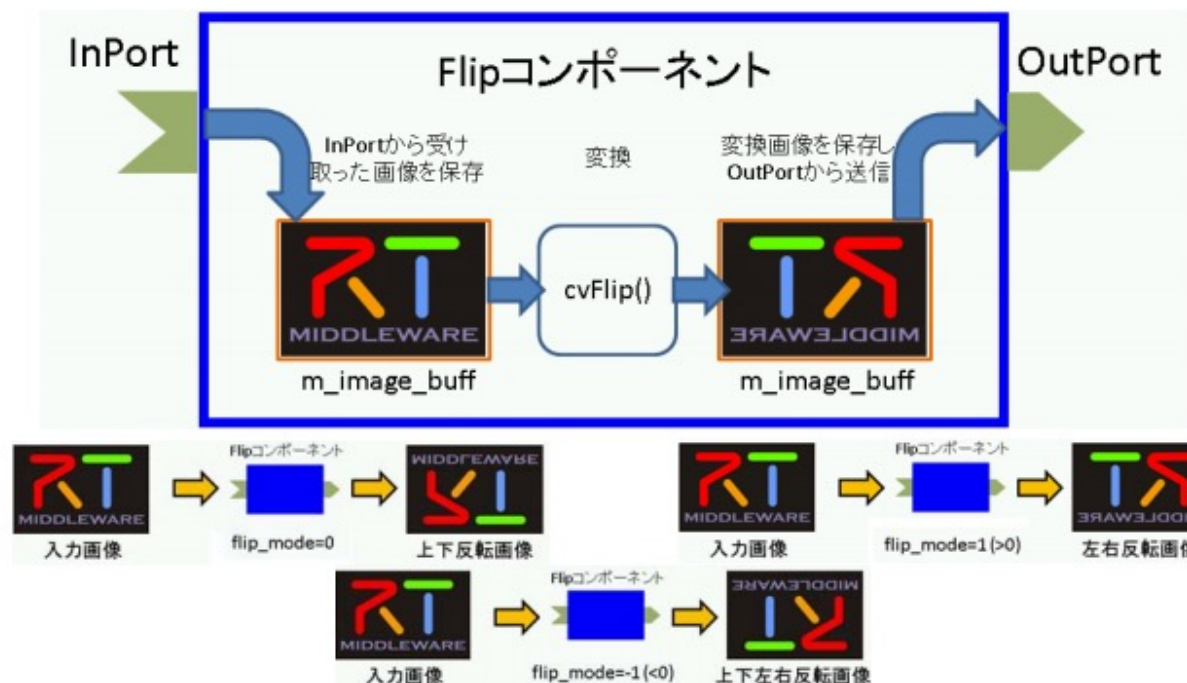
目標：

1. RTCのひな形コードをRTCBuilderを使って生成する方法を学ぶ。
2. RTCのコアロジックを実装しコンパイルする方法を学ぶ。
3. 作成したRTCを他のRTCでシステムを構築する方法を学ぶ。

(1) RTCBuilderによるひな形 コードの生成

実習内容

- 画像の反転を行うコンポーネントの作成
 - InPortで受信した画像データを処理してOutPortから出力
 - データポートの使用方法を習得
 - コンフィギュレーションパラメータにより反転する方向を設定
 - コンフィギュレーションパラメータの使用方法を習得
 - RT System Editorにより他のRTCと接続、RTCをアクティブ化
 - RT System Editorの使い方を習得



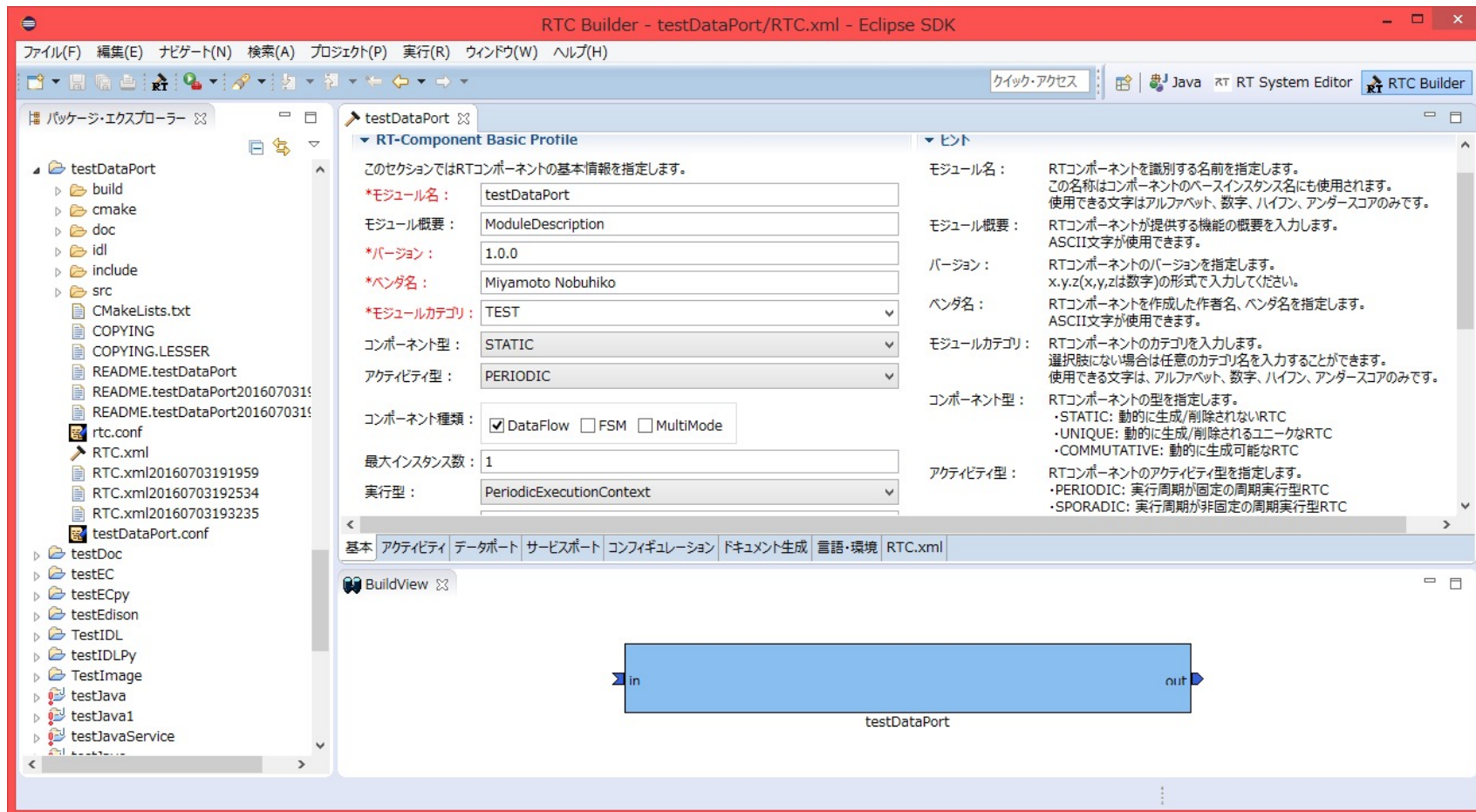
全体の手順

- 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
 - CMakeLists.txtの編集
 - 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
 - Visual Studio
- 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 RTCBuilderについて

RTCBuilder

- 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のソースコードを出力



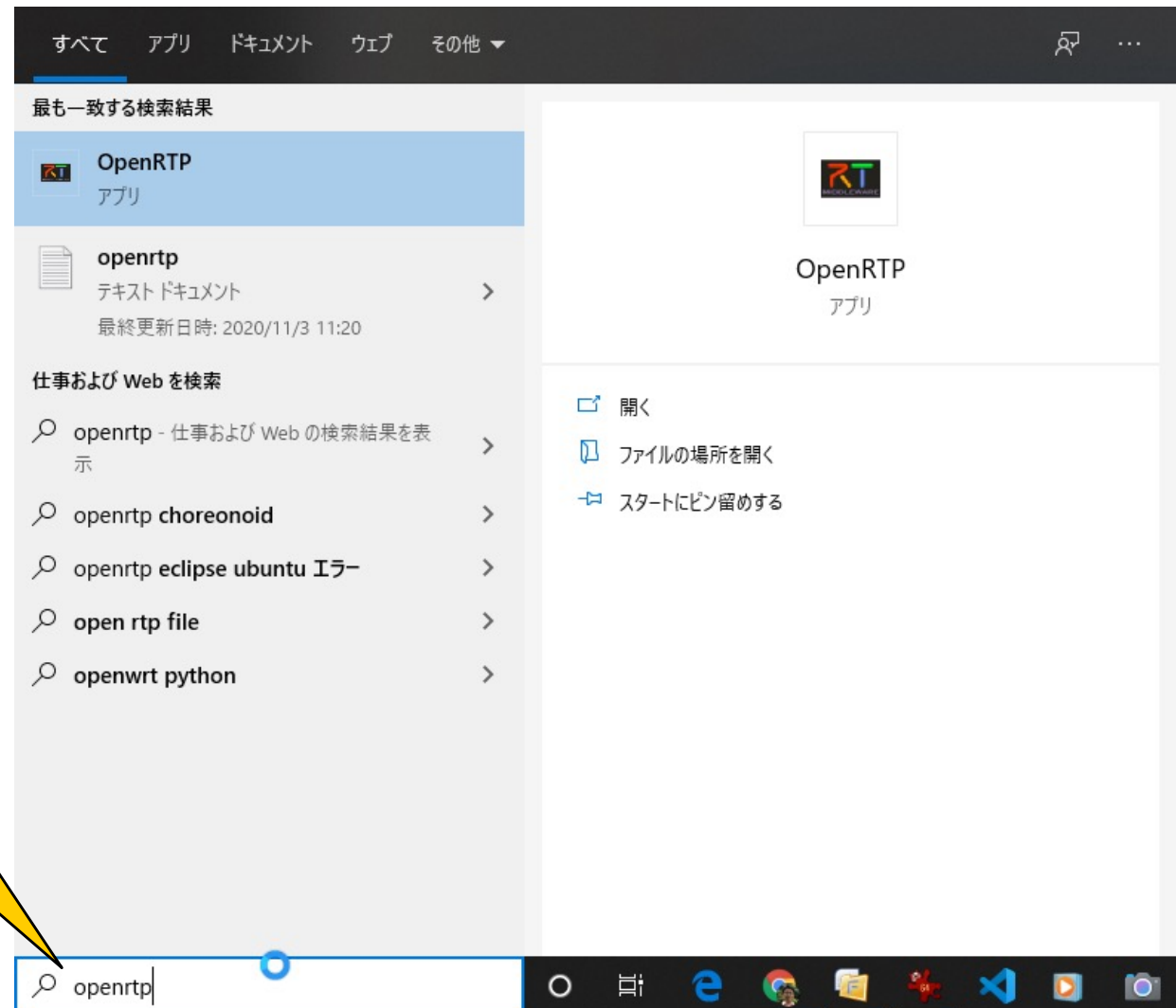
RTC Builderの起動

- 起動する手順
 - Windows 10
 - 左下検索窓に「OpenRTP」を入力
 - Ubuntu
 - 以下のコマンドを入力
 - \$./openrtp

RTC Builderの起動

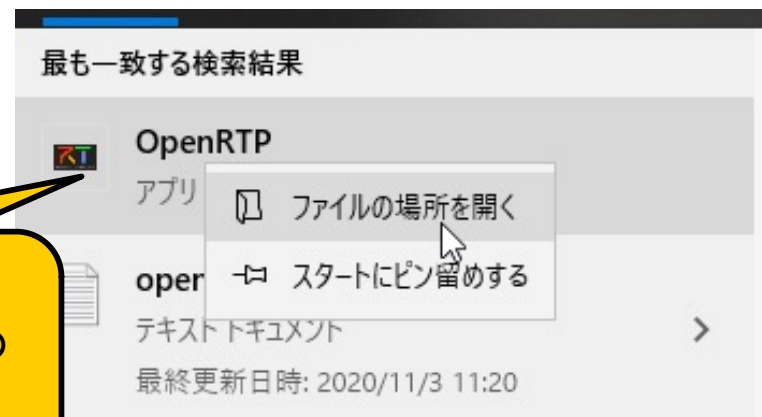
- Windows 10

左下の検索窓に"openrtp"と入力すると"OpenRTP"が候補として出てくる

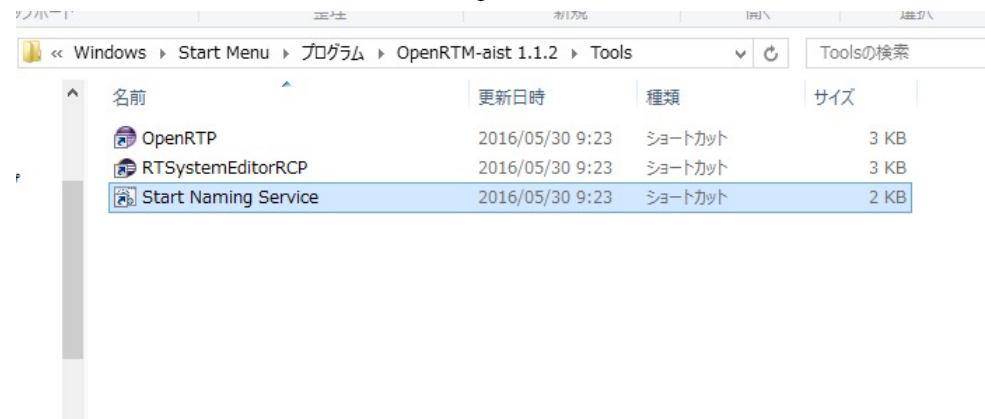


RTC Builderの起動

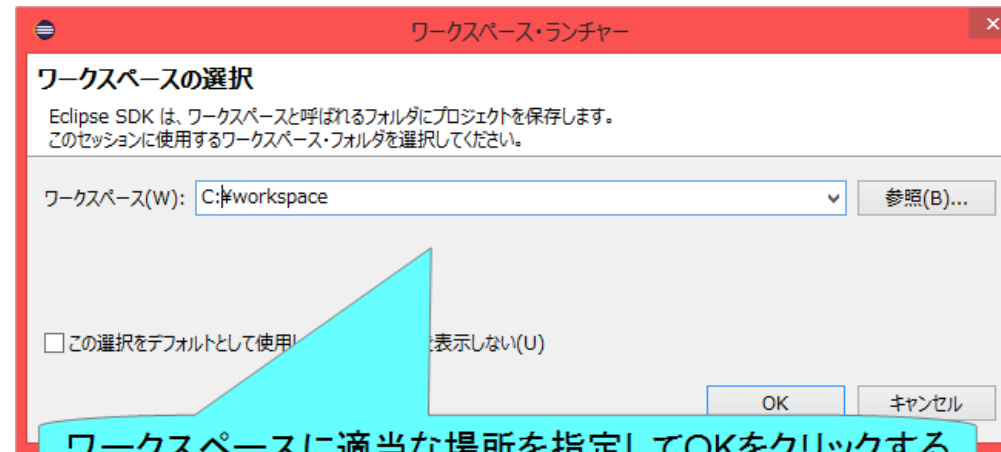
- いちいちアプリビューから起動するのは非常に手間がかかるため、以下の作業をしてスタートメニューのフォルダを開いておくことをお勧めします。



ここを右クリックして「ファイルの場所を開く」を選択



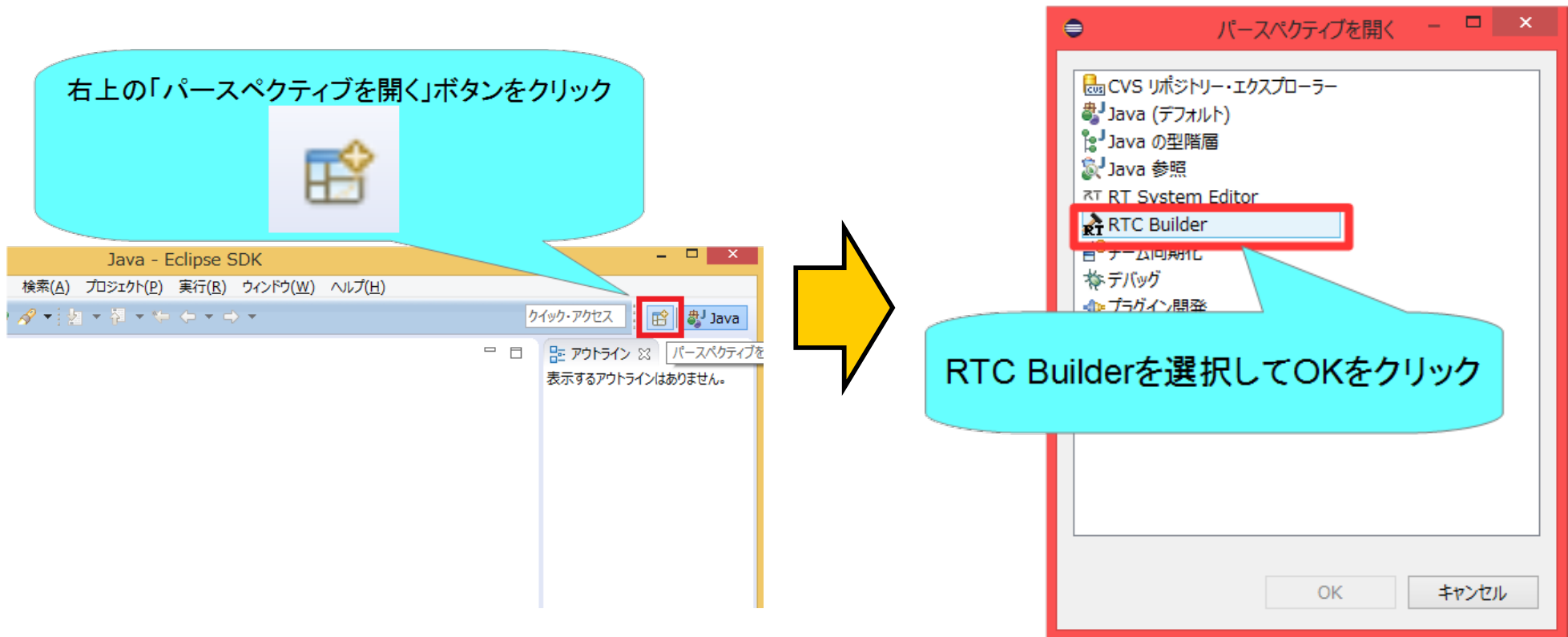
RTC Builderの起動



最初に起動したときはWelcomeページが開くので×を押して閉じる

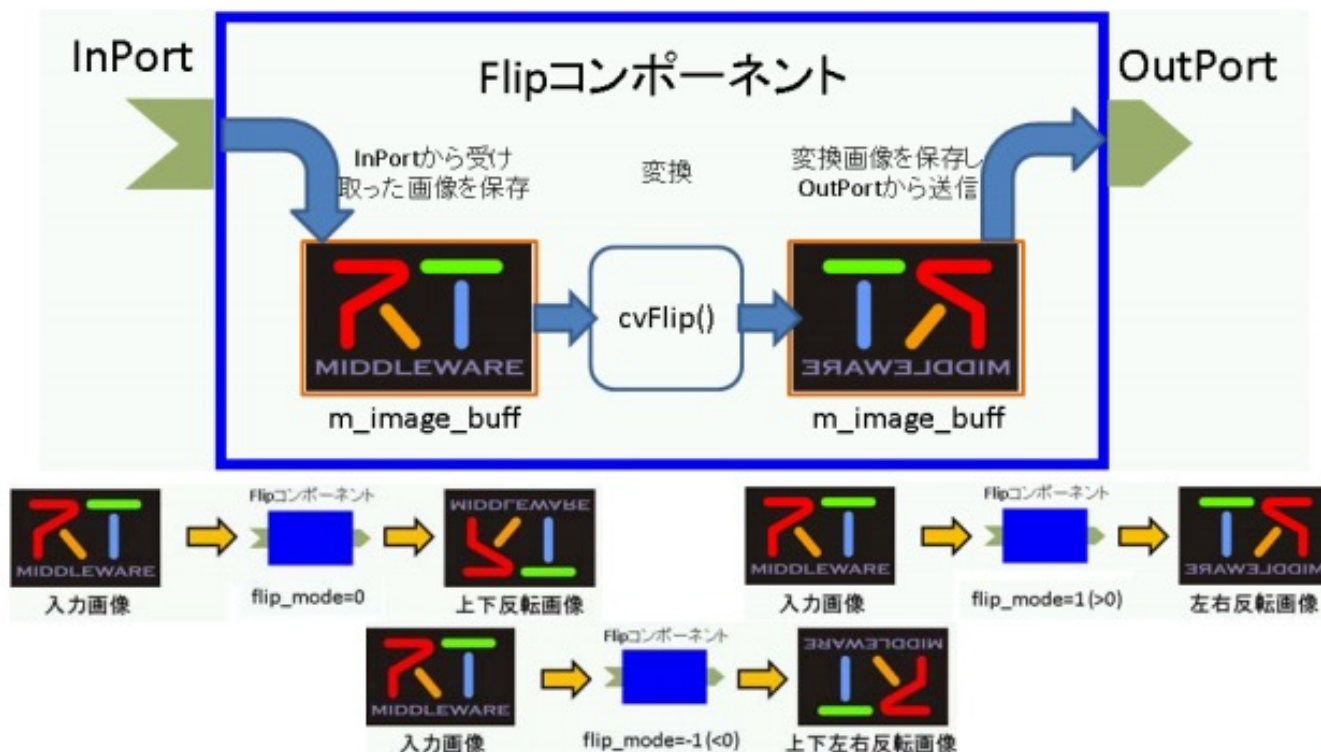


RTC Builderの起動



プロジェクト作成

- Flipコンポーネントのひな形コードを作成する。
 - 画像の反転を行うコンポーネント
 - InPortで受信した画像データ进行处理してOutPortから出力
 - コンフィギュレーションパラメータにより反転する方向を設定
 - RT System Editorにより他のRTCと接続、RTCをアクティブ化

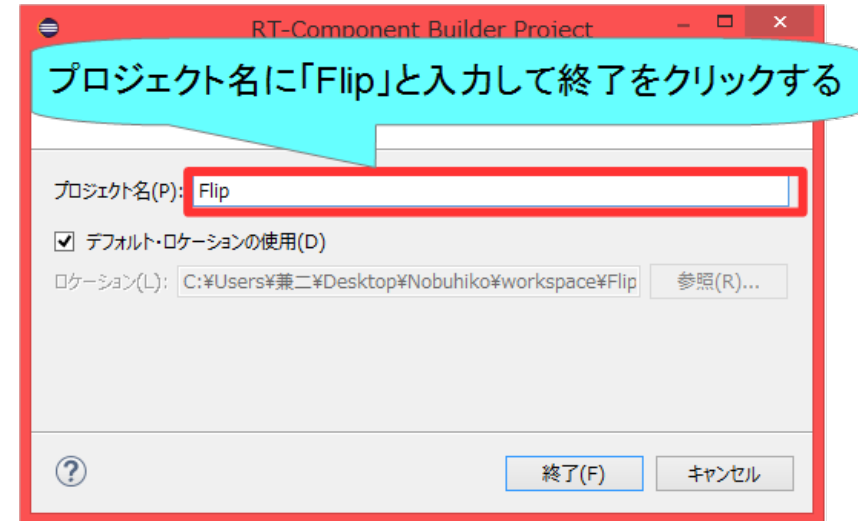
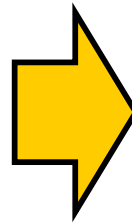
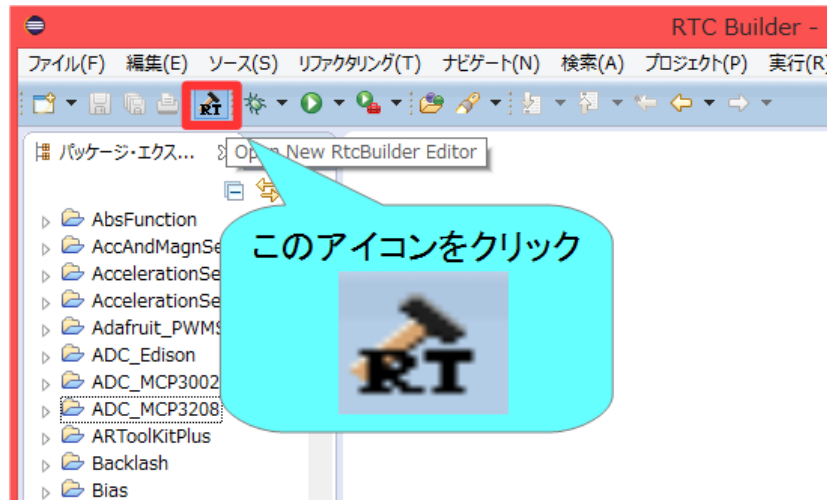


資料

- 右図のようにOpenRTM-aist公式サイトの本日の講習会のページから、チュートリアルを開く
 - <https://bit.ly/3EI3peN>
- Windowsで開発を行いますので「チュートリアル（画像処理コンポーネントの作成: Windows）」を開いてください
 - 参考のためLinux版のチュートリアルも掲載しています
- Openrtm.orgのメニューからは
 - ドキュメント→ケーススタディ→画像処理コンポーネント
 - で到達可能

11月09日（火）	
10:00 -10:50	1. コース概要 (1) ロボットシステムプログラミングの現状 (2) ロボットOS・ミドルウェア (3) RTミドルウェア(RTM)を用いたロボット開発 資料: 211109-01.pdf
11:00 -11:45	2. プログラミングの基礎 (1) OpenRTM-aistのインストール (2) RTCプログラミング概要 ・インストールするソフトウェア ・(参考)Windowsへのインストール ・OpenRTMを10分で始めよう 資料: 211109-02.pdf
11:45 -12:30	昼食
12:30-16:45	3. RTCプログラミング演習 (1) RTCBuilderによるひな形コードの生成 (2) プログラムの作成とコンパイル (3) システム構築とテスト ・チュートリアル（画像処理コンポーネントの作成: Windows） ・チュートリアル（画像処理コンポーネントの作成: Linux） 資料: 211109-03.pdf

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
- 以下の項目を設定する
 - 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

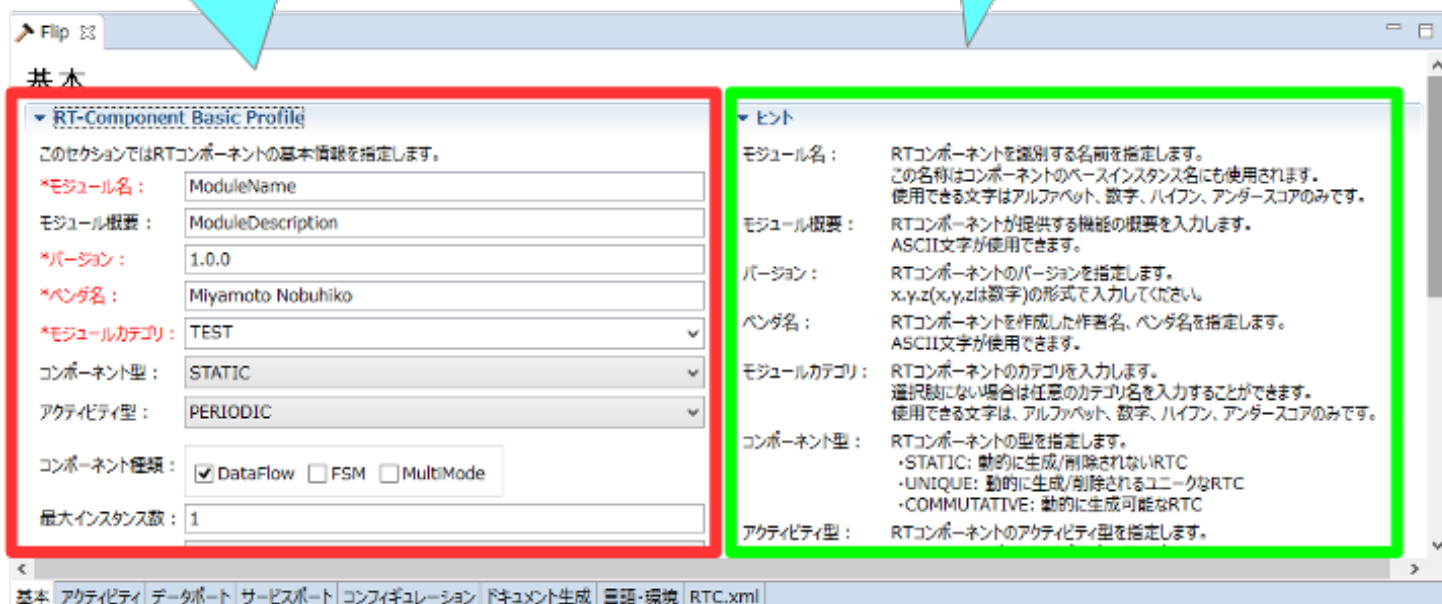
基本プロファイルの入力

- 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

RTCプロファイルエディタ
ここに各項目を入力する

ヒント

「基本」タブを選択



基本プロファイルの入力

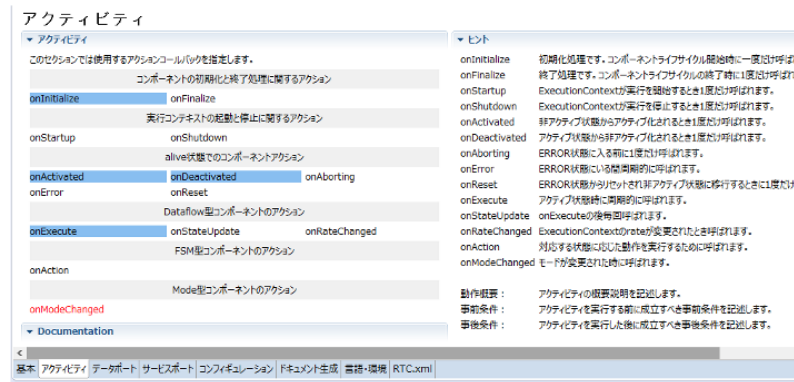
- モジュール名
 - Flip
- モジュール概要
 - 任意(Flip image component)
- バージョン
 - 任意(1.0.0)
- ベンダ名
 - 任意
- モジュールカテゴリ
 - 任意(ImageProcessing)
- コンポーネント型
 - STATIC
- アクティビティ型
 - PERIODIC
- コンポーネントの種類
 - DataFlow
- 最大インスタンス数
 - 1
- 実行型
 - PeriodicExecutionContext
- 実行周期
 - 1000.0
- 概要
 - 任意

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 :	Flip
モジュール概要 :	Flip image component
*バージョン :	1.0.0
*ベンダ名 :	AIST
*モジュールカテゴリ :	ImageProcessing ▼
コンポーネント型 :	STATIC ▼
アクティビティ型 :	PERIODIC ▼
コンポーネント種類 :	☒ DataFlow ☐ FSM ☐ MultiMode
最大インスタンス数 :	1
実行型 :	PeriodicExecutionContext ▼
実行周期 :	1000.0
概要 :	
RTC Type :	

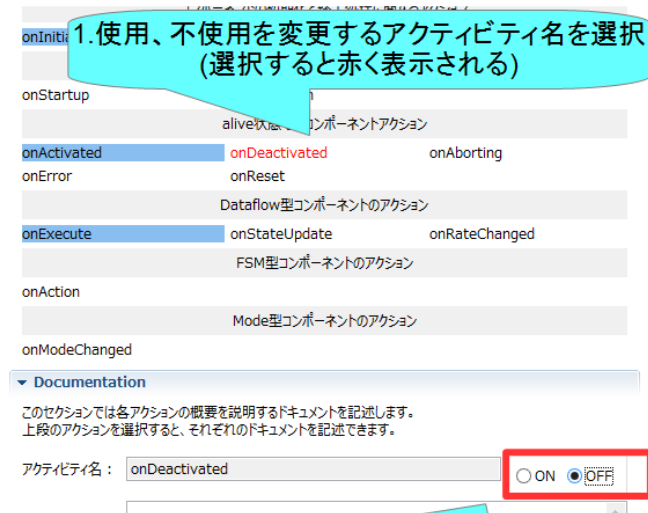
アクティビティの設定

- 使用するアクティビティを設定する

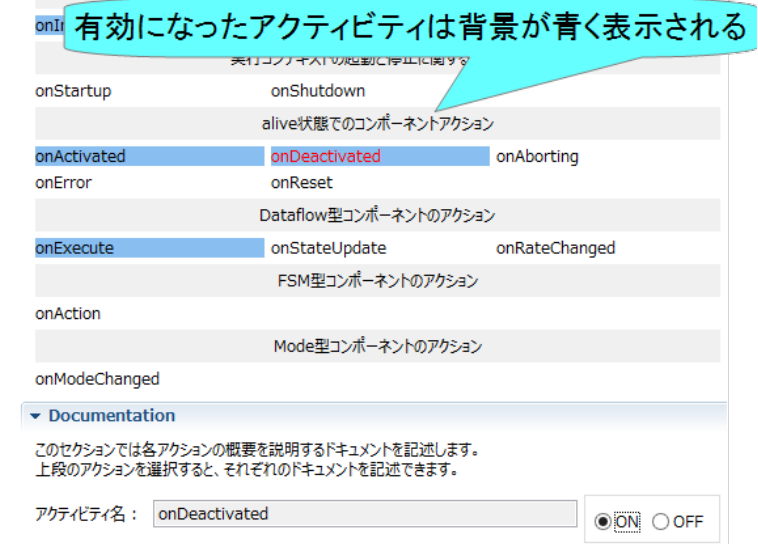


「アクティビティ」タブを選択

- 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

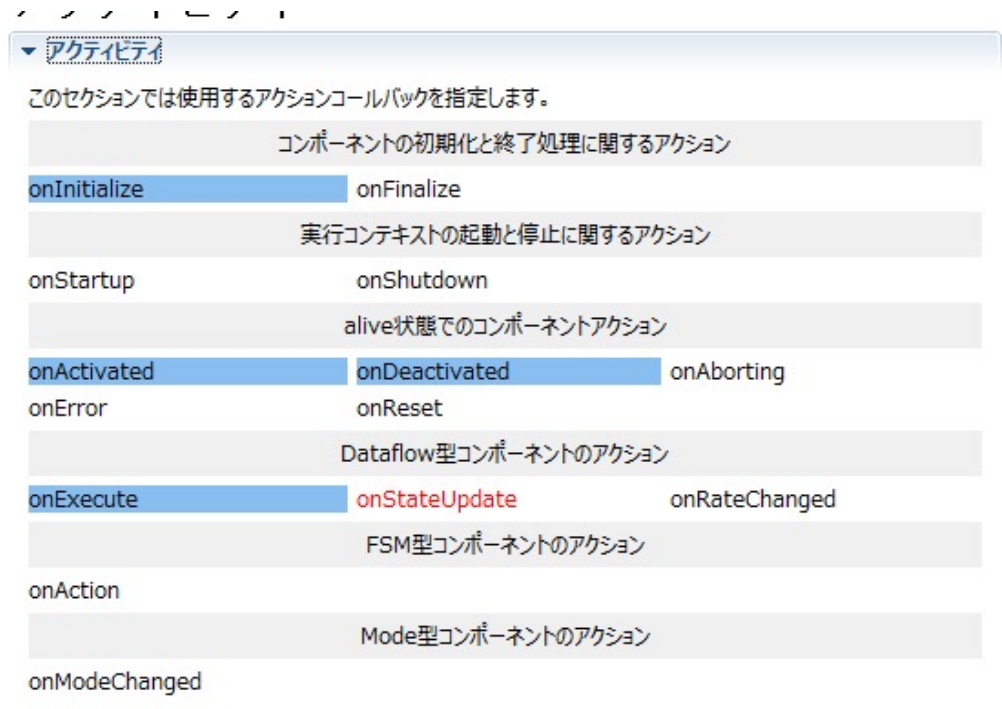


アクティビティの設定

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
- Documentationは適当に書いておいてください
 - 空白でも大丈夫です



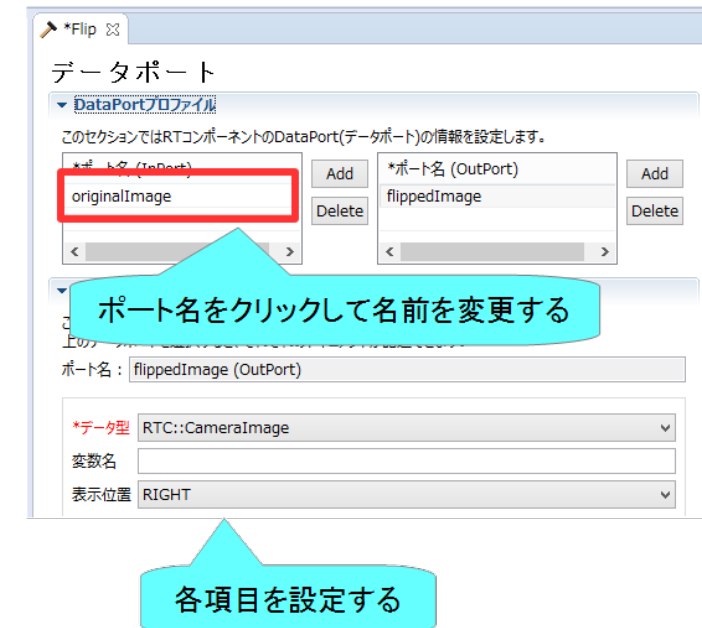
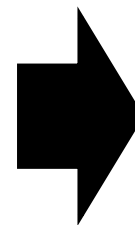
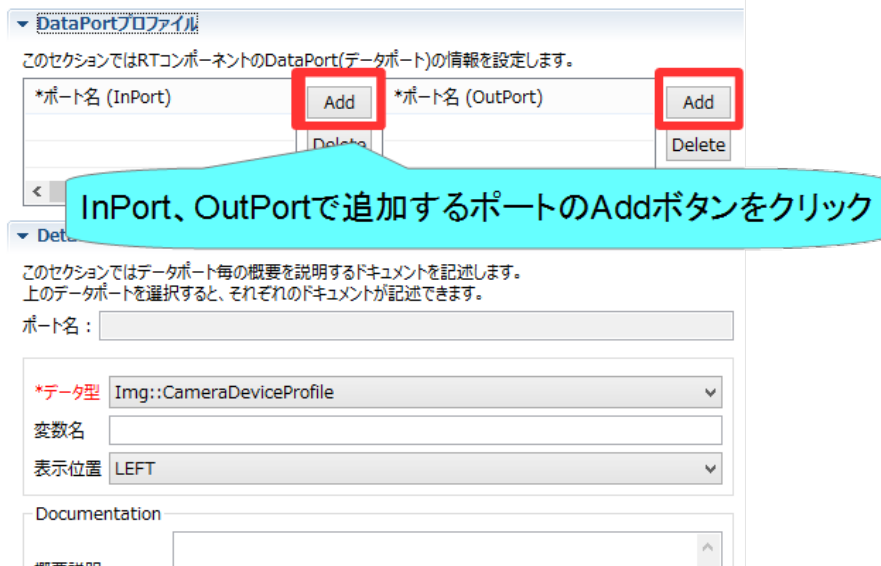
データポートの設定

- InPort、OutPortの追加、設定を行う



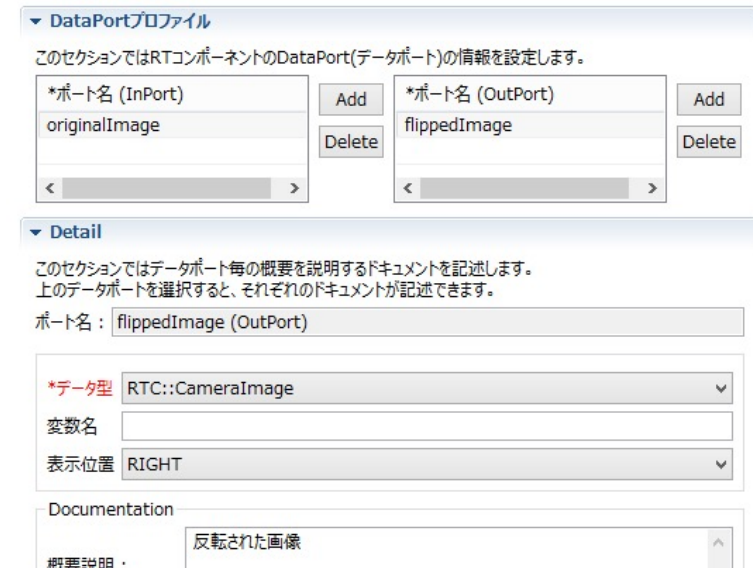
「データポート」タブを選択

- 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
- 以下のOutPortを設定する
 - flippedImage
 - データ型 :
RTC::CameraImage
 - 他の項目は任意
- ※今回使用するのは
RTC::CameraImageなので
Img::CameraImageと間違えないようにする。
- ※ポート名を間違えないようにしてください。



The screenshot shows the 'DataPortプロファイル' (DataPort Profile) configuration window. It is divided into two main sections: 'DataPortプロファイル' and 'Detail'.

DataPortプロファイル

このセクションではRTコンポーネントのDataPort(データポート)の情報を設定します。

On the left, under '*ポート名 (InPort)', there is a list containing 'originalImage' and an 'Add' button. On the right, under '*ポート名 (OutPort)', there is a list containing 'flippedImage' and an 'Add' button. Both lists have 'Delete' buttons and scrollbars.

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。

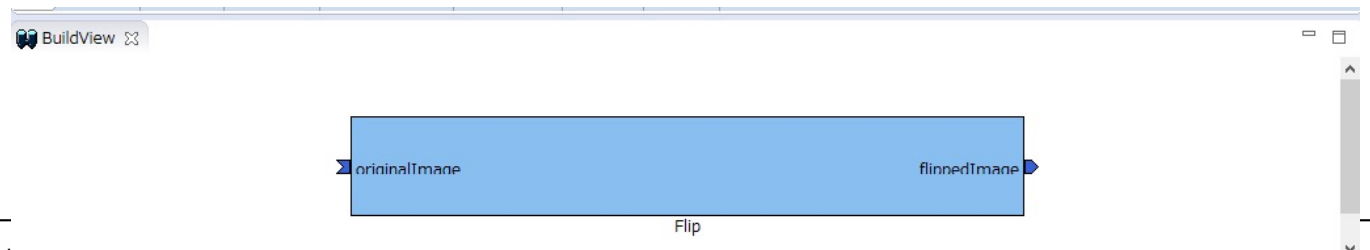
ポート名: flippedImage (OutPort)

*データ型: RTC::CameraImage (dropdown menu)

変数名: (empty text field)

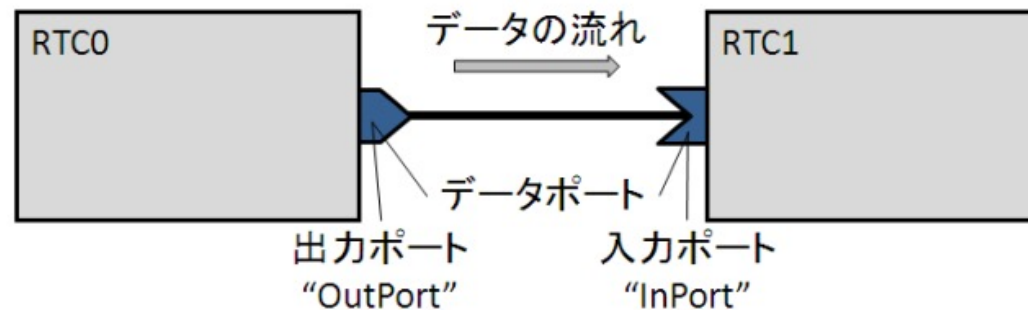
表示位置: RIGHT (dropdown menu)

Documentation: 反転された画像 (text area)

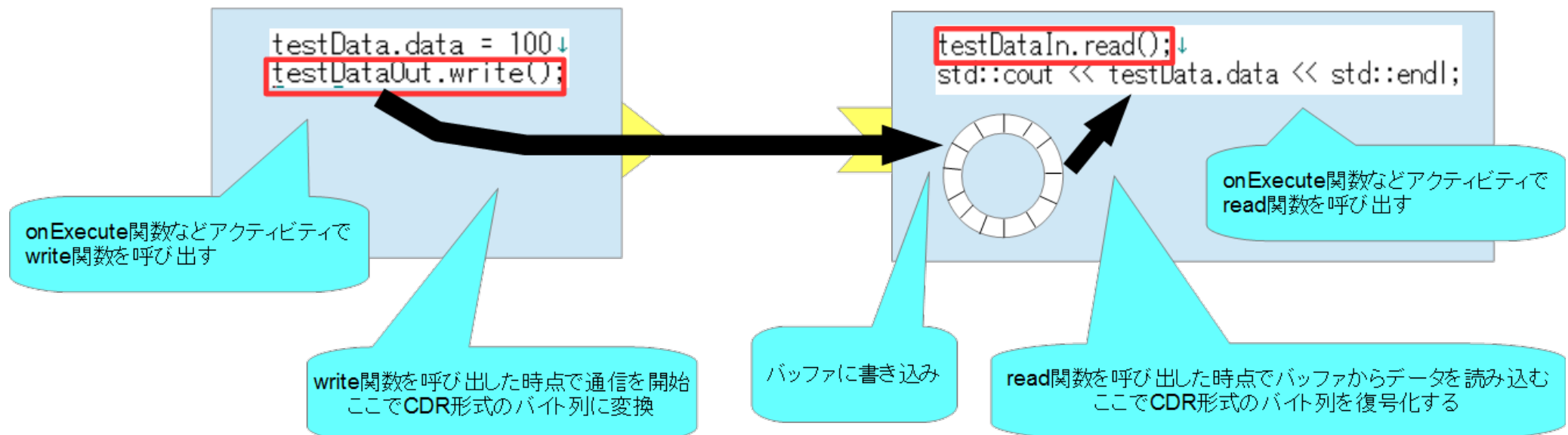


データポートについて

- 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::CameraImage型について

- InterfaceDataTypes.idlで定義されている画像データ通信のためのデータ型

タイムスタンプを格納する。

```
struct Time
{
    unsigned long sec; // sec
    unsigned long nsec; // nano sec
};
```

画像の幅を格納する。

画像の高さを格納する。

1ピクセル当たりのbit数を格納する。

```
struct CameraImage↓
{↓
    Time tm;↓
    unsigned short width;↓
    unsigned short height;↓
    unsigned short bpp;↓
    string format;↓
    ↓
    double fDiv;↓
    ↓
    sequence<octet> pixels;↓
};↓
```

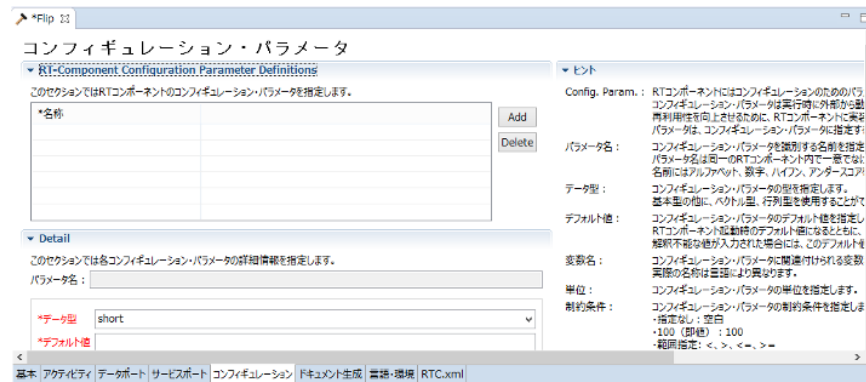
画像フォーマット (bitmap、jpeg等) の名前を格納。
今回は使用しない。

スケールファクタを格納する。
今回は使用しない。

画像データを格納する。
sequenceは配列のようなデータを扱うときに使用する。
この場合、octet型の可変長配列を利用できる。
※octet型はC++ではunsigned char型にマッピングされている

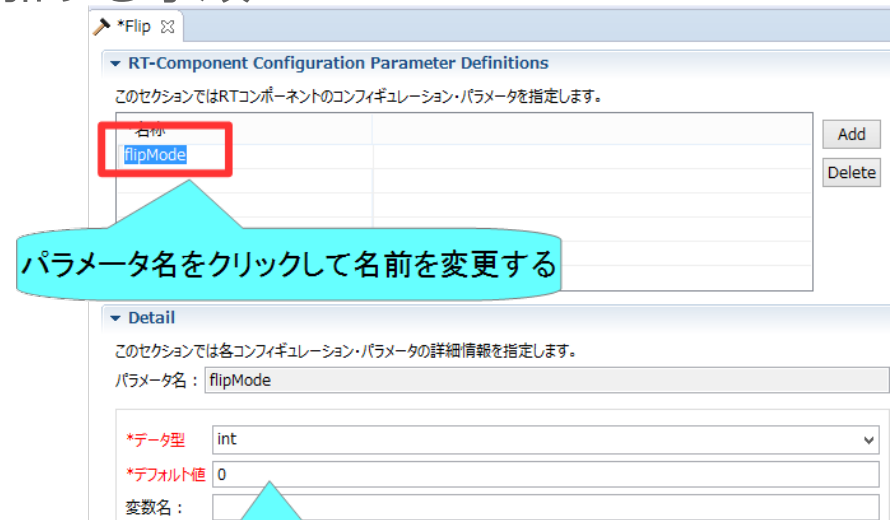
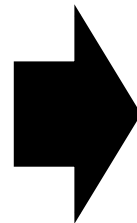
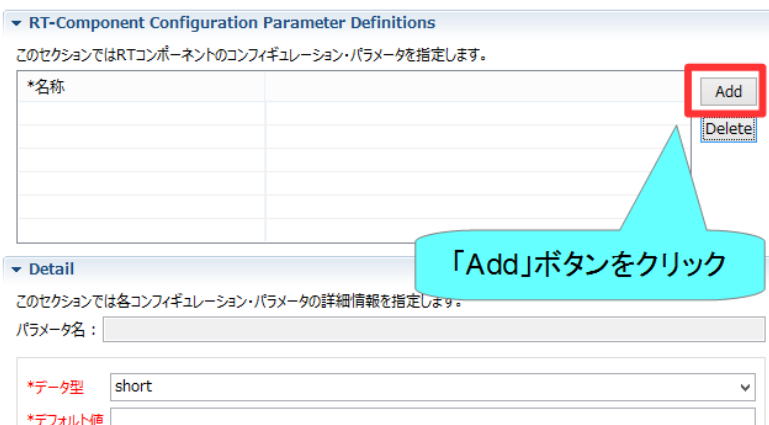
コンフィギュレーションの設定

- 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

*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

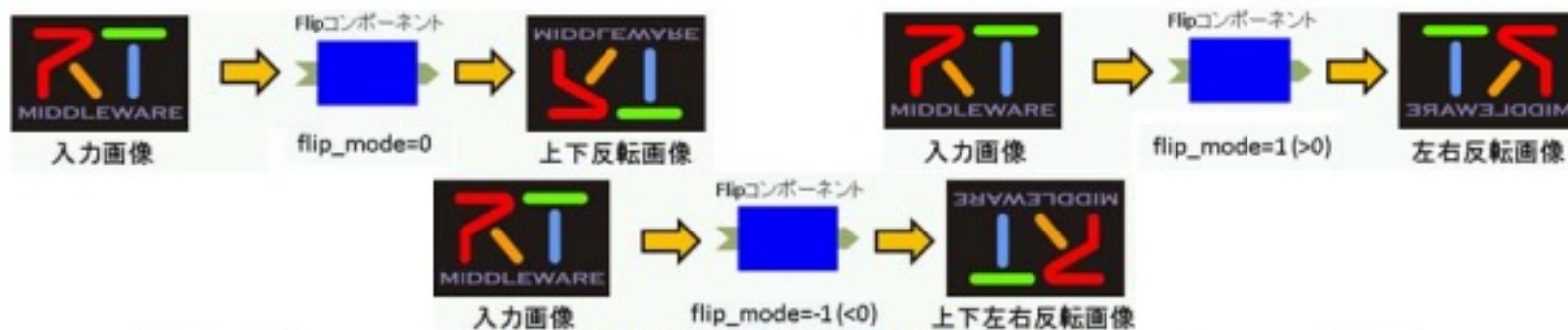
変数名 :

単位 :

制約条件 : (0,-1,1)

Widget : radio

Step :



- 反転する方向を設定可能にする

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 :
 $0 \leq x \leq 100$
- 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

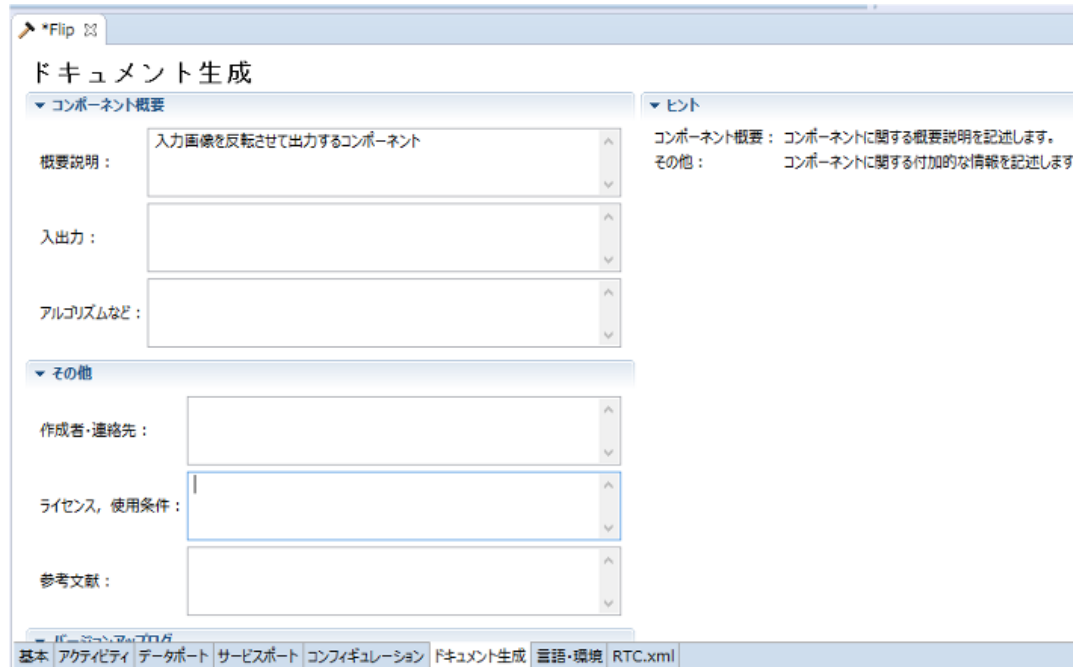
>
 <

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



ドキュメント生成

▼ コンポーネント概要

概要説明 : 入力画像を反転させて出力するコンポーネント

入出力 :

アルゴリズムなど :

▼ 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。

▼ その他

作成者・連絡先 :

ライセンス, 使用条件 :

参考文献 :

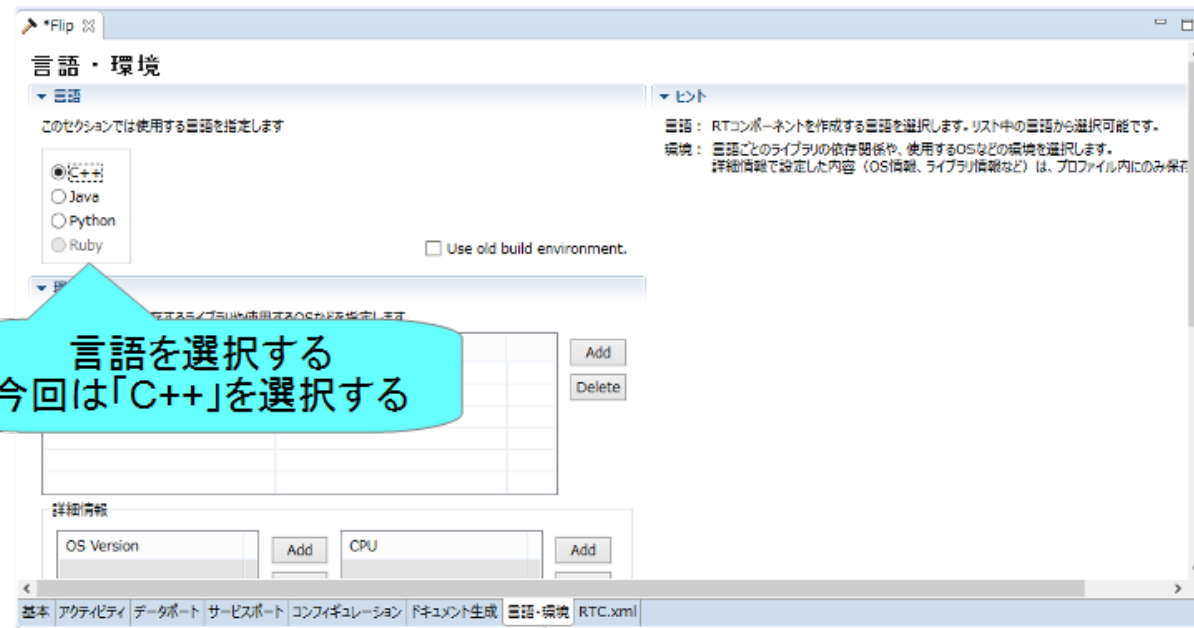
基本 アクティビティ データポート サービスポート コンフィギュレーション ドキュメント生成 言語・環境 RTC.xml

「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

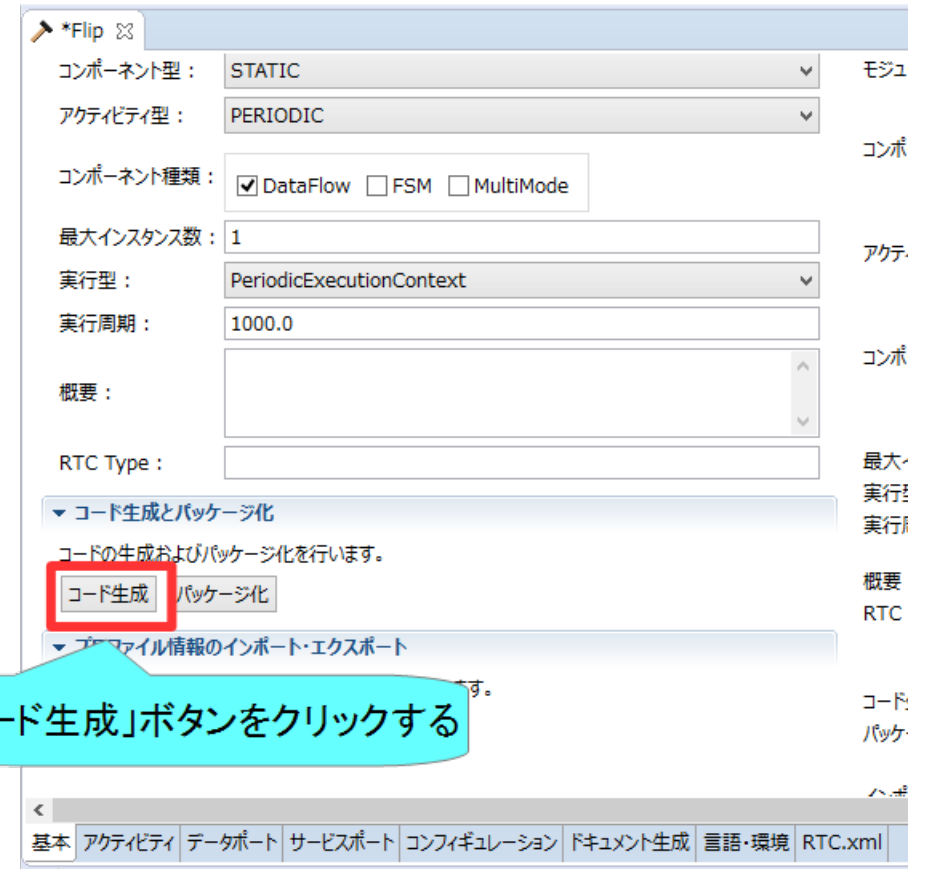
- 実装する言語，動作環境に関する情報を設定



「言語・環境」タブを選択

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画像を反転させる処理を記述する
 - CMakeの設定ファイル
 - CMakeLists.txt
 - rtc.conf、Flip.conf
 - 以下略
 - ファイルが生成できているかを確認してください



*Flip

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

実行型: PeriodicExecutionContext

実行周期: 1000.0

概要:

RTC Type:

▼ コード生成とパッケージ化

コードの生成およびパッケージ化を行います。

コード生成 パッケージ化

▼ ファイル情報のインポート・エクスポート

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

(2) プログラムの作成とコンパイル

手順

- ビルドに必要な各種ファイルを生成
 - CMakeLists.txtの編集
 - 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

CMake

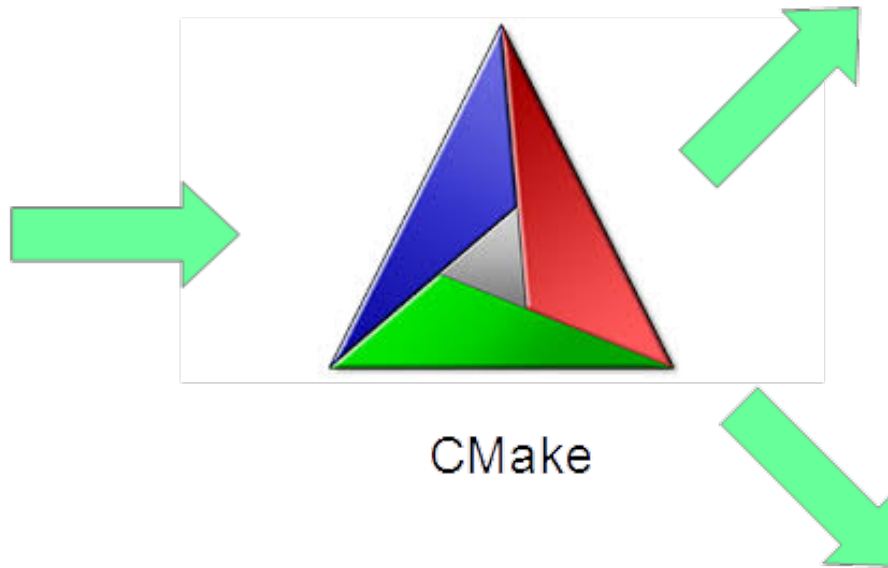
- ビルドに必要な各種ファイルを生成
 - CMakeLists.txtに設定を記述
 - RTC Builderでスケルトンコードを作成した時にCMakeLists.txtも生成されている



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



CMakeLists.txt



CMake



Makefile

CMakeLists.txtの編集

- OpenCVを利用するためにCMakeLists.txtを修正する
 - workspace¥Flipの**srcフォルダ**のCMakeLists.txtをメモ帳などで開いて編集する

```

1 set(comp_srcs Flip.cpp) ↓
2 set(standalone_srcs FlipComp.cpp) ↓
3 ↓
4 find_package(OpenCV REQUIRED) ↓
5 ↓
6 if (DEFINED OPENRTM_INCLUDE_DIRS) ↓
7     string(REGEX REPLACE "-I" ";" ↓
8         OPENRTM_INCLUDE_DIRS "${OPENRTM_INCLUDE_DIRS}") ↓
9     list(APPEND OPENRTM_INCLUDE_DIRS "-I${CMAKE_CURRENT_SOURCE_DIR}/include")

```

find_package(OpenCV REQUIRED)
を追加

```

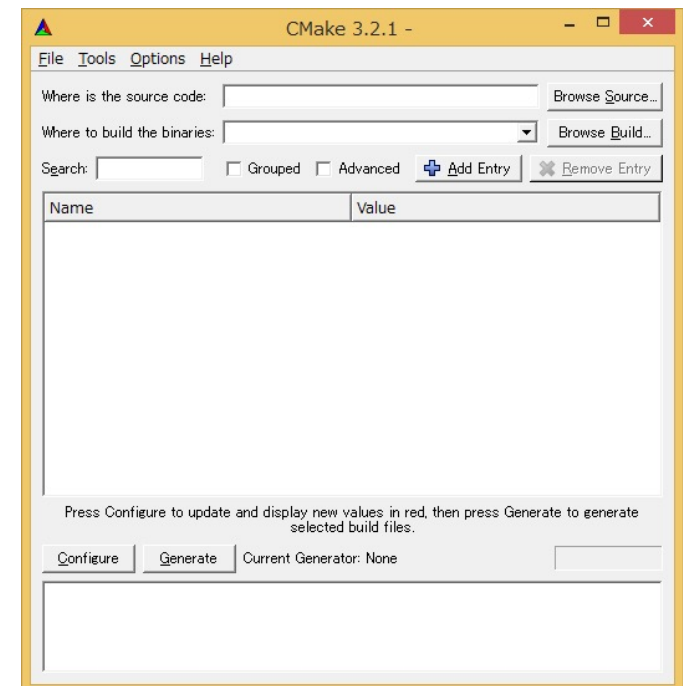
47 endif(NOT TARGET ALL_IDL_TGT) ↓
48 add_dependencies(${PROJECT_NAME} ALL_IDL_TGT) ↓
49 target_link_libraries(${PROJECT_NAME} ${OPENRTM_LIBRARIES} ${OpenCV_LIBS}) ↓
50 ↓
51 add_executable(${PROJECT_NAME}Comp ${standalone_srcs} ↓
52     ${comp_srcs} ${comp_headers} ${ALL_IDL_SRCS}) ↓
53 target_link_libraries(${PROJECT_NAME}Comp ${OPENRTM_LIBRARIES} ${OpenCV_LIBS})
54 ↓

```

target_link_librariesの引数に\${OpenCV_LIBS}を追加
※2箇所あるので注意

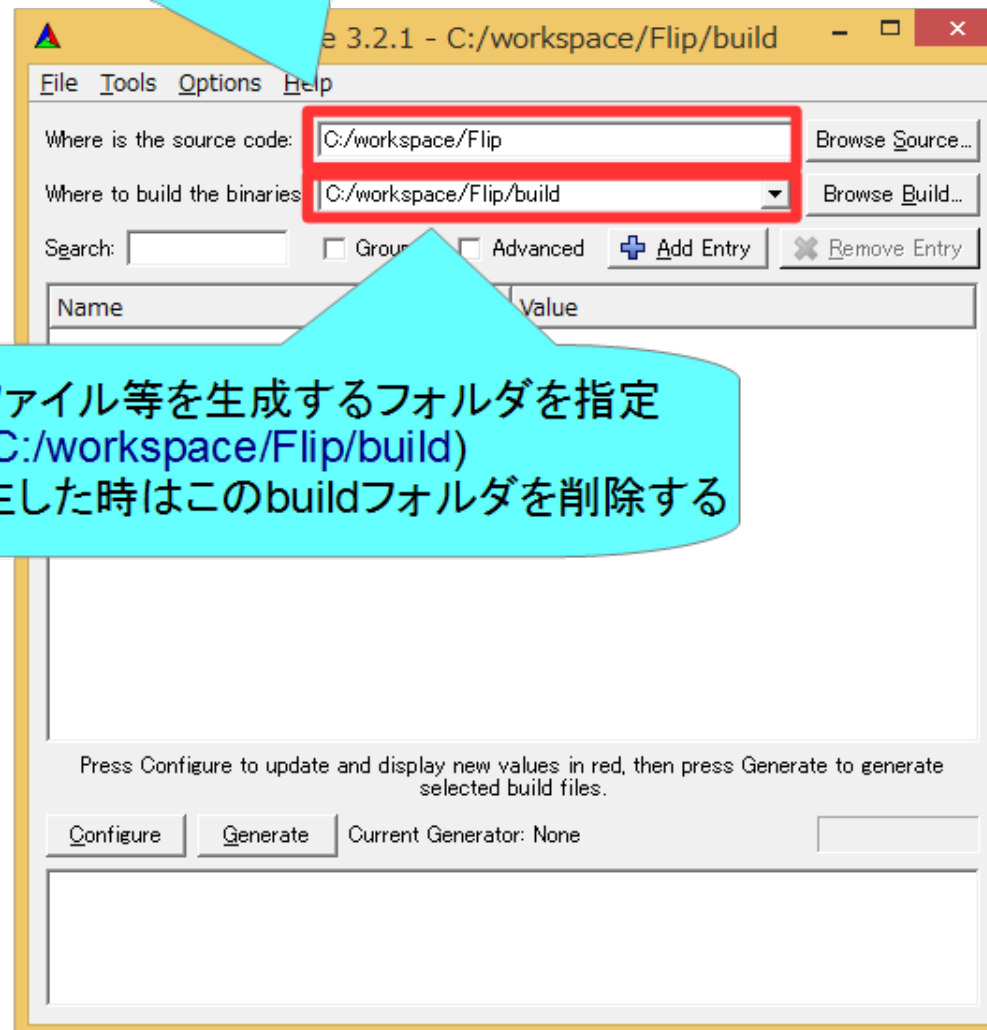
ビルドに必要なファイルの生成

- CMakeを使用する
 - Windows 7
 - 「スタート」→「すべてのプログラム」→「CMake 3.5.2」→「CMake (cmake-gui)」
 - Windows 8.1
 - 「スタート」→「アプリビュー(右下矢印)」→「CMake 3.5.2」→「CMake (cmake-gui)」
 - Windows 10
 - 右下検索窓「cmake」→「CMake (cmake-gui)」
 - Ubuntu
 - コマンドで「cmake-gui」を入力



ビルドに必要なファイルの生成

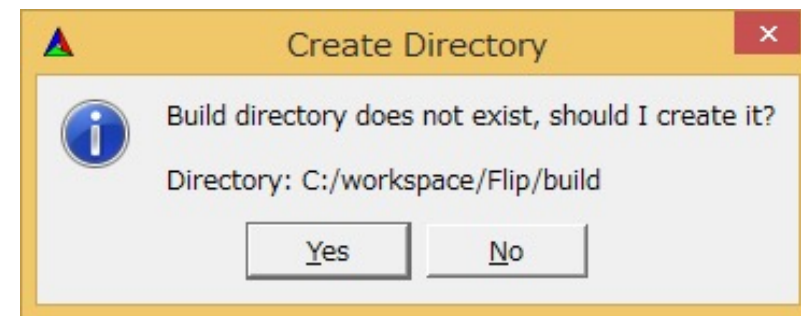
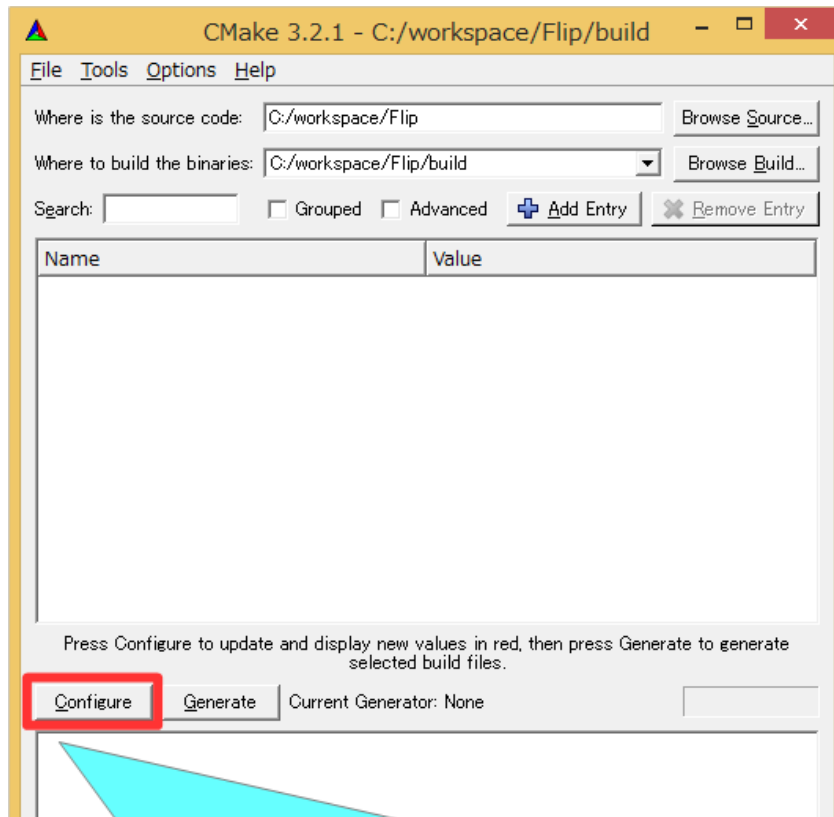
RTC Builderで生成したプロジェクトのフォルダを指定
(例: C:/workspace/Flip)



The screenshot shows the RTC Builder 3.2.1 window with the title bar "RTC Builder 3.2.1 - C:/workspace/Flip/build". The menu bar includes File, Tools, Options, and Help. The main configuration area has two text boxes: "Where is the source code:" with the value "C:/workspace/Flip" and "Where to build the binaries:" with the value "C:/workspace/Flip/build". Both text boxes are highlighted with a red rectangle. To the right of each text box is a button: "Browse Source..." and "Browse Build...". Below these are checkboxes for "Search:", "Group", and "Advanced", along with "Add Entry" and "Remove Entry" buttons. At the bottom, there is a "Configure" button, a "Generate" button, and a "Current Generator: None" label. A message at the bottom states: "Press Configure to update and display new values in red, then press Generate to generate selected build files."

ソリューションファイル等生成するフォルダを指定
(例: C:/workspace/Flip/build)
何かトラブルが発生した時はこのbuildフォルダを削除する

ビルドに必要なファイルの生成



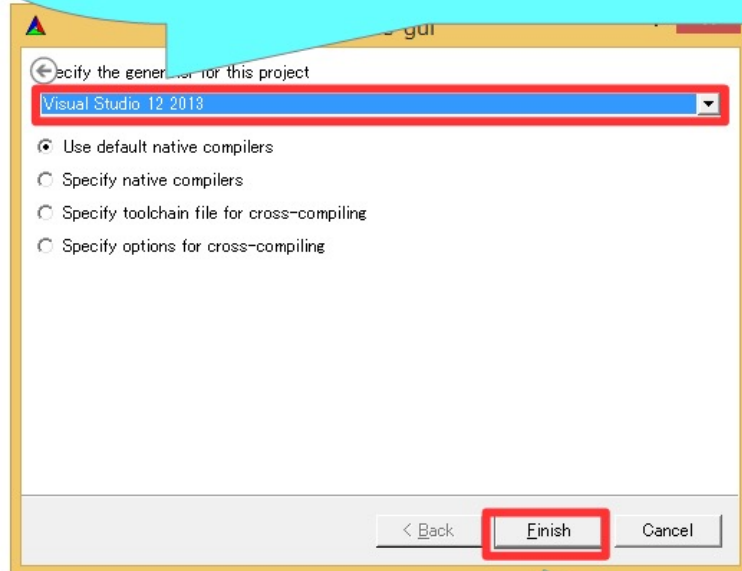
buildフォルダが存在しない場合は
作成するかどうかきかれるため「Yes」を選択

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

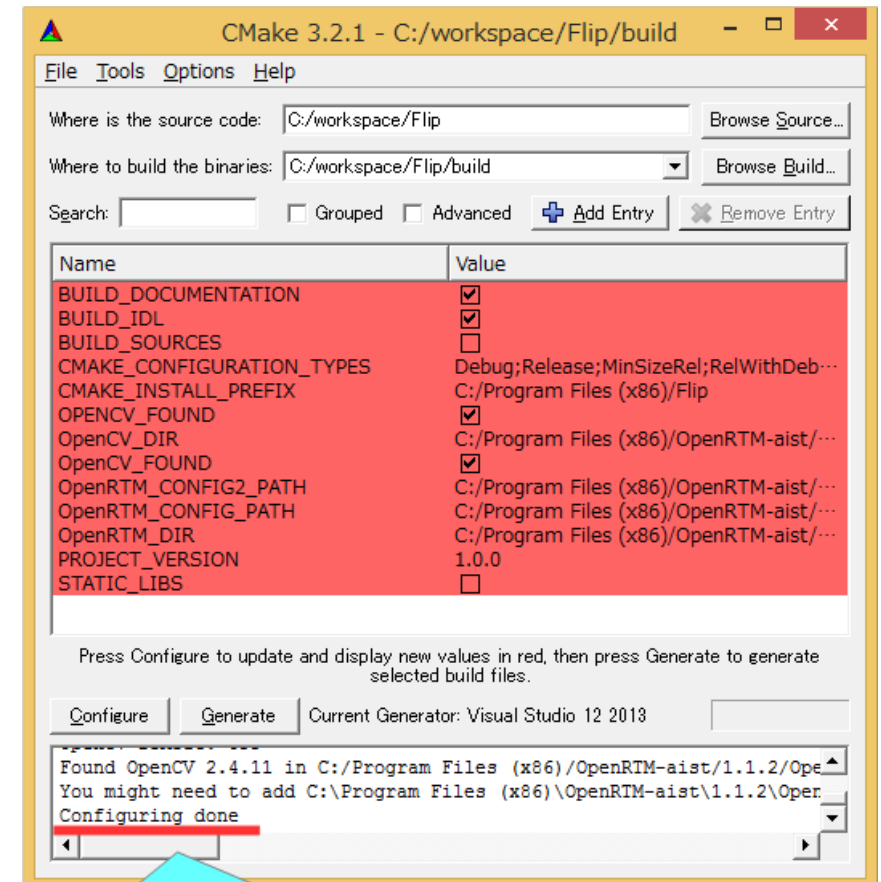
ビルドに必要なファイルの生成

ビルド環境の設定

Visual Studio 2010 32bit → Visual Studio 10 2010
 Visual Studio 2012 32bit → Visual Studio 11 2012
 Visual Studio 2012 64bit → Visual Studio 11 2012 Win 64
 Visual Studio 2013 32bit → Visual Studio 12 2013
 Visual Studio 2013 64bit → Visual Studio 12 2013 Win 64
 Code::Blocks → CodeBlocks -Unix Makefiles

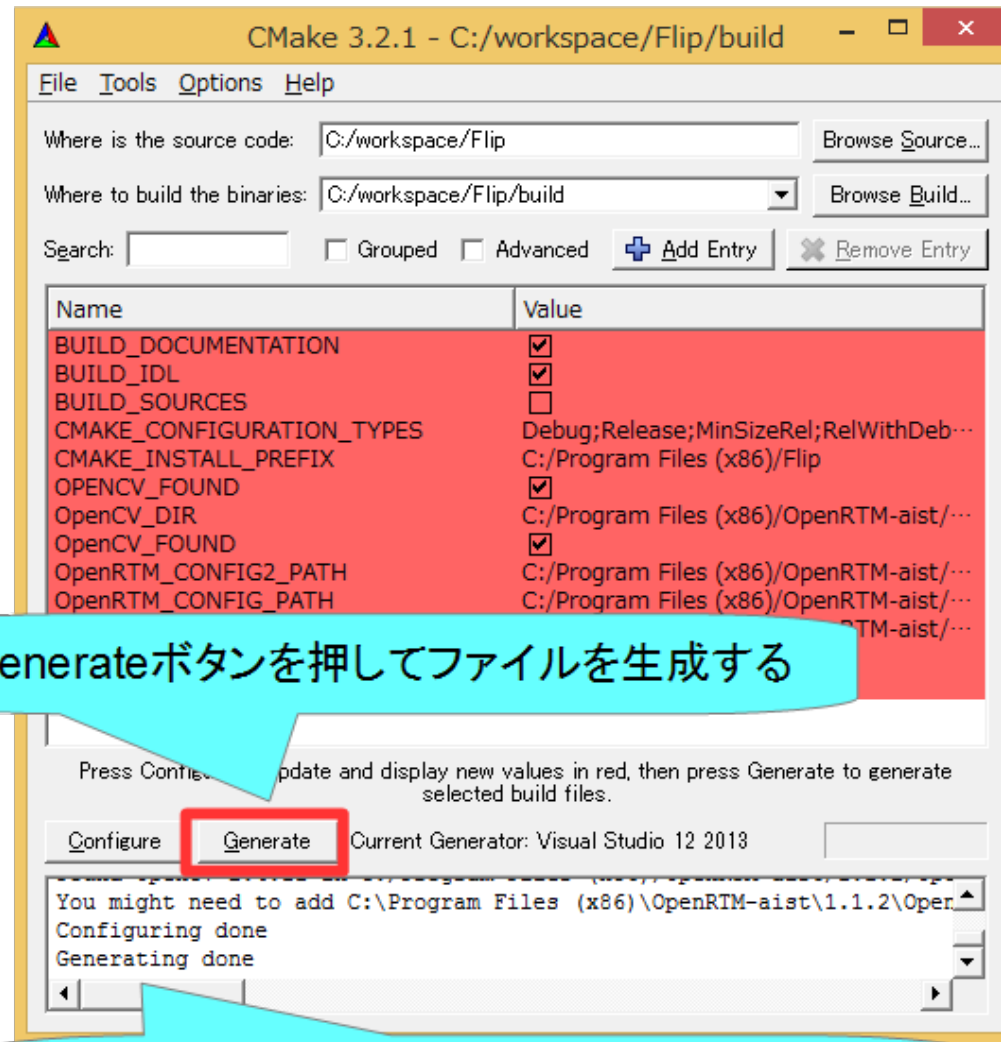


設定後、Finishボタンを押す



「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

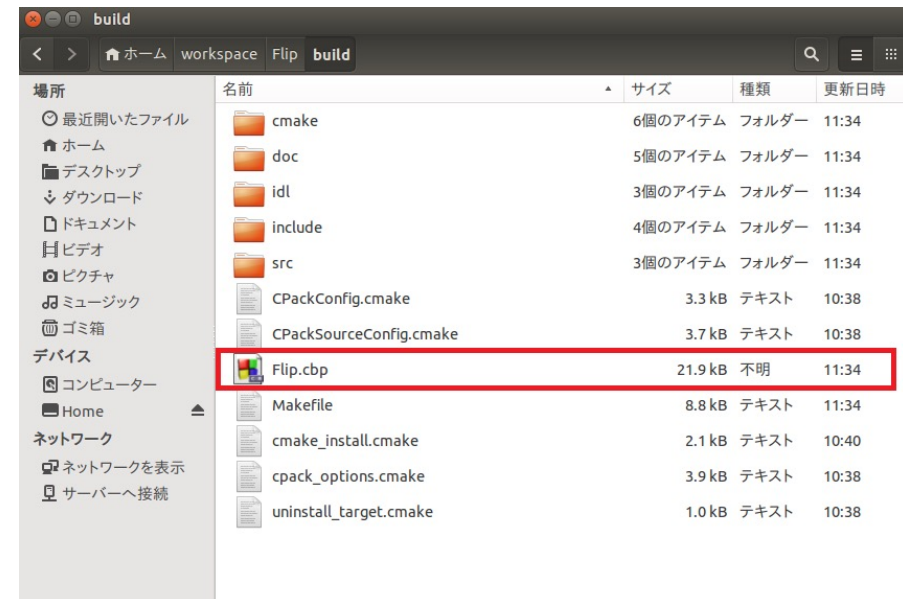
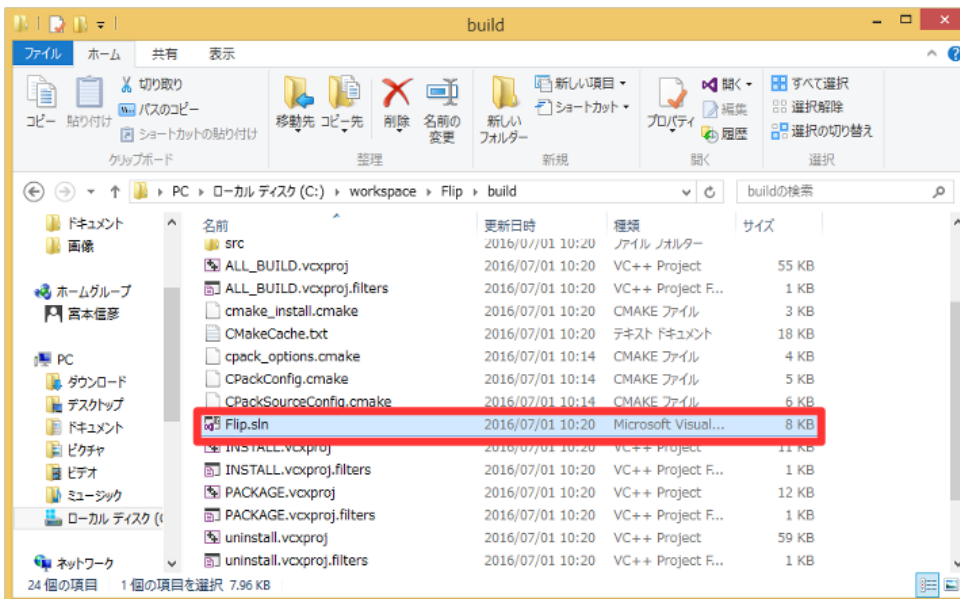


Generateボタンを押してファイルを生成する

「Generating done」が表示されていれば成功

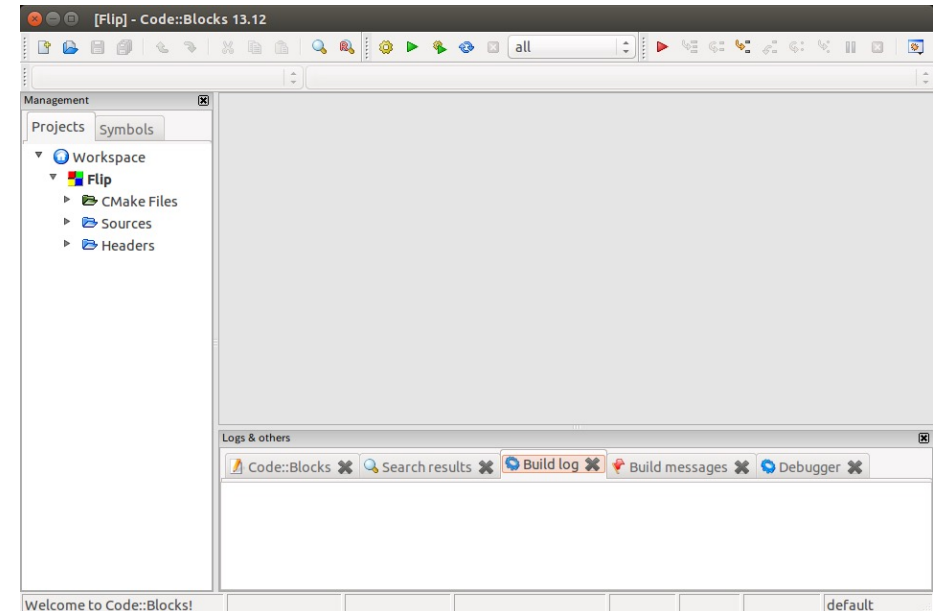
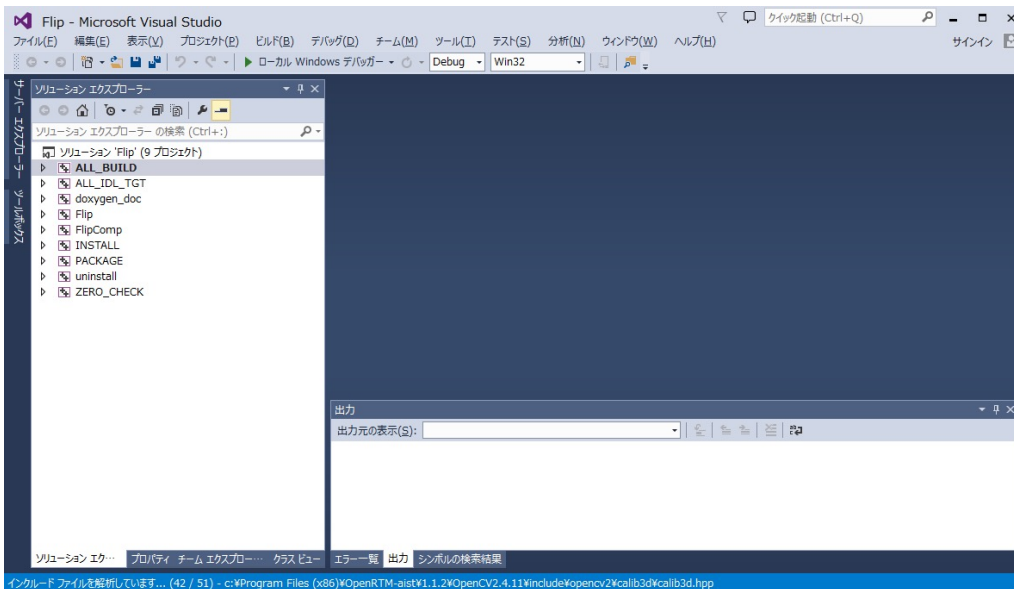
ソースコードの編集

- 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

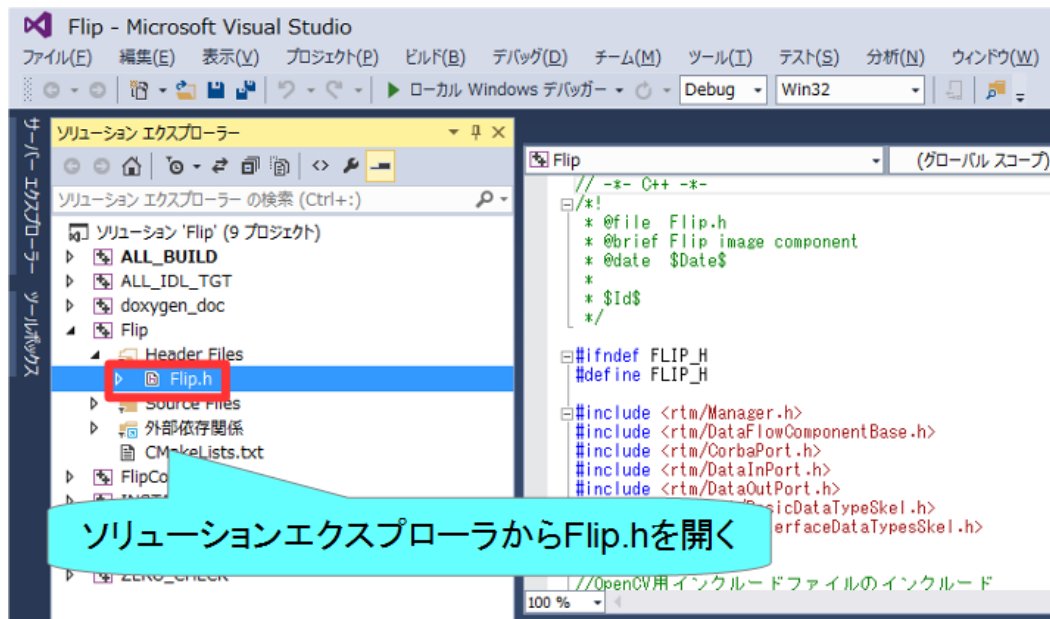
- Windows
 - Visual Studioが起動
- Ubuntu
 - Code::Blocksが起動



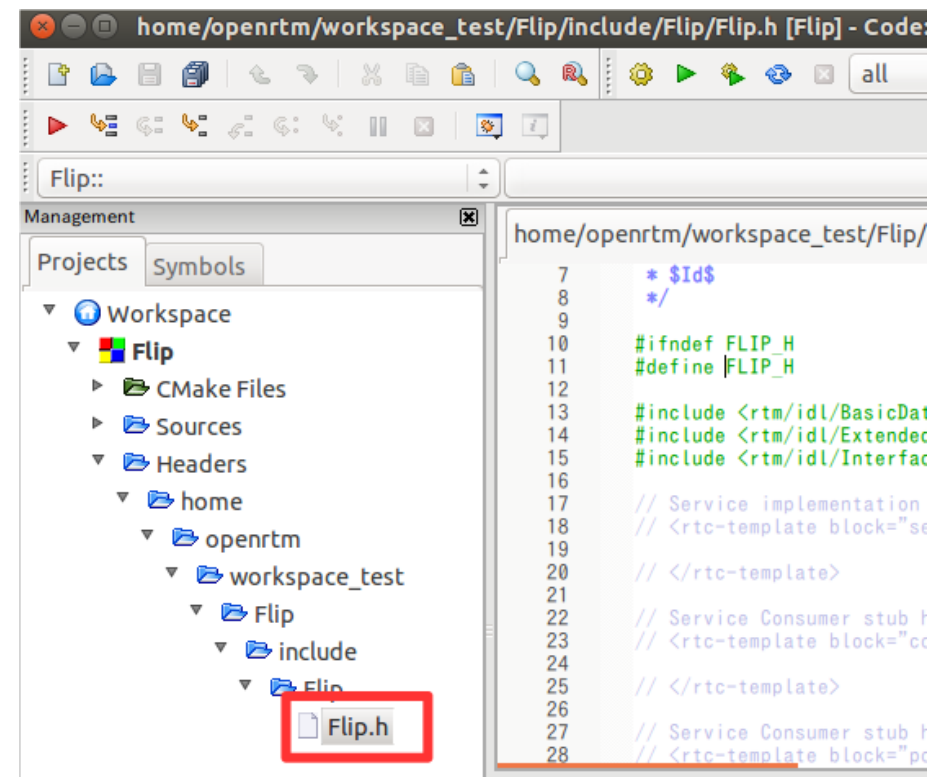
ソースコードの編集

- Flip.hの編集

Visual Studio



Code::Blocks



ProjectsからFlip.hを開く

ソースコードの編集

- Flip.hの編集

```

37 #include <rtm/CorbaPort.h>
38 #include <rtm/DataInPort.h>
39 #include <rtm/DataOutPort.h>
40
41 //OpenCV用インクルードファイルのインクルード
42 #include <opencv2/opencv.hpp>
43
44 using namespace RTC;
45

```

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

```

272 private:
273     // <rtc-template block="private_attribute">
274
275     // </rtc-template>
276
277     // <rtc-template block="private_operation">
278
279     // </rtc-template>
280     cv::Mat m_imageBuff;
281     cv::Mat m_flipImageBuff;
282
283 };

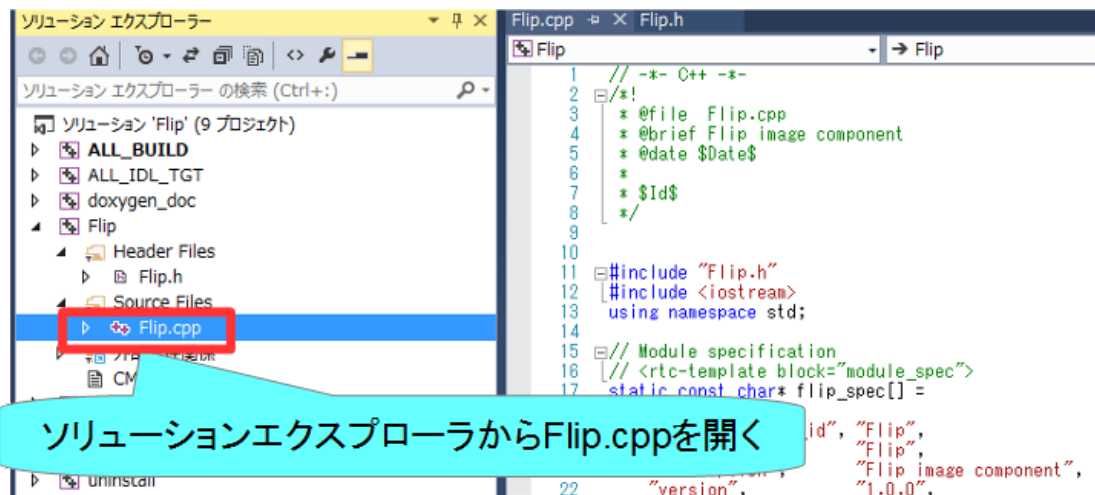
```

変数の宣言
cv::Mat m_imageBuff;
cv::Mat m_flipImageBuff;

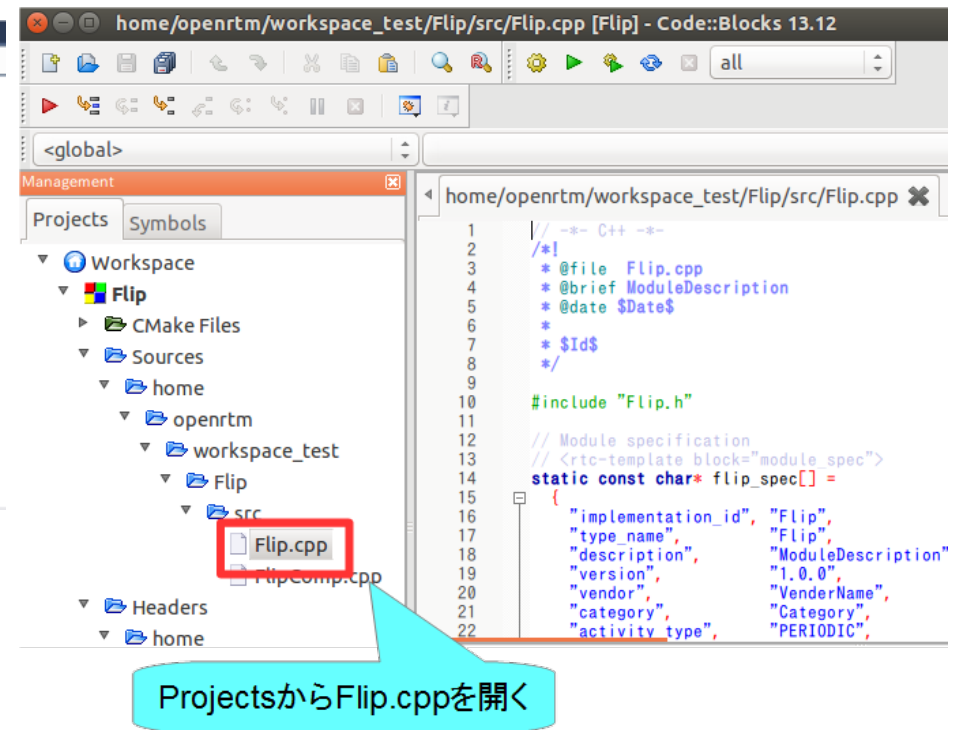
ソースコードの編集

- Flip.cppの編集
 - 配布したUSBメモリの「Flip」→「ソースコード」内に見本を同梱

Visual Studio



Code::Blocks



ソースコードの編集

- Flip.cppの編集

```

110  RTC::ReturnCode_t Flip::onActivated(RTC::UniqueId ec_id)
111  {
112
113      // OutPortの画面サイズの初期化
114      m_flippedImage.width = 0;
115      m_flippedImage.height = 0;
116
117      return RTC::RTC_OK;
118  }

```

onActivateに追加

```

121  RTC::ReturnCode_t Flip::onDeactivated(RTC::UniqueId ec_id)
122  {
123
124      if (!m_imageBuff.empty())
125      {
126          // 画像用メモリの解放
127          m_imageBuff.release();
128          m_flipImageBuff.release();
129      }
130
131      return RTC::RTC_OK;

```

onDeactivateに追加

ソースコードの編集

- Flip.cppの編集
 - 配布したUSBメモリの「Flip」→「ソースコード」内に見本を同梱

```

134  RTC::ReturnCode_t Flip::onExecute(RTC::UniqueId ec_id)
135  {
136      // 新しいデータのチェック
137      if (m_originalImageIn.isNew()) {
138          // InPortデータの読み込み
139          m_originalImageIn.read();
140
141          // 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
142          if (m_originalImage.width != m_flippedImage.width || m_originalImage.height != m_flippedImage.height)
143          {
144              m_flippedImage.width = m_originalImage.width;
145              m_flippedImage.height = m_originalImage.height;
146
147              m_imageBuff.create(cv::Size(m_originalImage.width, m_originalImage.height), CV_8UC3);
148              m_flipImageBuff.create(cv::Size(m_originalImage.width, m_originalImage.height), CV_8UC3);
149
150          }
151
152          // InPortの画像データをm_imageBuffにコピー
153          memcpy(m_imageBuff.data, (void *)&(m_originalImage.pixels[0]), m_originalImage.pixels.length());
154
155          // 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
156          cv::flip(m_imageBuff, m_flipImageBuff, m_flipMode);
157
158          // 画像データのサイズ取得
159          int len = m_flipImageBuff.channels() * m_flipImageBuff.cols * m_flipImageBuff.rows;
160          m_flippedImage.pixels.length(len);
161
162          // 反転した画像データをOutPortにコピー
163          memcpy((void *)&(m_flippedImage.pixels[0]), m_flipImageBuff.data, len);
164
165          // 反転した画像データをOutPortから出力する。
166          m_flippedImageOut.write();
167      }
168  }
169
170  return RTC::RTC_OK;
171  }

```

onExecute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

```
// 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)
    {
```

read関数でデータの読み込み

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

- データを書き込む手順

変数m_flippedImageにデータを格納

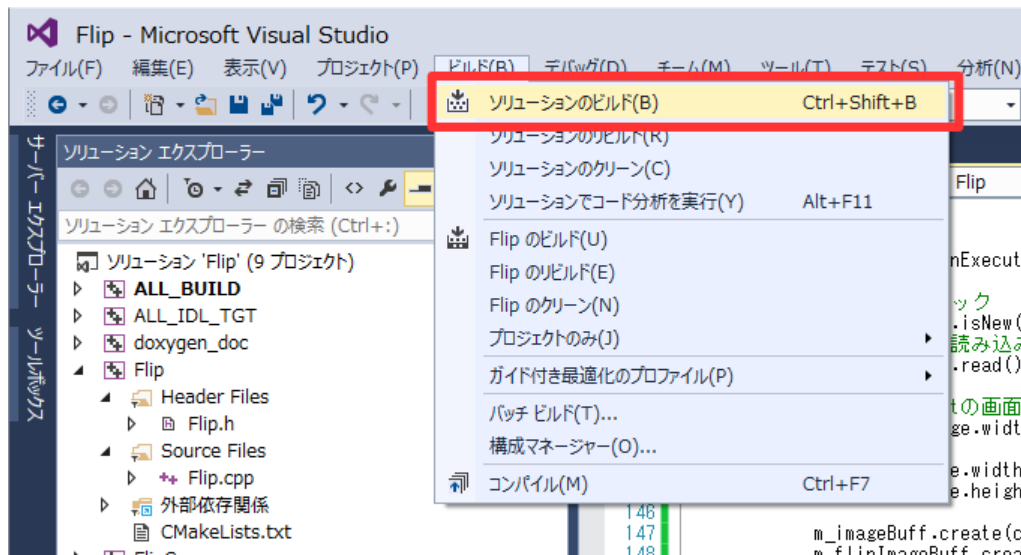
```
// 反転した画像データをOutPortにコピー
memcpy((void *)&(m_flippedImage.pixels[0]), m_flipImageBuff.data, len);

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

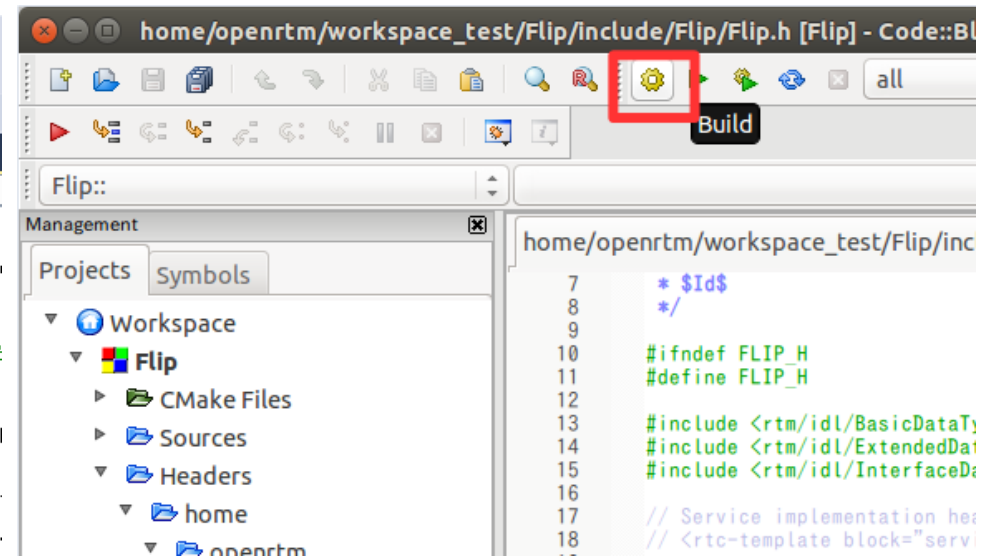
write関数でデータの書き込み

ソースコードのコンパイル

Visual Studio



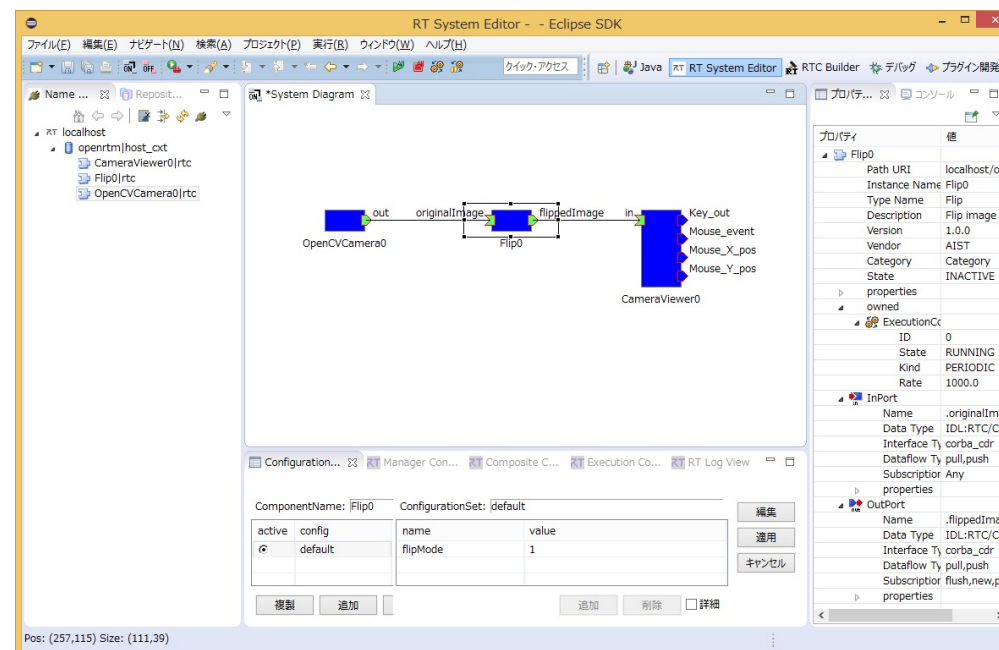
Code::Blocks



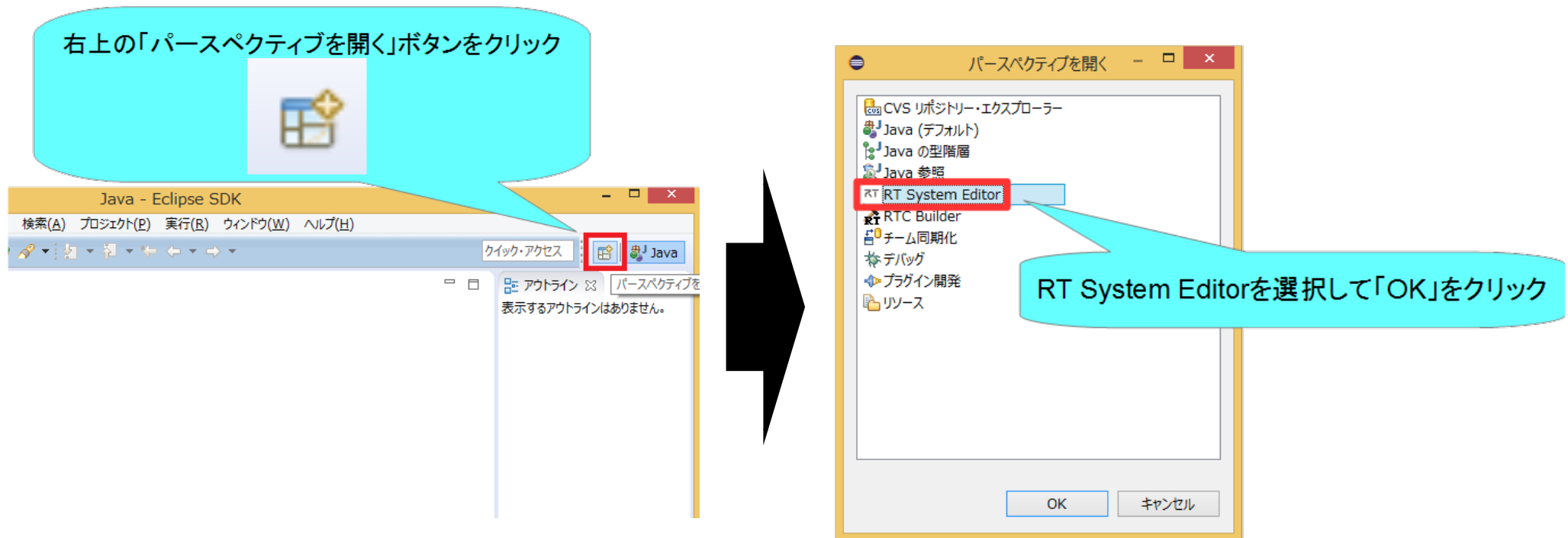
(3) システム構築とテスト

RT System Editor

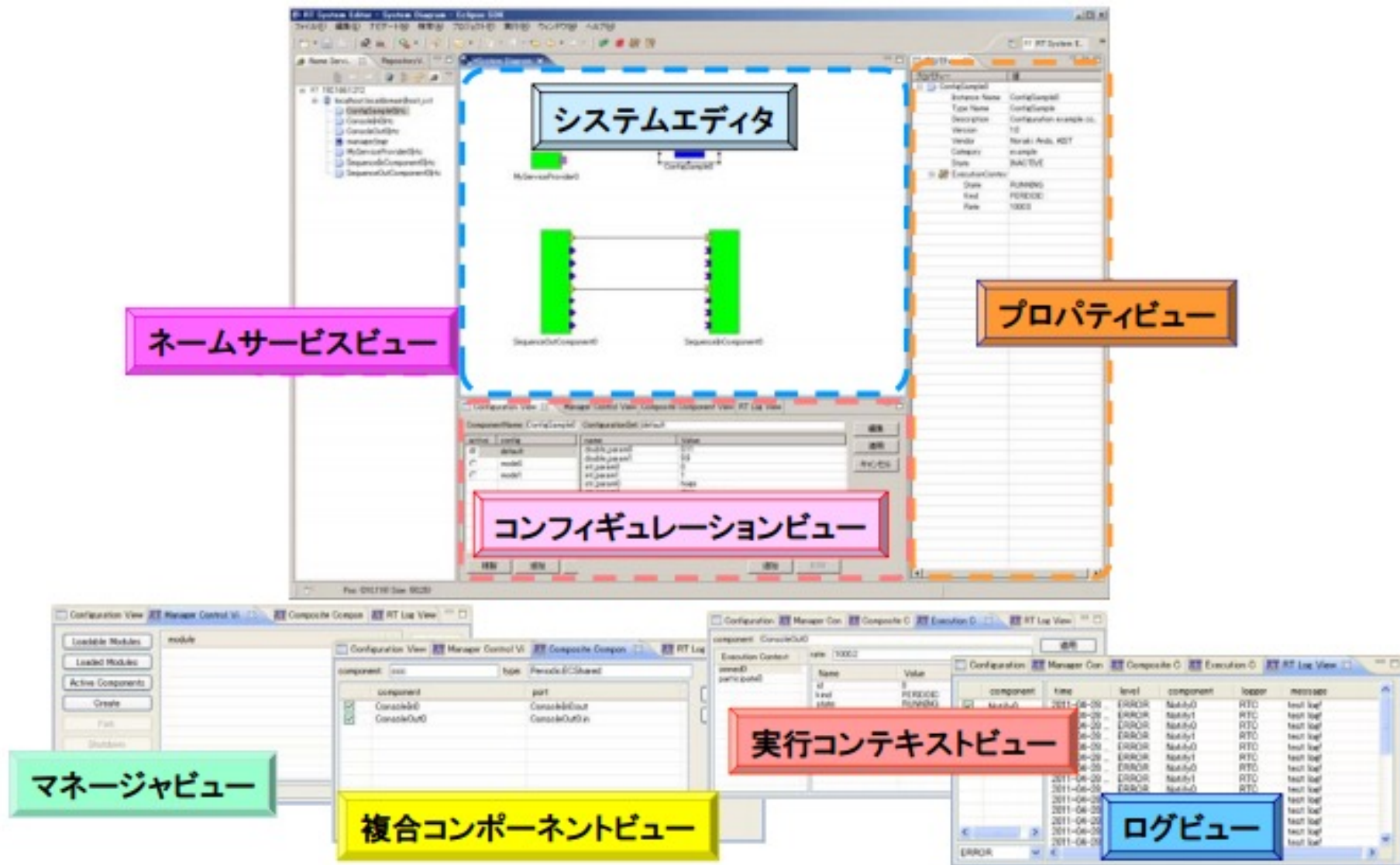
- RTCをGUIで操作するためのツール
 - データポート、サービスポートの接続
 - アクティブ化、非アクティブ化、リセット、終了
 - コンフィギュレーションパラメータの操作
 - 実行コンテキストの操作
 - 実行周期変更
 - 実行コンテキストの関連付け
 - 複合化
 - マネージャからRTCを起動
 - 作成したRTシステムの保存、復元



RT System Editorの起動

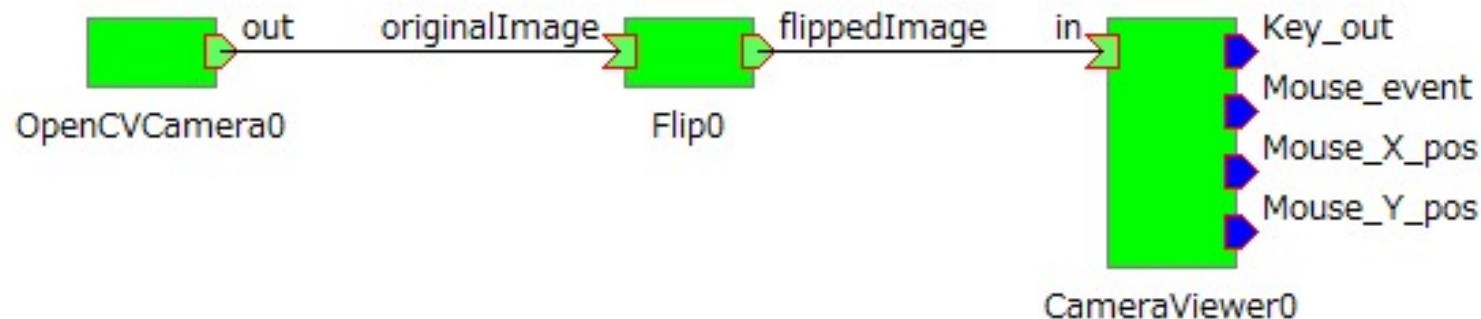


RT System Editorの画面構成



Flipコンポーネントの動作確認

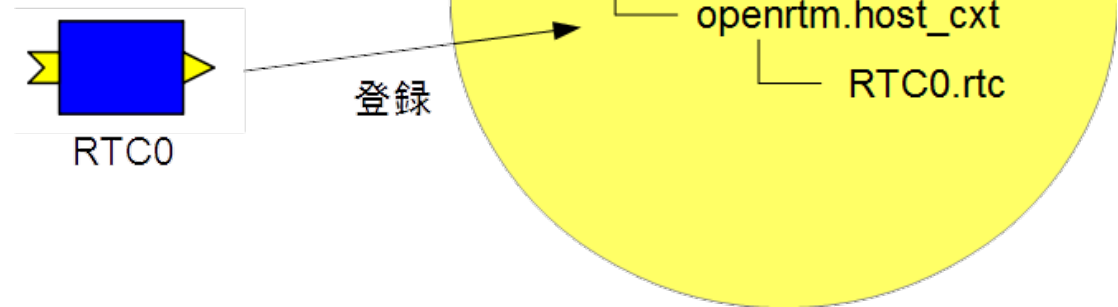
- WEBカメラで撮影した画像を反転させて表示するRTシステムを作成する
 - ネームサーバーを起動する
 - CameraViewerコンポーネント、OpenCVCameraコンポーネントを起動する
 - Windows
 - 「OpenRTM-aist 1.2.2」 → 「C++」 → 「Components」 → 「OpenCVExamples」
 - Ubuntu
 - \$ /usr/local/share/openrtm-1.1/components/c++/opencv-rtcs/CameraViewerComp
 - \$ /usr/local/share/openrtm-1.1/components/c++/opencv-rtcs/OpenCVCameraComp
 - 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
 - CameraViewerコンポーネント、OpenCVCameraコンポーネント、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

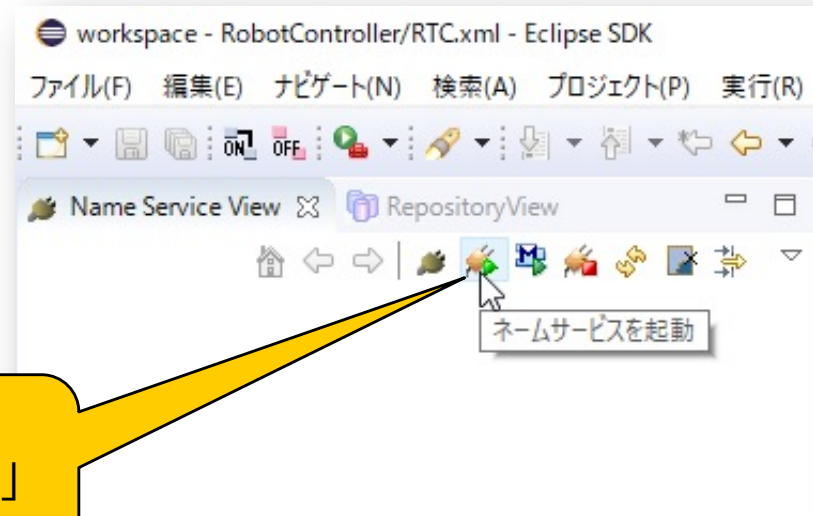
- RTCを一意の名前で登録する
 - RTシステムエディタ等のツールはネームサーバーから名前でRTCの参照を取得する



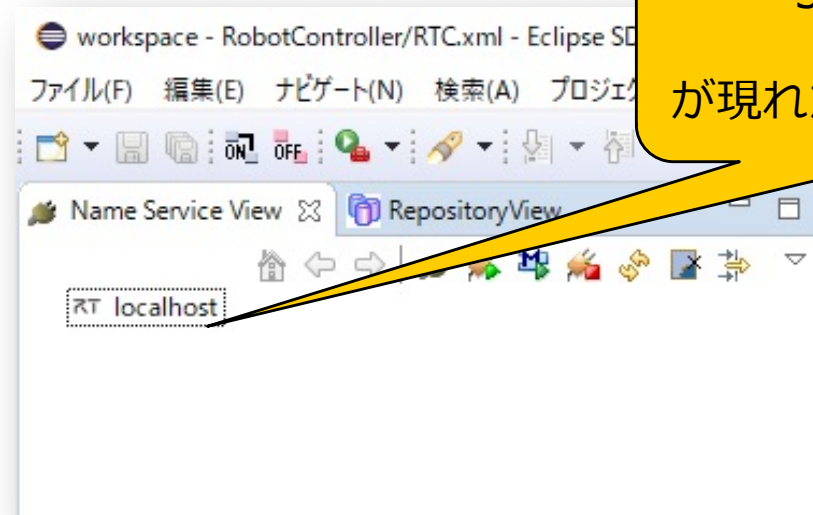
- 起動方法

- RTSystemEditorのName Service Viewから起動
 - 通常はこの方法
- コマンドを直接起動（スタートメニュー）
 - Windows: 「スタート」→あぷり一覧から「OpenRTM-aist 1.X.Y」→「Start Naming Service」を選択
 - または: 「ここに入力して検索」からStart Naming程度まで入力すると「Start Naming Service」が表示される
- Ubuntu
 - \$ rtm-naming

RTSystemEditorからのネームサーバー起動・接続



このアイコン
「ネームサービスを起動」
をクリック

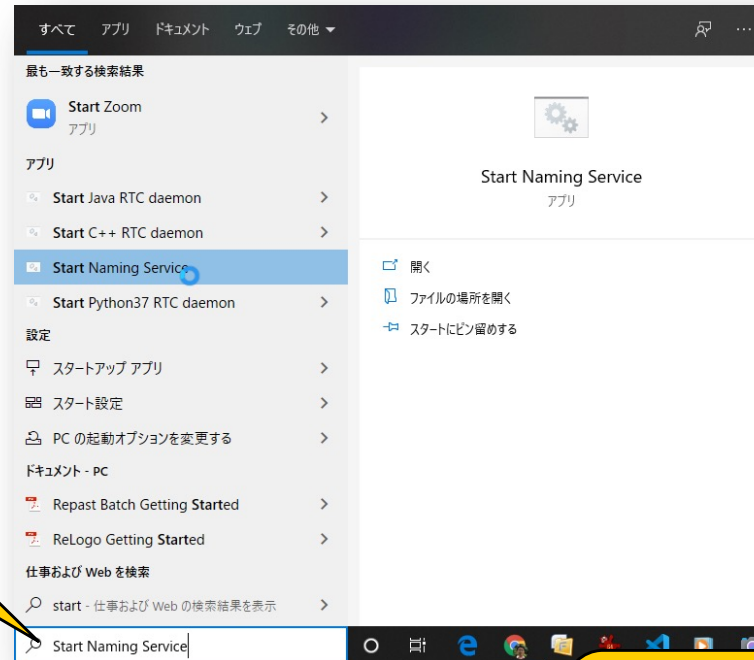


5秒ほど待つて
“localhost”
が現れたら起動・接続成功

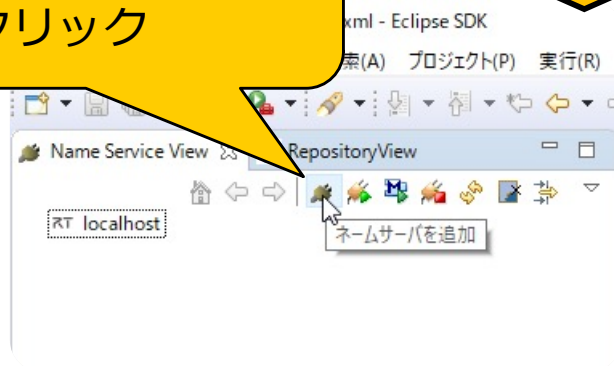
(参考) ネームサーバー起動・接続を個別に行う

- Windows 10

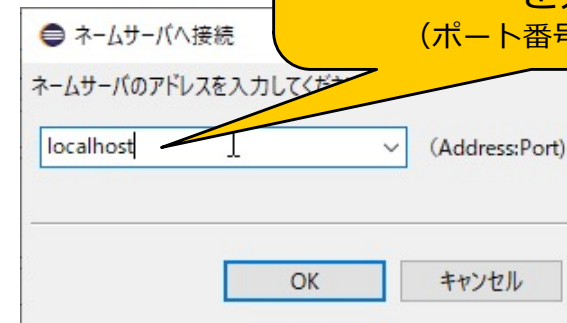
左下の検索窓に"start"と入力すると"Start Naming Service"が候補として出てくる



このアイコン
「ネームサーバを追加」
をクリック

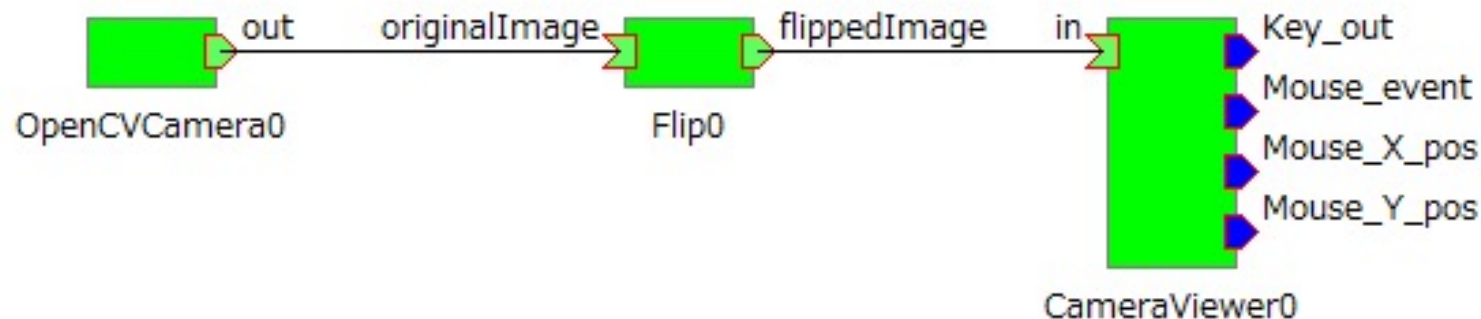


現れるダイアログに
「ホスト名 : ポート番号」
を入力
(ポート番号は省略可能)

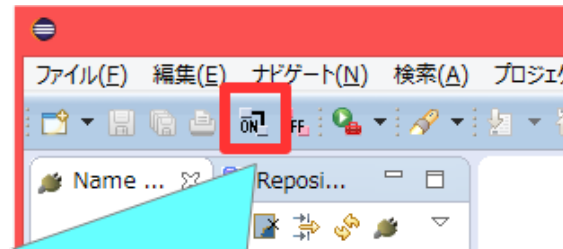


Flipコンポーネントの動作確認

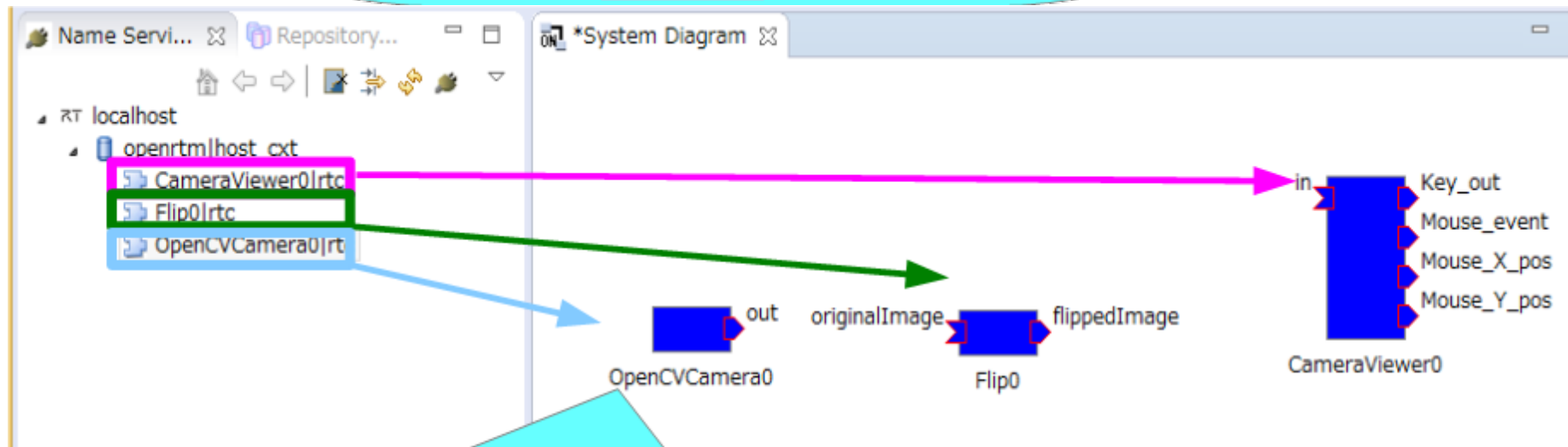
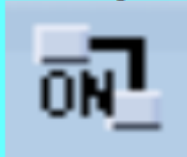
- WEBカメラで撮影した画像を反転させて表示するRTシステムを作成する
 - ネームサーバーを起動する
 - CameraViewerコンポーネント、OpenCVCameraコンポーネントを起動する
 - Windows
 - 「OpenRTM-aist 1.2.2」 → 「C++」 → 「Components」 → 「OpenCVExamples」
 - Ubuntu
 - \$ /usr/share/openrtm-1.2/components/c++/opencv-rtcs/CameraViewerComp
 - \$ /usr/share/openrtm-1.2/components/c++/opencv-rtcs/OpenCVCameraComp
 - 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
 - CameraViewerコンポーネント、OpenCVCameraコンポーネント、Flipコンポーネントを接続して「All Activate」を行う



データポートの接続



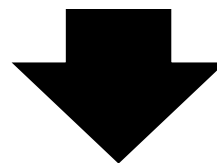
このボタンを押すことでSystem Diagramを表示する



左側のネームサービスビューから
CameraViewer0.rtc、OpenCVCamera0.rtc、Flip0.rtcを
System Diagramに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「originalImage」にドラッグアンドドロップ



Connector Profile

ConnectorProfileを入力してください。

Name :

Data Type :

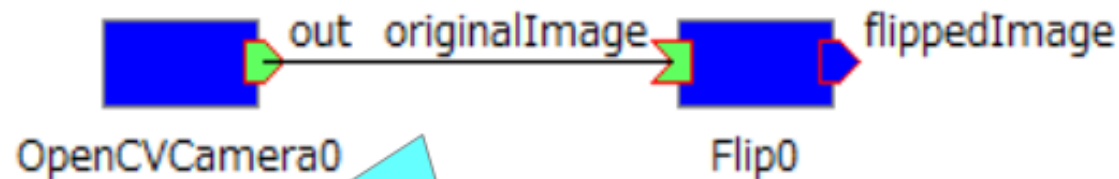
Interface Type :

Dataflow Type :

Skip Count :

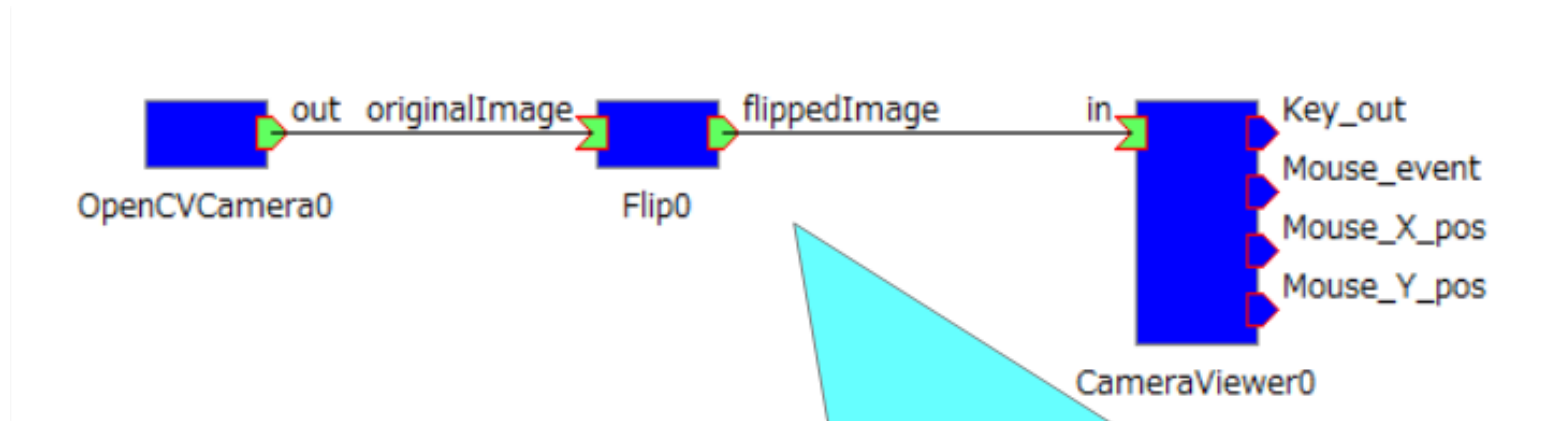
☐ 詳細...

そのままの設定でOKをクリック



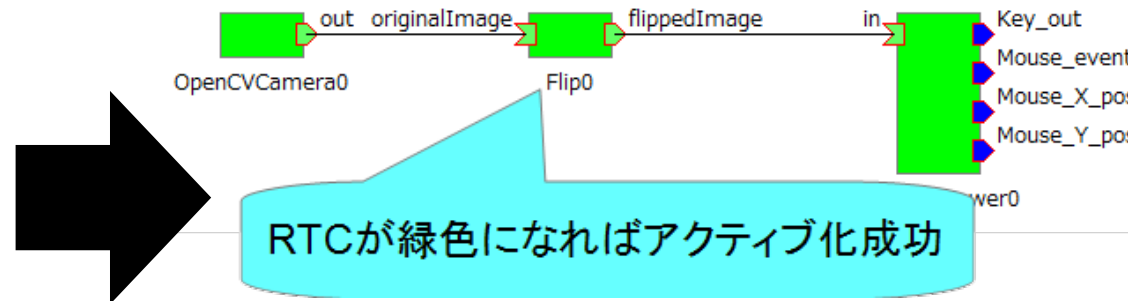
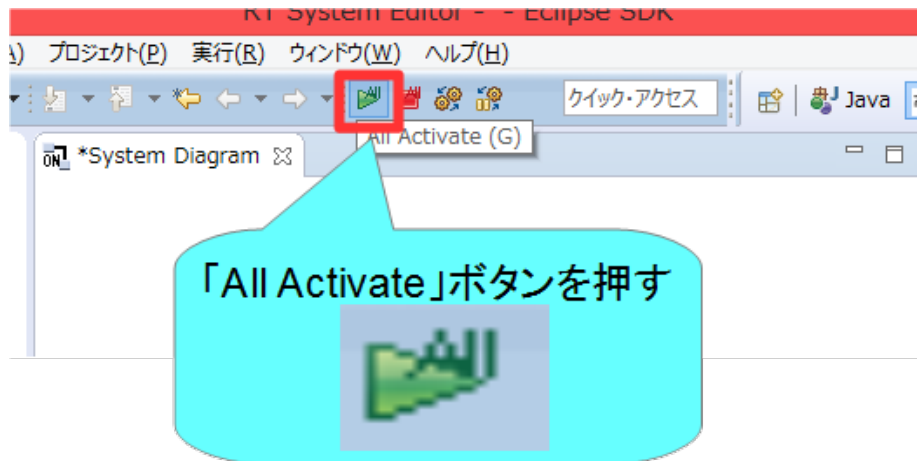
1. ポートの間に線が表示される
2. InPort、OutPortが緑色で表示される

データポートの接続



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

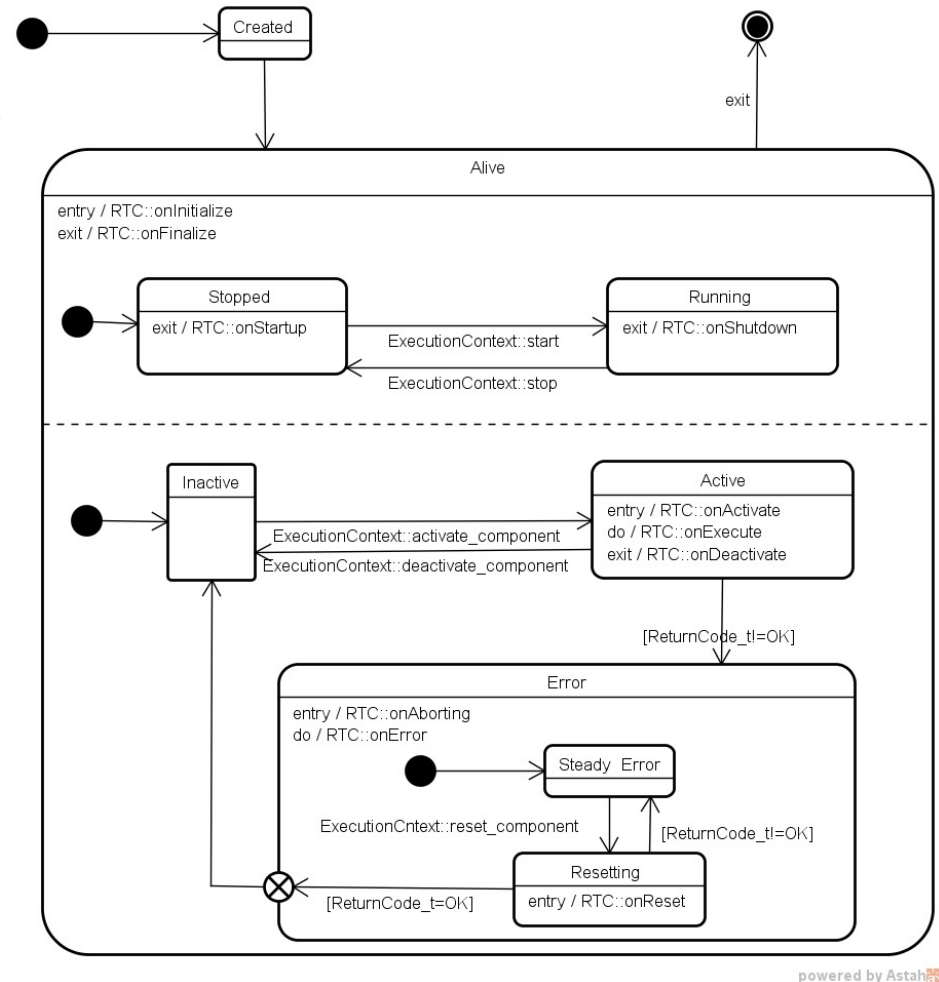
アクティブ化



- WEBカメラで撮影した画像が反転して表示されるかを確認してください
 - 表示されない場合
 - カメラがPCに接続されていない
 - データポートを接続していない
 - RTCがアクティブになっていない

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



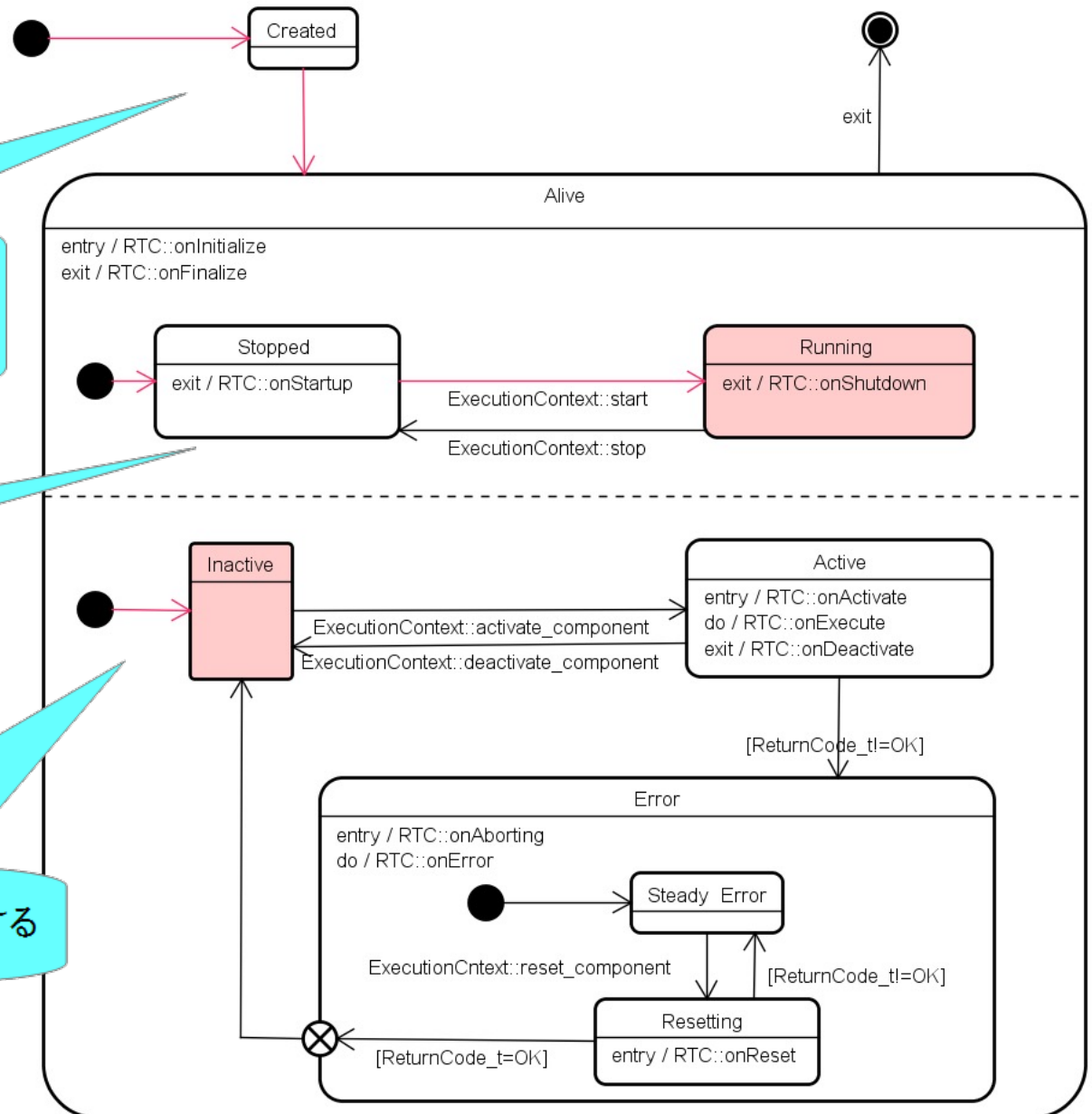
powered by Astah

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

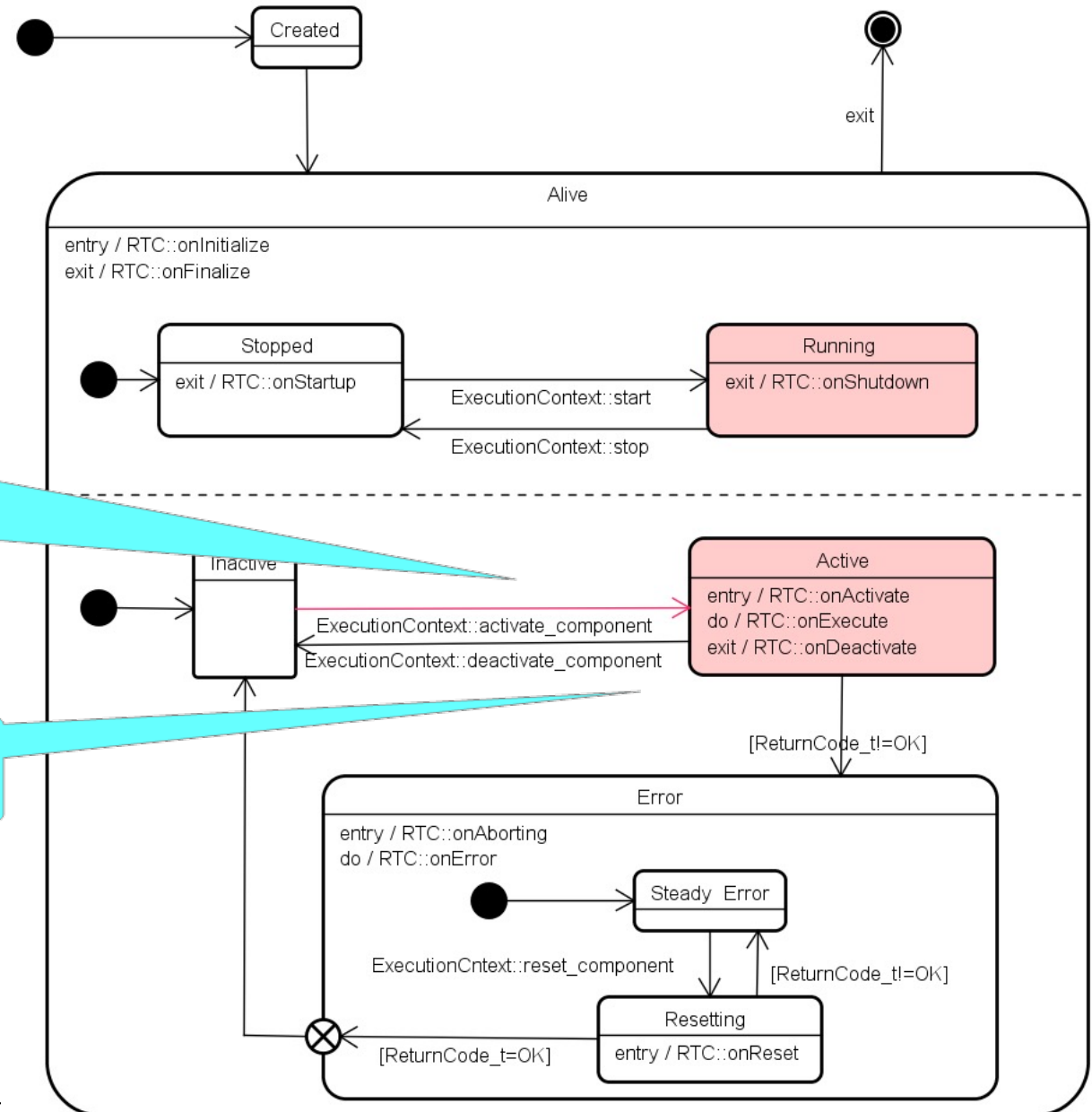
Created状態の次にInactive状態に遷移する



RTコンポーネントの状態遷移(アクティブ化)

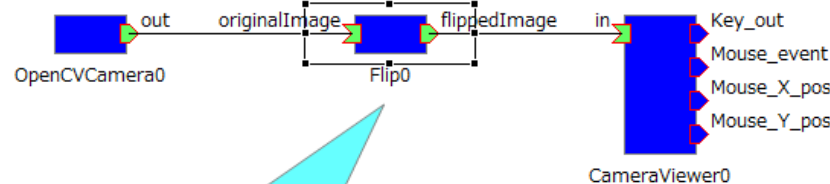
RTシステムエディタの操作によりRTコンポーネントのアクティブ化を行うとactivate_componentメソッドが呼び出される。
activate_componentメソッドによりコンポーネントがActive状態に遷移する。
この時onActivatedコールバックが実行される

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。

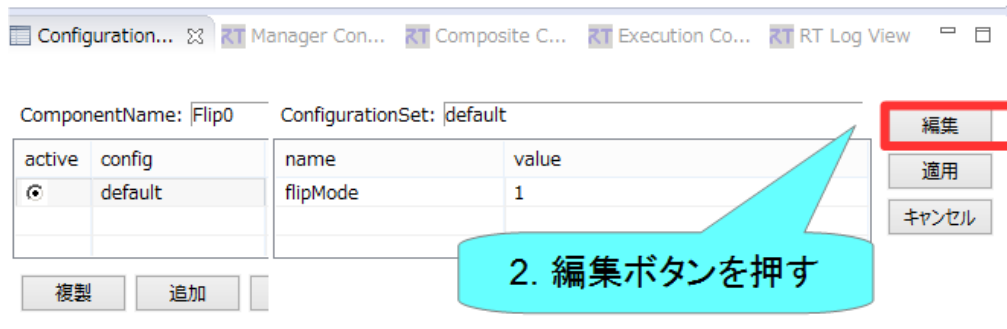


コンフィギュレーションパラメータの操作

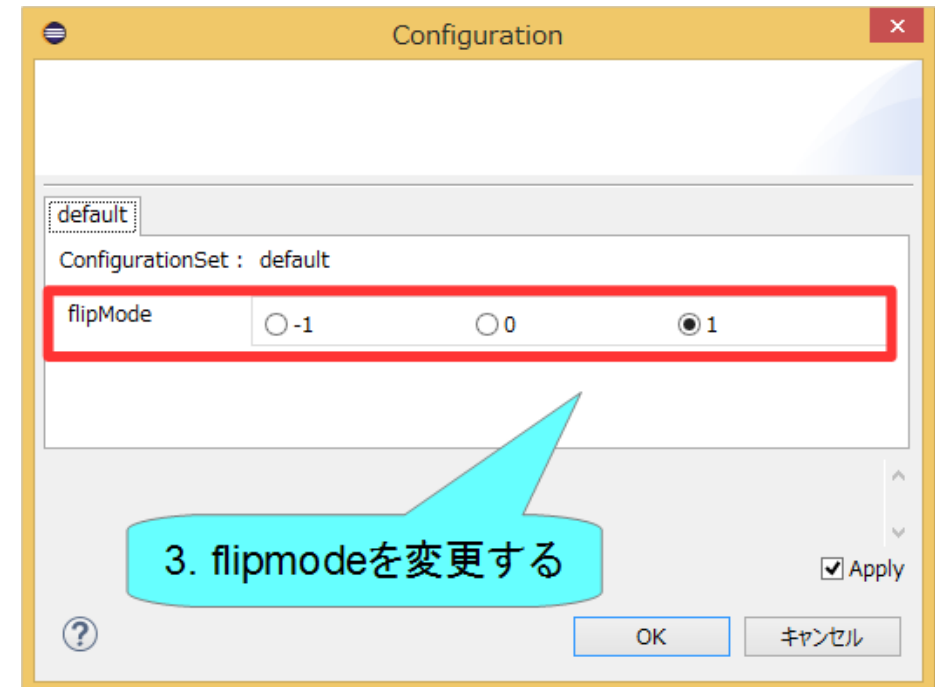
- コンフィギュレーションパラメータをRTシステムエディタから操作する
 - 反転する方向を設定



1. Flip0を選択する

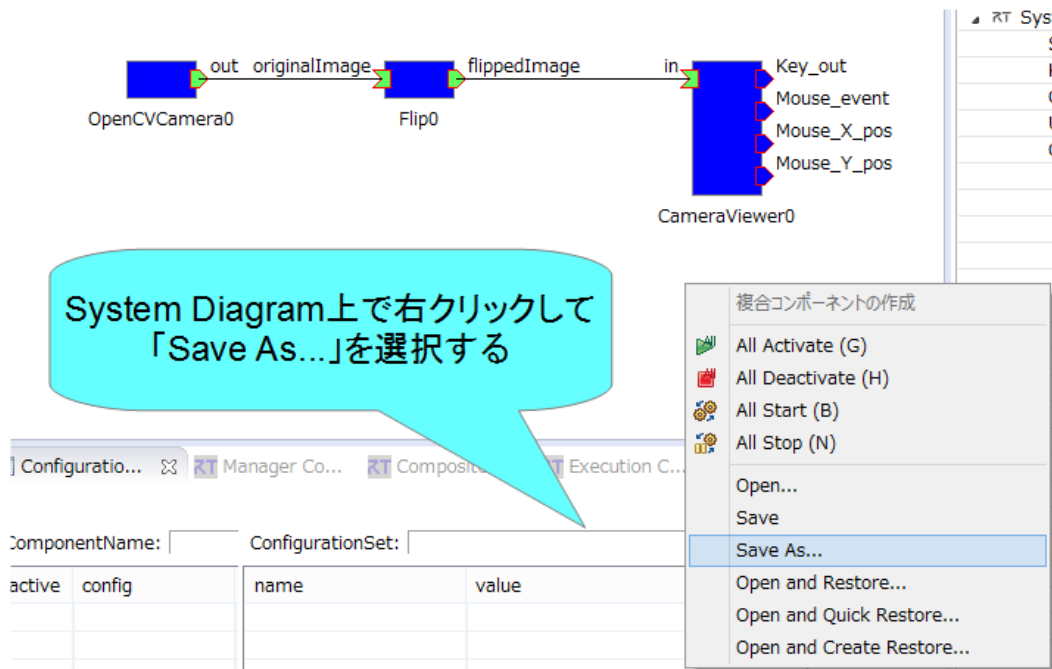


2. 編集ボタンを押す

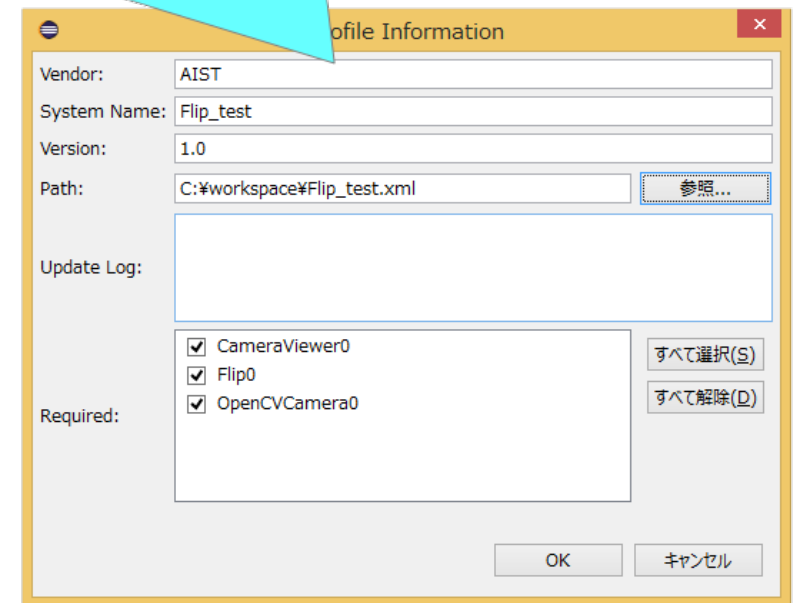


3. flipmodeを変更する

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

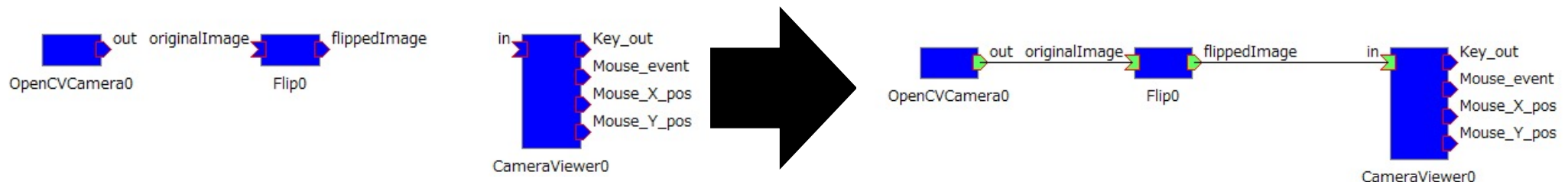
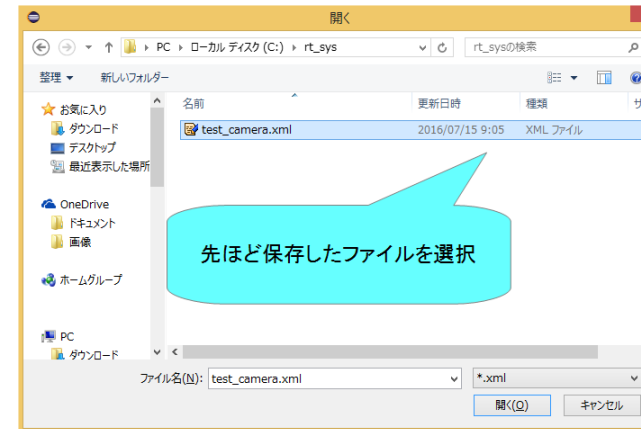
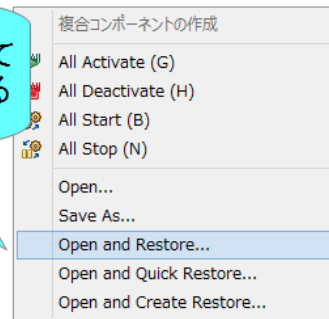


The screenshot shows the 'Profile Information' dialog box. It contains the following fields and options:

- Vendor: AIST
- System Name: Flip_test
- Version: 1.0
- Path: C:\workspace\Fip_test.xml (with a '参照...' button)
- Update Log: (empty text area)
- Required: (checkboxes for CameraViewer0, Flip0, and OpenCVCamera0, all checked)
- Buttons: 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



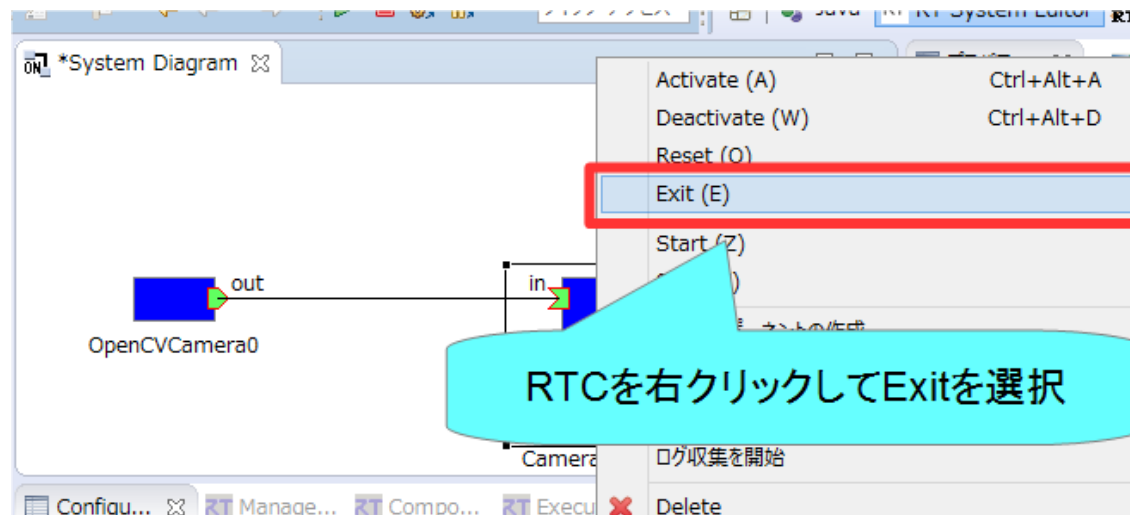
- 以下の内容を復元
 - ポート間の接続
 - コンフィギュレーション
 - 「Open and Create Restore」を選択した場合はマネージャからコンポーネント起動

非アクティブ化、終了

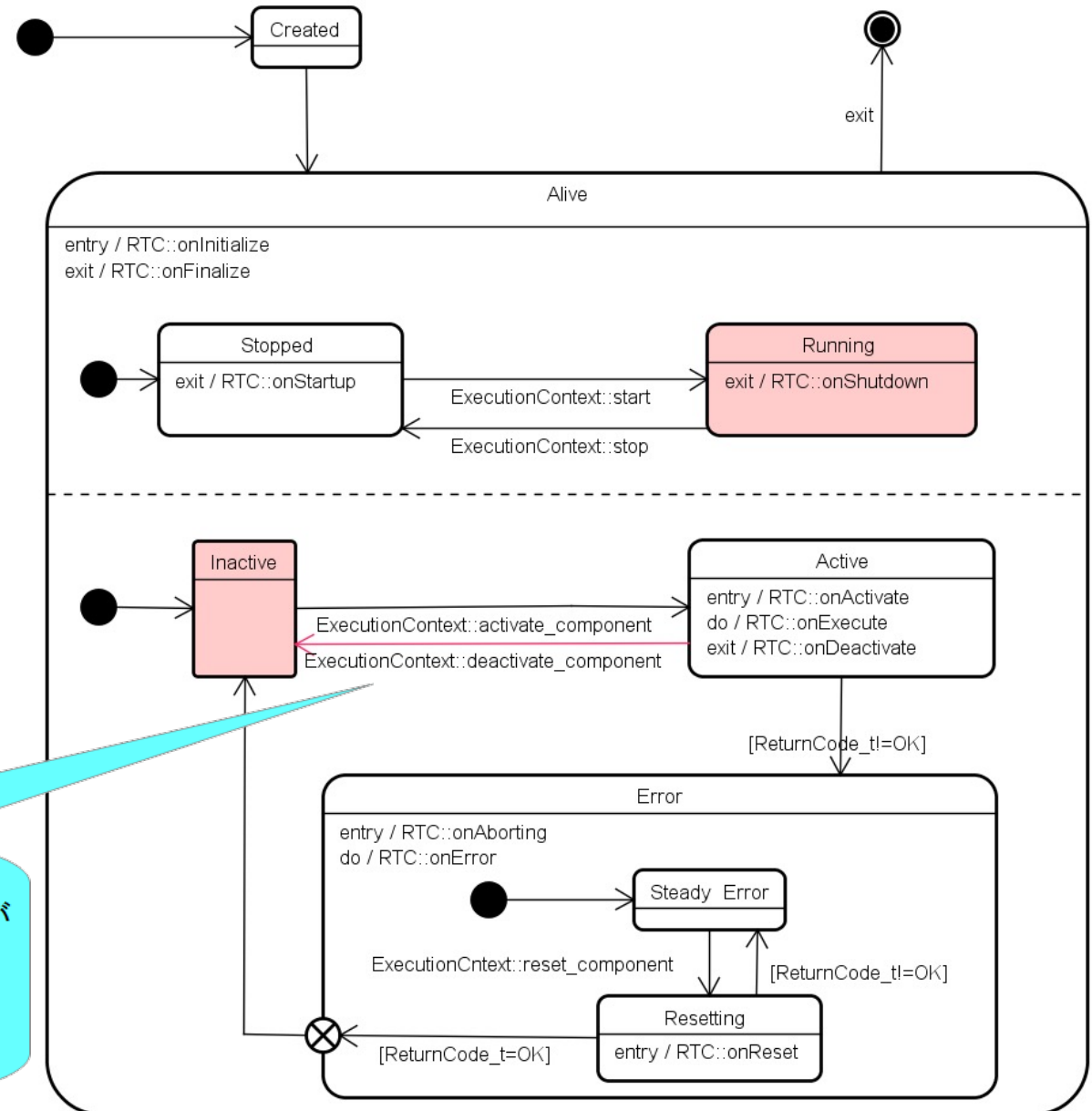
- 非アクティブ化



- 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

RTCBuilder 補足

リセット

- RTCがエラー状態に遷移した場合にエディタ上には赤く表示される。



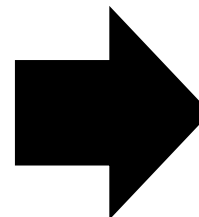
```
RTC::ReturnCode_t Test::onActivated(RTC::UniqueId ec_id)↓
{↓
    HANDLE hCom = INVALID_HANDLE_VALUE;↓
    hCom = CreateFile("COM5", GENERIC_READ | GENERIC_WRITE, 0, NULL, OPEN_EXISTING, 0, NULL);↓
    if (hCom == INVALID_HANDLE_VALUE)↓
    {↓
        return RTC::RTC_ERROR;↓
    }↓
}
```

例えばonActivated関数で初期化(この例ではCOMポートの初期化)に失敗した場合はRTC_ERRORを返すようにしておけば、初期化に失敗した場合にエラー状態に遷移する

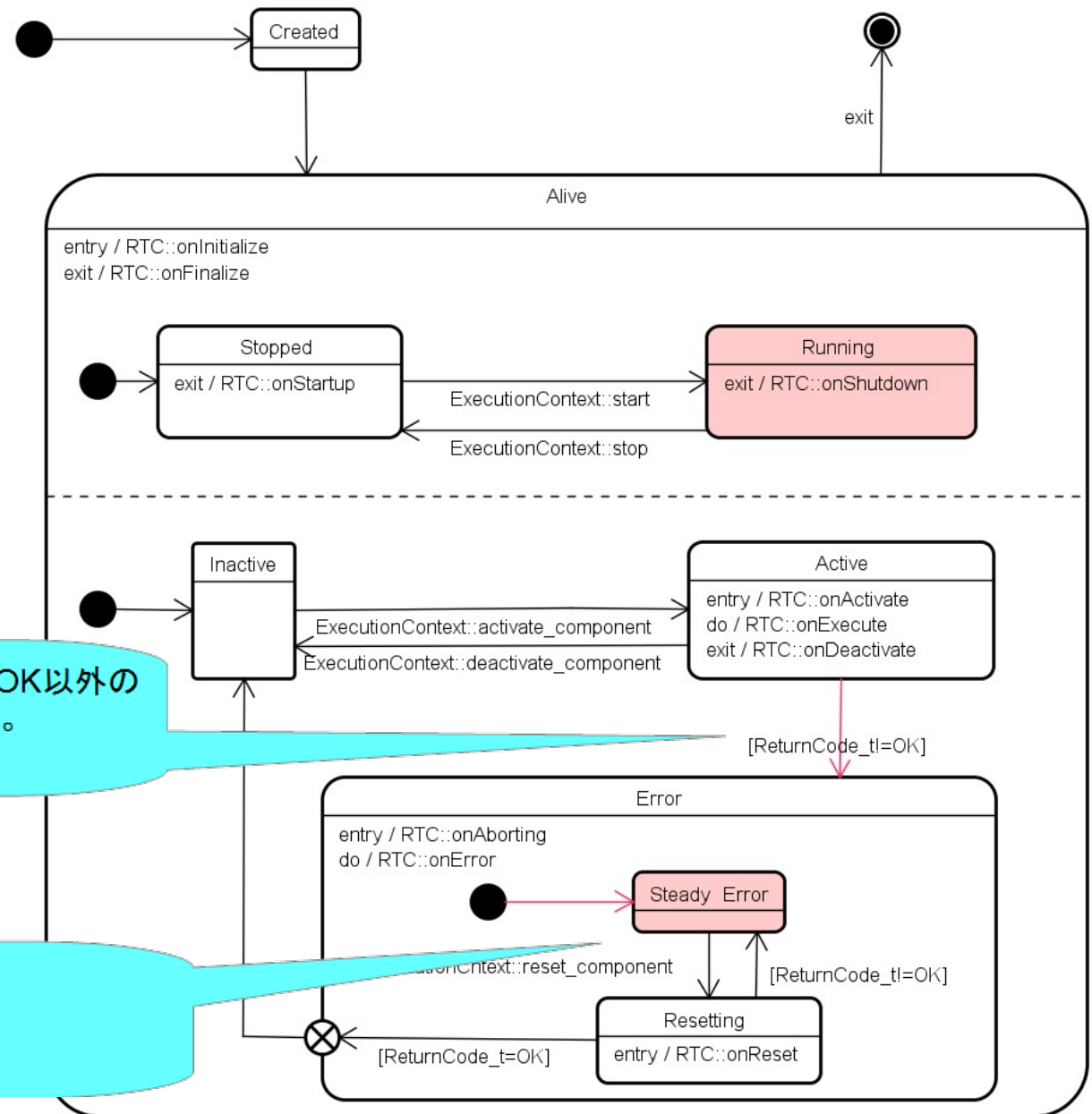
- 以下の操作で非アクティブ状態に戻す



RTCを右クリックしてResetを選択



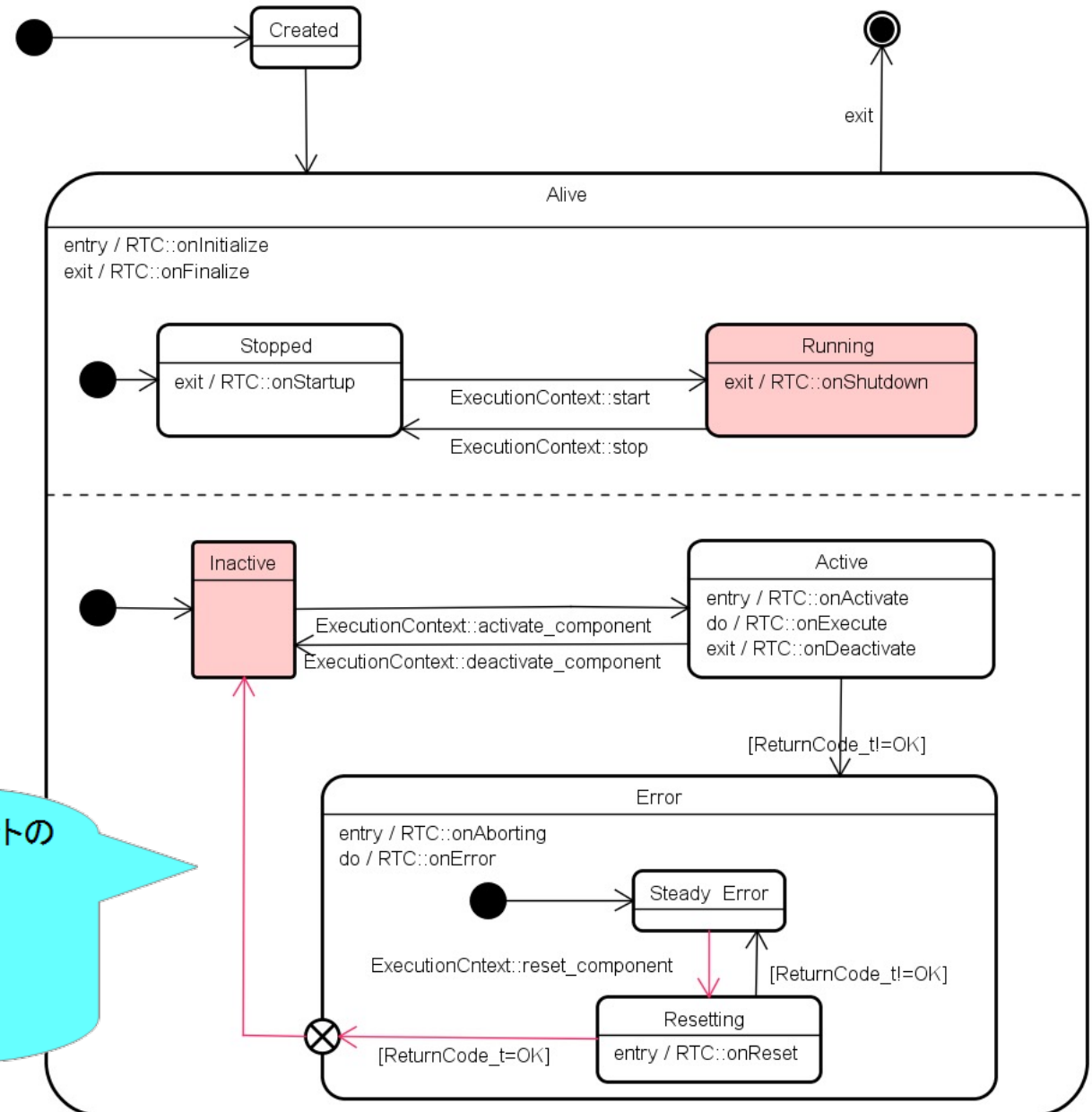
RTコンポーネントの状態遷移(エラー)



onActivated、onExecute、onDeactivatedがRTC OK以外のリターンコードを返した場合、エラー状態に遷移する。この時、onAborting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、onErrorコールバックが周期的に呼び出される。

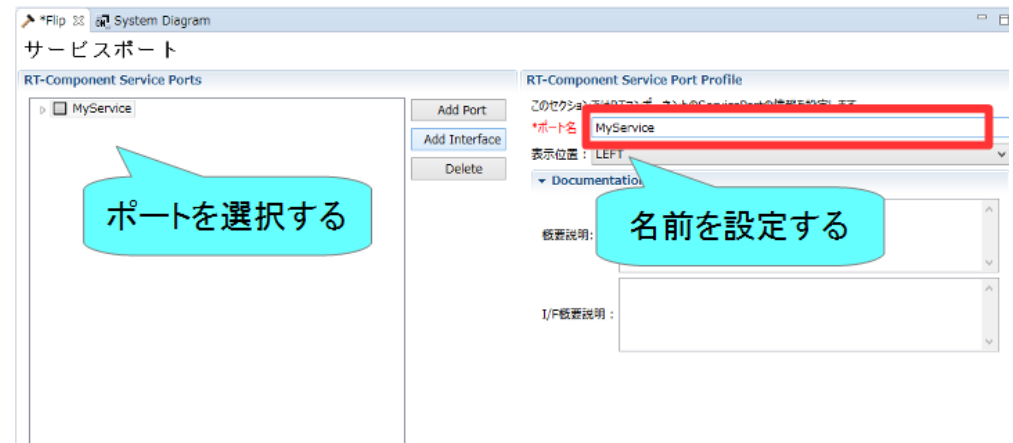
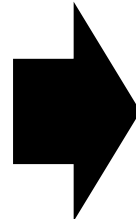
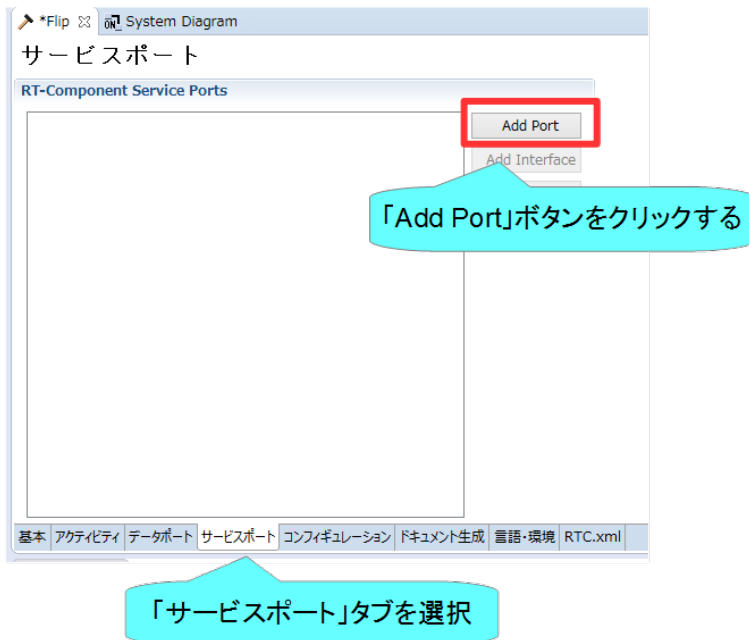
RTコンポーネントの状態遷移(リセット)



RTシステムエディタの操作によりRTコンポーネントのリセットを行うとreset_componentメソッドが呼び出される。
reset_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onResetコールバックが実行される

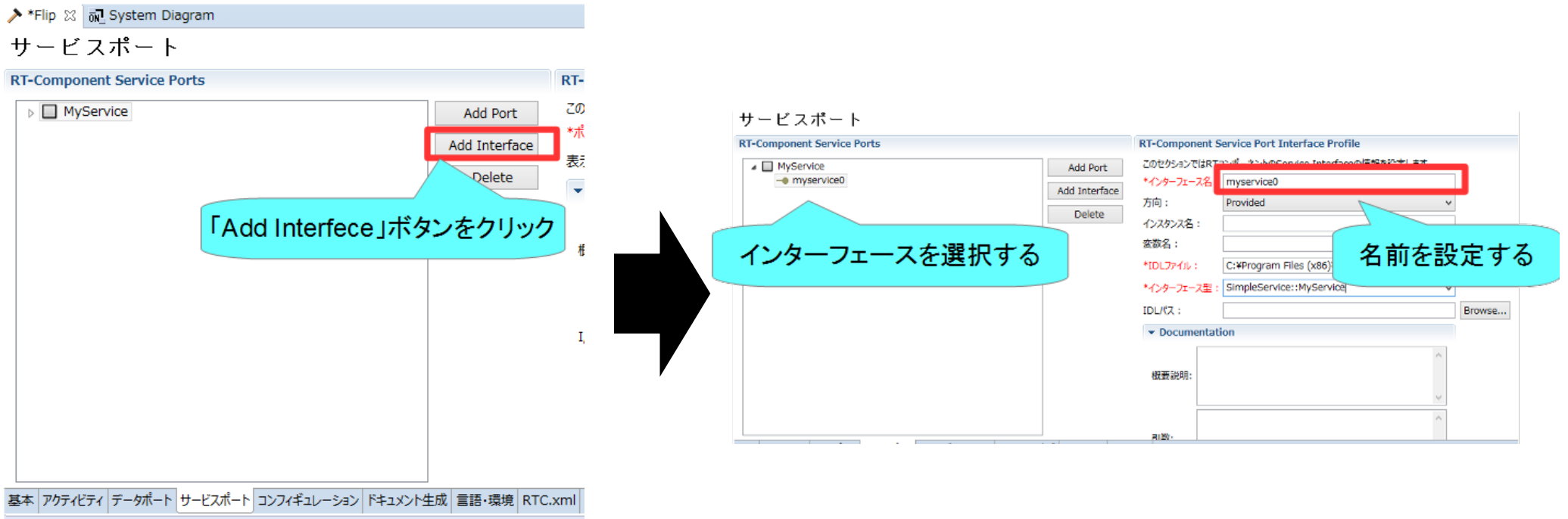
サービスポートの設定

- サービスポートの追加、インターフェースの追加、設定を行う



サービスポートの設定

- インターフェースを追加する



The image shows a two-step process for adding a service interface in the RT-Component software.

Step 1: Adding the Interface

In the **RT-Component Service Ports** window, the **Add Interface** button is highlighted with a red box. A callout bubble indicates: **「Add Interface」ボタンをクリック** (Click the 'Add Interface' button).

Step 2: Configuring the Interface Profile

The **RT-Component Service Port Interface Profile** dialog is shown. The ***インターフェース名** (Interface Name) field is highlighted with a red box and contains the text **myservice0**. A callout bubble indicates: **名前を設定する** (Set the name).

Other fields in the dialog include:

- 方向** (Direction): Provided
- インスタンス名** (Instance Name):
- 変数名** (Variable Name):
- *IDLファイル** (IDL File): C:\Program Files (x86)\
- *インターフェース型** (Interface Type): SimpleService::MyService
- IDLパス** (IDL Path):
- Documentation** section with **概要説明** (Summary) and **内容** (Content) fields.

A large black arrow points from the first step to the second step.

サービスポートの設定

- インターフェースの設定を行う

「Provided」・「Required」から選択
 Provided: サービスを提供する側
 Required: サービスを利用する側

ソケット



Required Interface
コンシューマ

機能を使う側

ロリポップ



Provided Interface
プロバイダ

機能を提供する側

このセクションではRTコンポーネントのService Interfaceの情報を設定

*インターフェース名:

方向: Provided ▼

インスタンス名:

変数名:

*IDLファイル: Browse...

*インターフェース型: SimpleService::MyService ▼

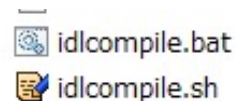
IDLパス: Browse...

「Browse...」をクリックしてIDLファイルを選択

インターフェース型を選択

IDLファイルが別のIDLファイルをインクルードしている場合にIDLパスを設定

- コード生成後、Pythonの場合は
idlcompile.bat(idlcompile.sh)を起動する



2016/07/03 18:07 Windows J
 2016/07/03 18:07 SH ファイル

サービスポートの設定

- IDLファイルについて
 - プログラミング言語に非依存のインターフェース定義言語

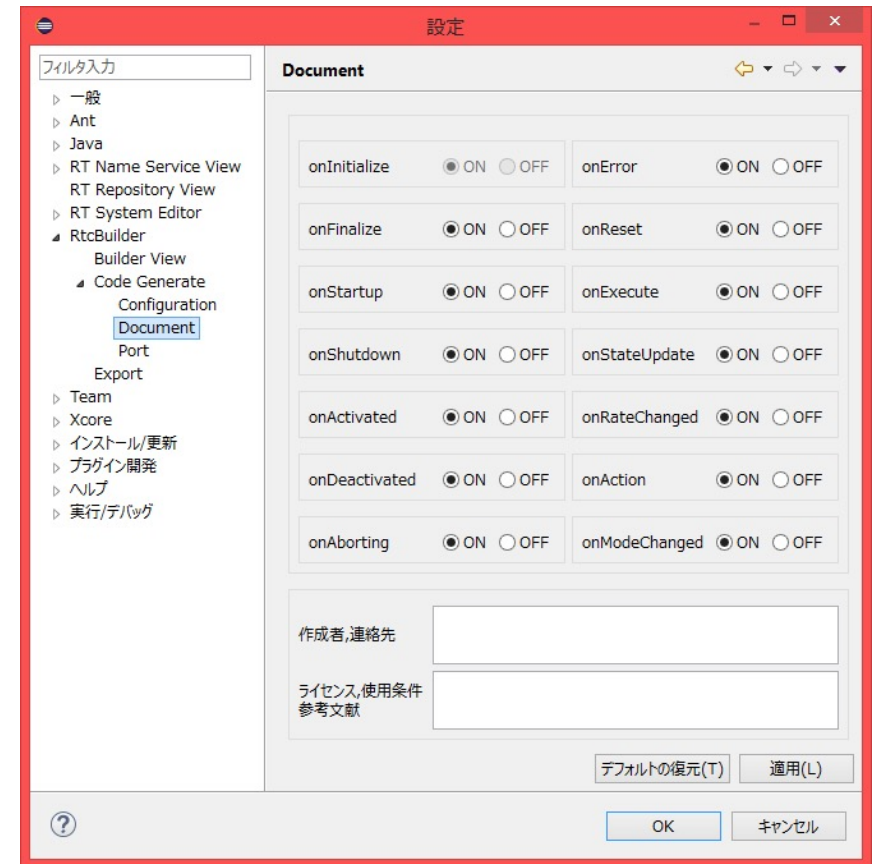
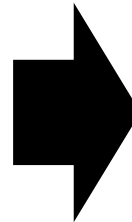
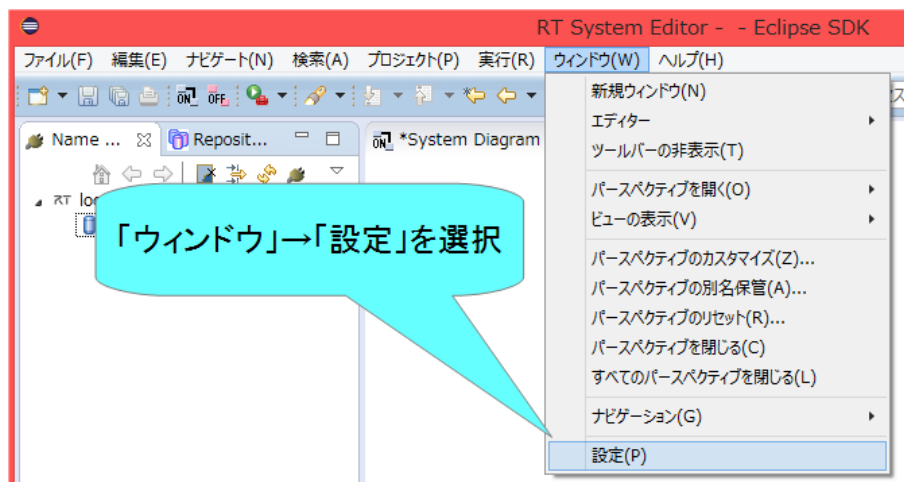
```

1 | module SimpleService {↓
2 |     typedef sequence<string> EchoList;↓
3 |     typedef sequence<float> ValueList;↓
4 |     interface MyService↓
5 |     {↓
6 |         string echo(in string msg);↓
7 |         EchoList get_echo_history();↓
8 |         void set_value(in float value);↓
9 |         float get_value();↓
10 |         ValueList get_value_history();↓
11 |     };↓
12 | };↓

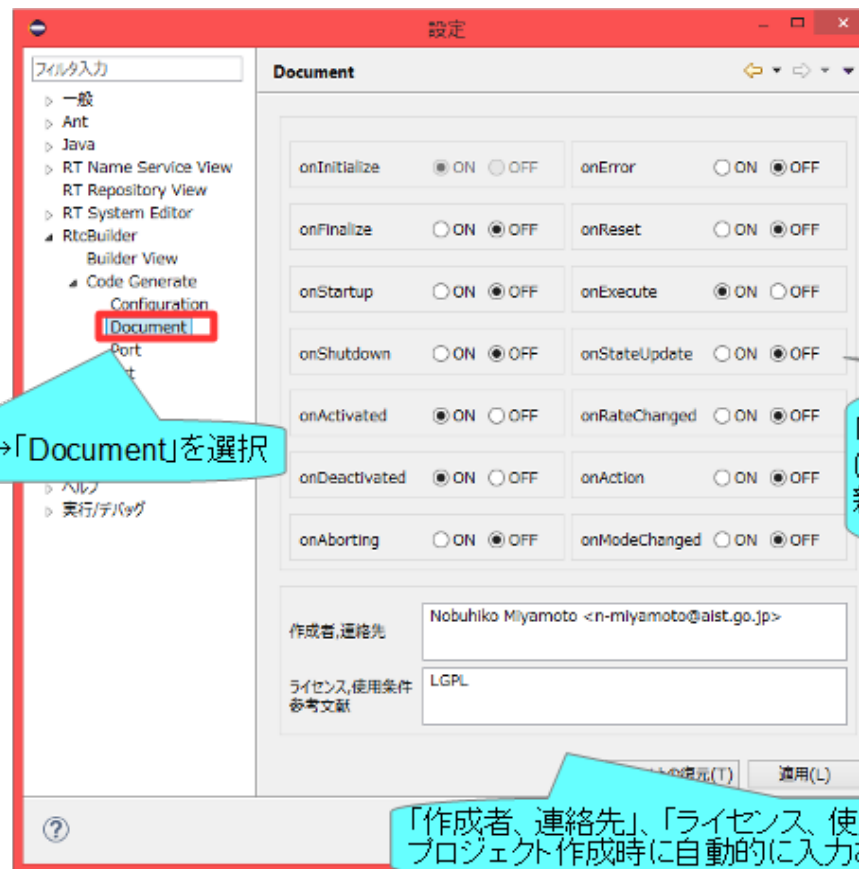
```

- コンシューマー側でプロバイダ側のecho、get_valueなどのオペレーションを呼び出す

RTCBuilderに関する設定



RTCBuilderに関する設定



「RtcBuilder」→「Code Generate」→「Document」を選択

「onActivated」、「onDeactivated」、「onExecute」はよく使うので「ON」にしておくとプロジェクトを新規作成したときに自動的にONになるので作業が減る。

「作成者、連絡先」、「ライセンス、使用条件、参考文献」を入力しておくとプロジェクト作成時に自動的に入力されるので便利。

独自のデータ型の利用

- 独自のデータ型でデータポートの通信を行う手順
 - IDLファイルを作成する
 - MyDataType.idlを任意のフォルダ(ここではC:¥UserDefType)作成

```

1 // @file MyDataType.idl ↓
2 #include "BasicDataType.idl" ↓
3   ↓
4 struct MyData ↓
5 { ↓
6   RTC::Time tm; ↓
7   short shortVariable; ↓
8   long longVariable; ↓
9   sequence<double> data; ↓
10 }; [EOF]

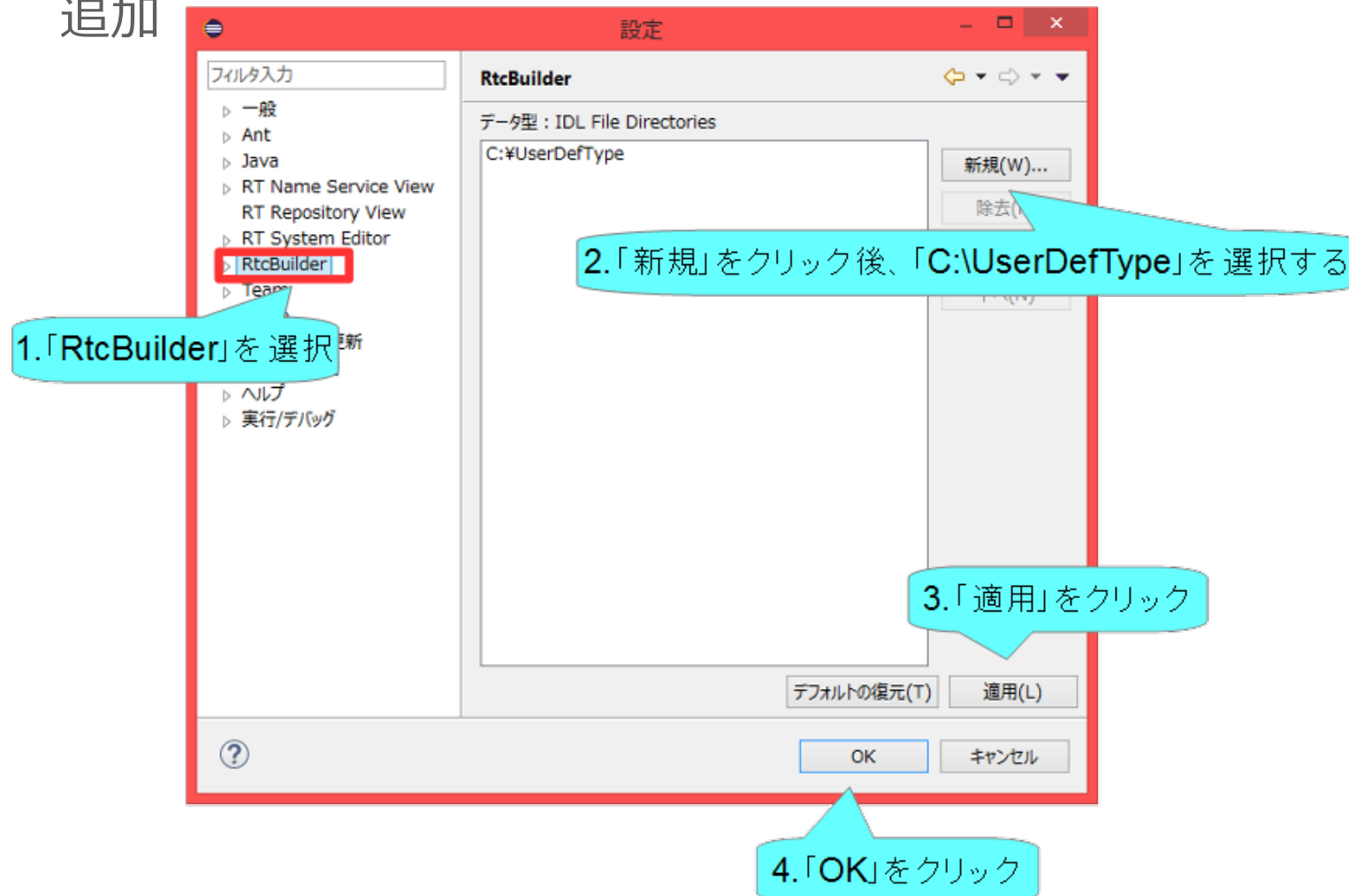
```

- 別のIDLファイルをインクルードしている場合は同じフォルダにコピーする

Windows 8... ▸ UserDefType ▾ ↻			UserDefTypeの検索 🔍
名前	更新日時	種類	
 BasicDataType.idl	2014/08/28 20:06	IDL ファイル	
 MyDataType.idl	2016/07/03 18:57	IDL ファイル	

独自のデータ型の利用

- 独自のデータ型でデータポートの通信を行う手順
 - RTC Builderの設定でIDLファイルの存在するディレクトリを追加



独自のデータ型の利用

- 独自のデータ型でデータポートの通信を行う手順

▼ DataPortプロフィール

このセクションではRTコンポーネントのDataPort(データポート)の情報を設定します。

*ポート名 (InPort)

in

< >

Add

Delete

*ポート名 (OutPort)

out

< >

Add

Delete

▼ 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 :

*データ型

変数名

表示位置

Documen

MyData

MyData

RTC::Acceleration2D

RTC::Acceleration3D

RTC::ActuatorCurrent

RTC::ActuatorGeometry

データ型一覧にMyDataが追加

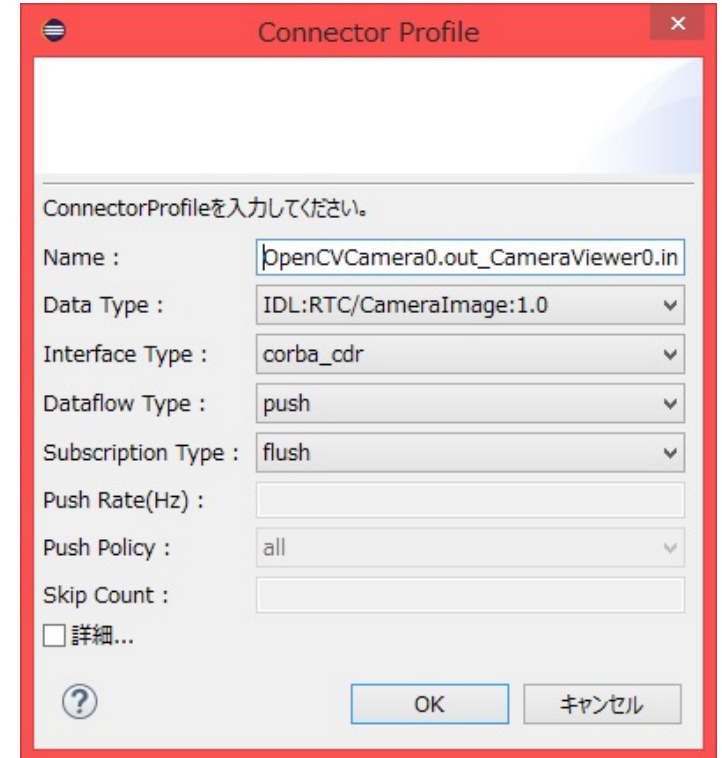
RTSystemEditor 補足

コネクタプロファイルの設定

項目	設定内容
Name	接続の名称
DataType	ポート間で送受信するデータの型. ex)TimedOctet, TimedShortなど
InterfaceType	データを送信方法. ex)corba_cdrなど
DataFlowType	データの送信手順. ex)push, pullなど
SubscriptionType	データ送信タイミング. 送信方法がPushの場合有効. New, Periodic, Flushから選択
Push Rate	データ送信周期(単位: Hz). SubscriptionTypeがPeriodicの場合のみ有効
Push Policy	データ送信ポリシー. SubscriptionTypeがNew, Periodicの場合のみ有効. all, fifo, skip, newから選択
Skip Count	送信データスキップ数. Push PolicyがSkipの場合のみ有効

コネクタプロファイルの設定

- InterfaceType
 - データの送信方法
 - corba_cdr (CORBAによる通信)
 - Direct (同一プロセスで起動したRTC間でデータを直接変数に渡す)
 - shared_memory (共有メモリによる通信)
- DataFlowType
 - データの送信手順
 - Push
 - OutPortがInPortにデータを送る
 - Pull
 - InPortがOutPortに問い合わせデータを受け取る
- SubscriptionType
 - データ送信タイミング(DataFlowTypeがPush型のみ有効)
 - flush(同期)
 - バッファを介さず即座に同期的に送信
 - new(非同期)
 - バッファ内に新規データが格納されたタイミングで送信
 - periodic(非同期)
 - 一定周期で定期的にデータを送信
- Push Policy(SubscriptionTypeがnew、periodicのみ有効)
 - データ送信ポリシー
 - all
 - バッファ内のデータを一括送信
 - fifo
 - バッファ内のデータをFIFOで1個ずつ送信
 - skip
 - バッファ内のデータを間引いて送信
 - new
 - バッファ内のデータの最新値を送信(古い値は捨てられる)



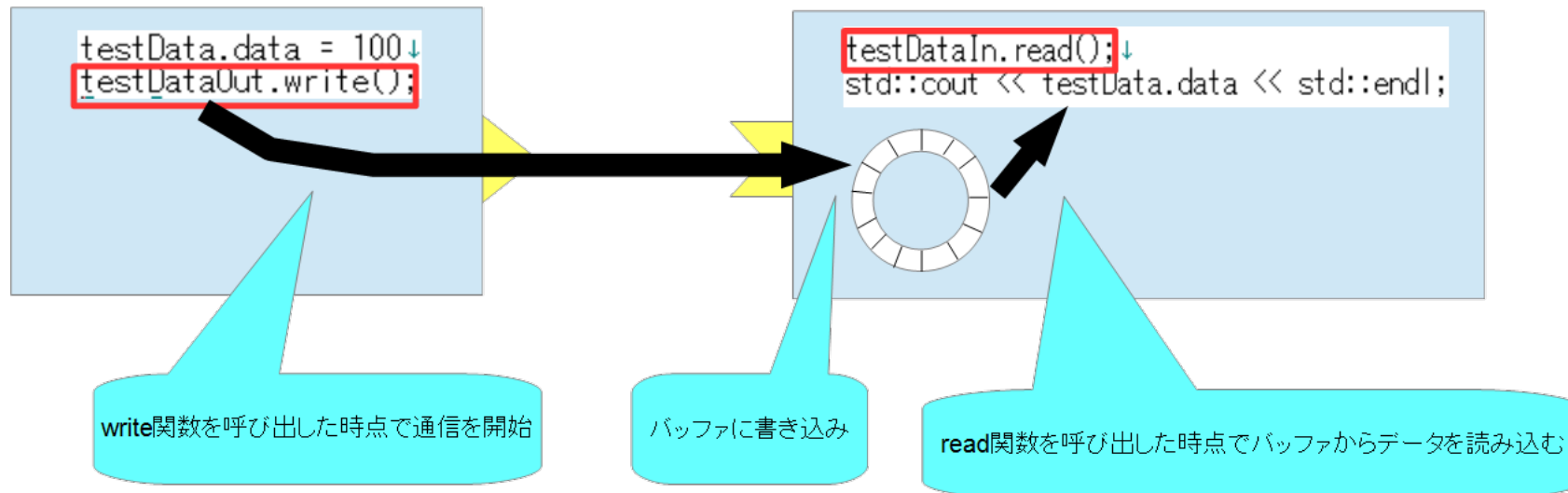
Connector Profile dialog box showing configuration for a connector profile. The title bar is red with a close button. The main area is white with a red border. The text 'ConnectorProfileを入力してください。' is at the top. The fields are:

- Name :
- Data Type :
- Interface Type :
- Dataflow Type :
- Subscription Type :
- Push Rate(Hz) :
- Push Policy :
- Skip Count :
- ☐ 詳細...

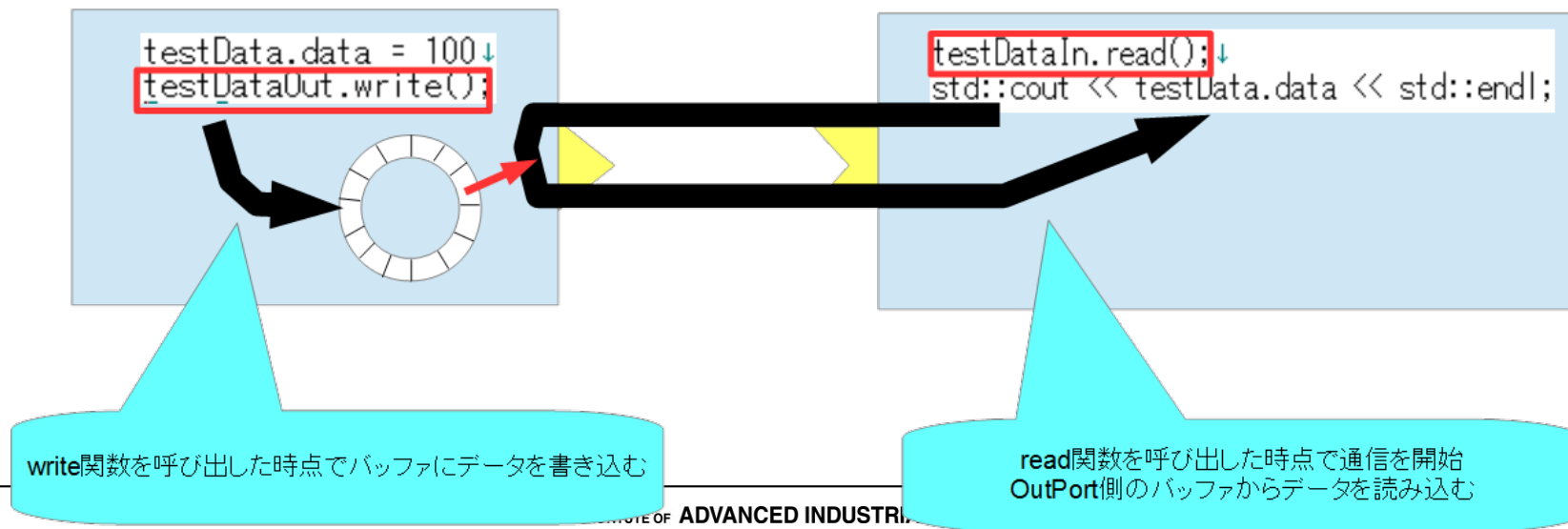
Buttons: OK, キャンセル

コネクタプロファイルの設定

- DataFlowType
 - Push

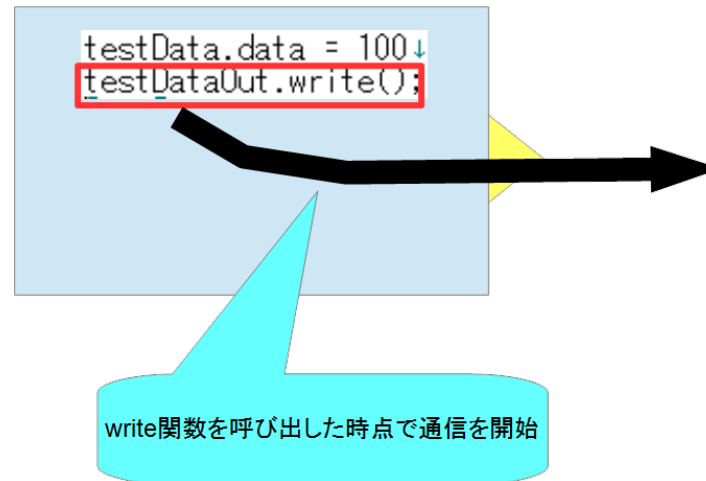


- Pull

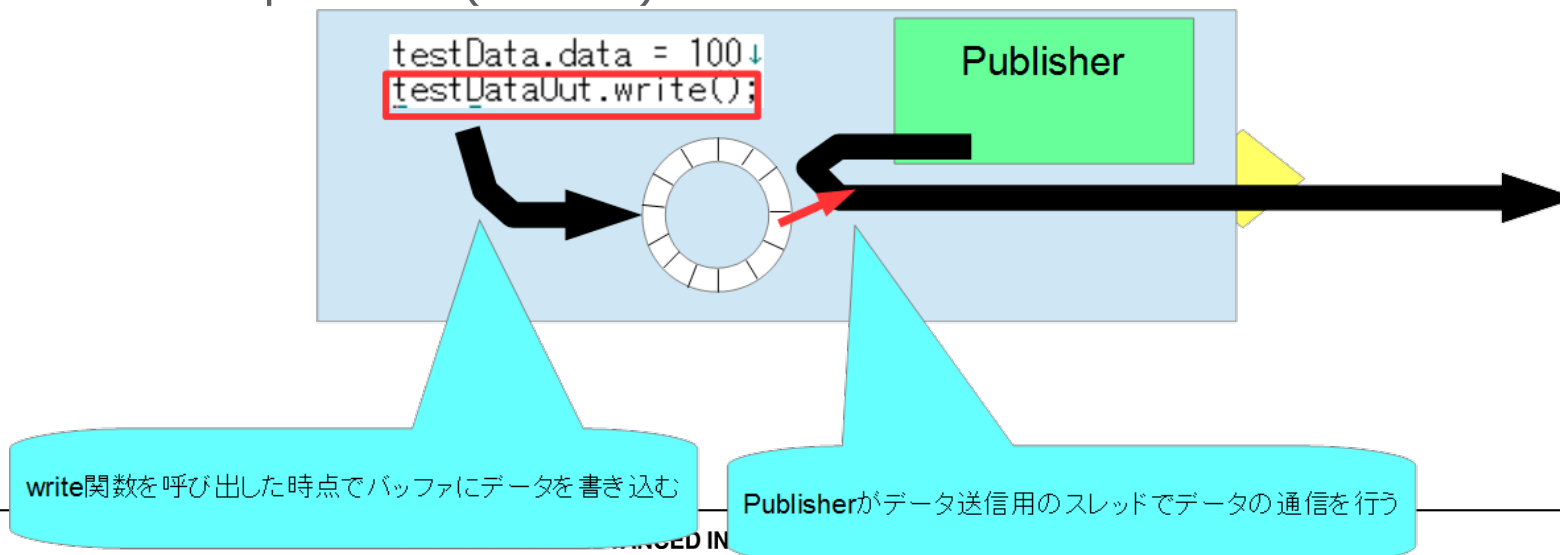


コネクタプロファイルの設定

- SubscriptionType
 - flush(同期)

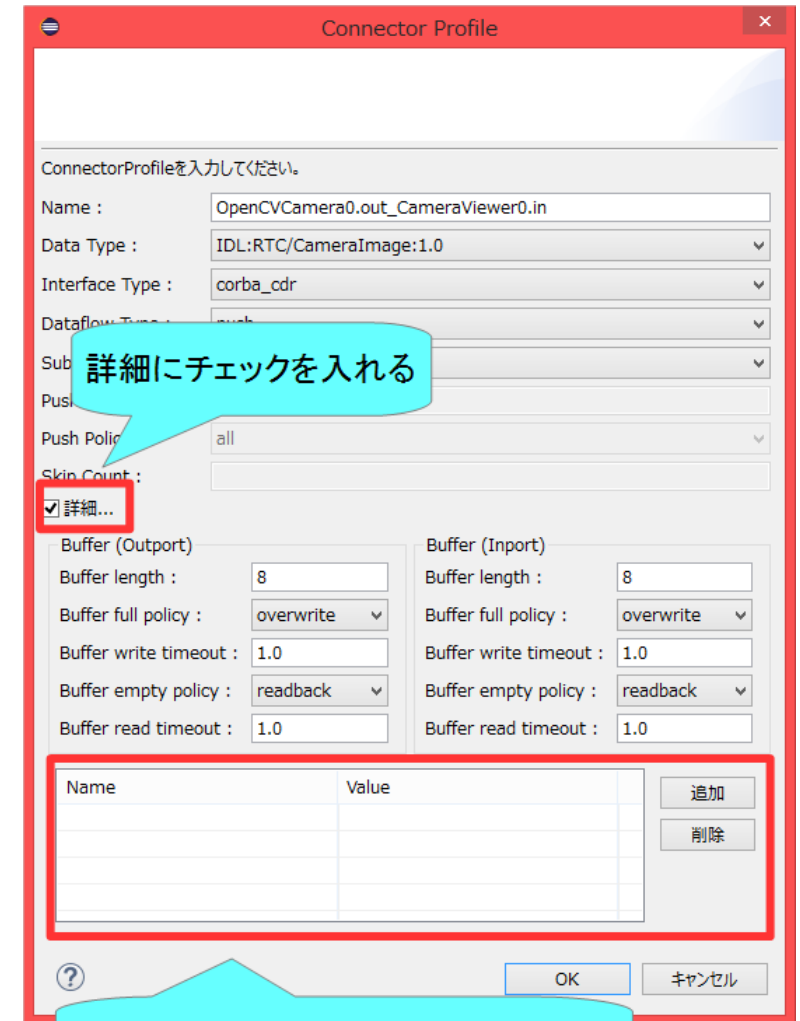


- new、periodic(非同期)



コネクタプロファイルの設定

項目	設定内容
Buffer length	バッファの大きさ
Buffer full policy	データ書き込み時に、バッファフルだった場合の処理. overwrite, do_nothing, blockから選択
Buffer write timeout	データ書き込み時に、タイムアウトイベントを発生させるまでの時間(単位:秒)
Buffer empty policy	データ読み出し時に、バッファが空だった場合の処理. readback, do_nothing, blockから選択
Buffer read timeout	データ読み出し時に、タイムアウトイベントを発生させるまでの時間(単位:秒)



Connector Profile

ConnectorProfileを入力してください。

Name : OpenCVCamera0.out_CameraViewer0.in

Data Type : IDL:RTC/CameraImage:1.0

Interface Type : corba_cdr

Dataflow Type : push

Sub :

Push Policy : all

Push Count :

☒ 詳細...

Buffer (Output)

Buffer length : 8

Buffer full policy : overwrite

Buffer write timeout : 1.0

Buffer empty policy : readback

Buffer read timeout : 1.0

Buffer (Input)

Buffer length : 8

Buffer full policy : overwrite

Buffer write timeout : 1.0

Buffer empty policy : readback

Buffer read timeout : 1.0

Name	Value

追加

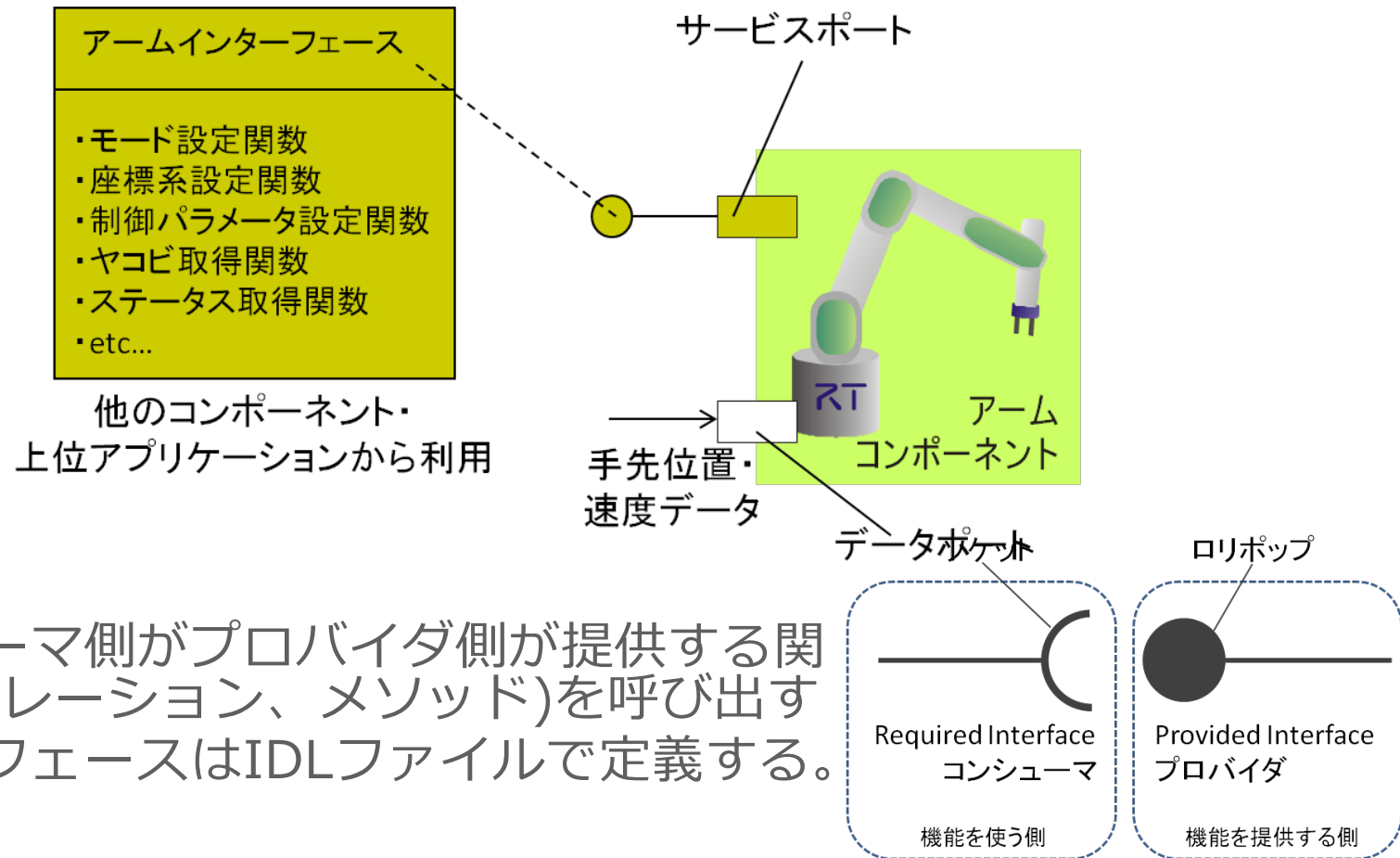
削除

OK

キャンセル

サービスポートについて

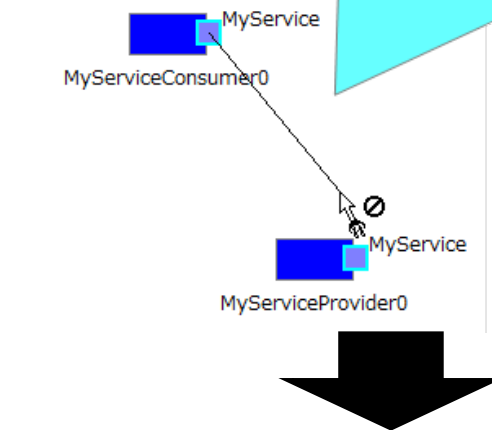
- コマンドレベルのやり取りを行うための仕組み
 - 任意のタイミングで操作を行いたい時などに使用
 - 例えばロボットアームのサーボを停止させる、ハンドを閉じる等



- コンシューマ側がプロバイダ側が提供する関数群(オペレーション、メソッド)を呼び出す
- インターフェースはIDLファイルで定義する。

サービスポートの接続

ポートの片方からもう片方へドラッグアンドドロップ



Port Profile

ポートプロファイルを入力してください。

Name : MyServiceProvider0.MyService_MyServiceConsumer0.MyService

☒ 詳細...

Consumer	Provider

Name	Value

追加 削除

追加 削除

OK キャンセル

名前の設定

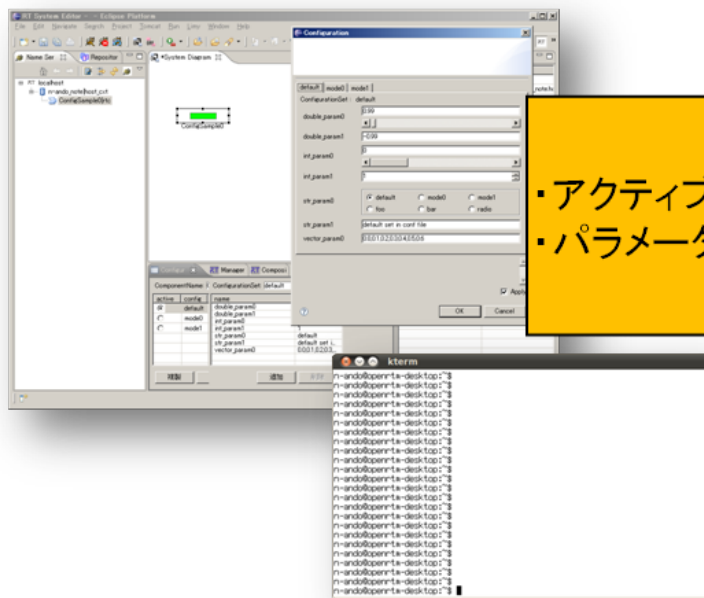
接続するインターフェースの設定
複数のインターフェースが定義されていた場合に、
どのインターフェースに接続するかを設定

その他の設定を直接入力

コンフィギュレーションパラメータについて

- パラメータを外部から操作する仕組み
 - コンポーネント作成後に変更が必要なパラメータを設定する
 - 例えばデバイスが接続されているCOMポート番号の設定等

ツール・アプリケーション



- アクティブセットの変更
- パラメータ値の変更

コンポーネント コンフィギュレーションパラメータ

modeA	名前	Kp
	値	0.2
modeB	名前	Kp
	値	0.4
modeC	名前	Kp
	値	0.6

アクティブ
コンフィギュレーション
セット

パラメータ変数: $m_Kp = 0.4$

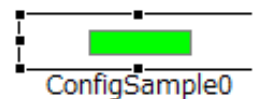
```
onExecute() {
  :
  output(  $m\_Kp * (x\_ref - x)$ );
  :
  return RTC::RTC_OK;
}
```

ツール・アプリケーションから、コンポーネント内部で使用する変数の値を変更できる。

コンフィギュレーションパラメータの設定

対象のRTCをクリックすると表示

「Configuration View」タブを選択



パラメーター一覧

Configuration... RT Manager Cont... RT Composite C... RT Execution Co... RT View

ComponentName: ConfigSample0 ConfigurationSet: default

active	config	name	value
☒	default	double_param0	0.99
☐	mode0	double_param1	-0.99
☐	mode1	int_param0	0
		int_param1	1
		str_param0	default
		str_param1	defau...
		vector_param0	0.0,0...

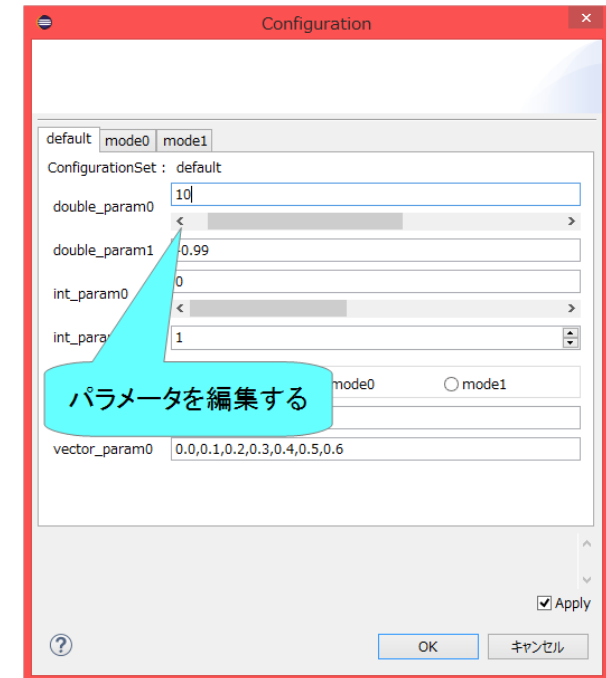
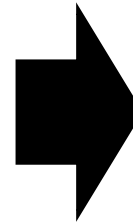
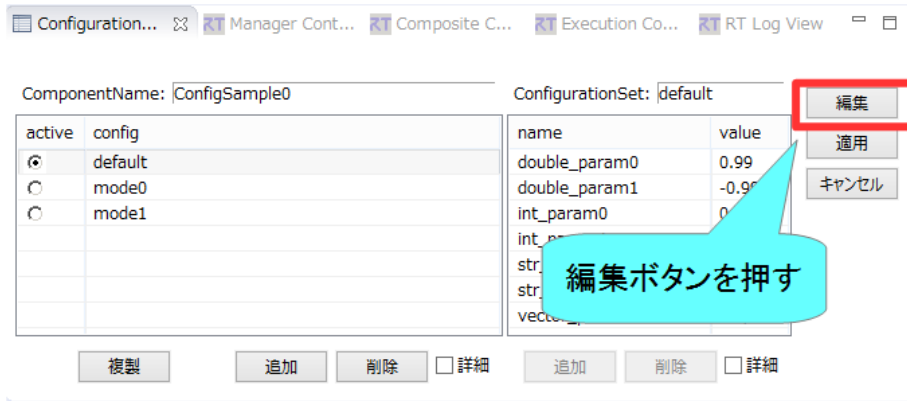
複製 追加 削除 ☐ 詳細 追加 削除 ☐ 詳細

編集 適用 キャンセル

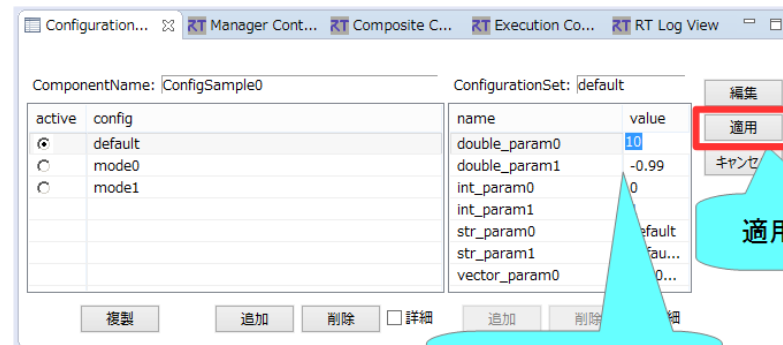
コンフィギュレーションセット一覧

コンフィギュレーションパラメータの設定

方法1



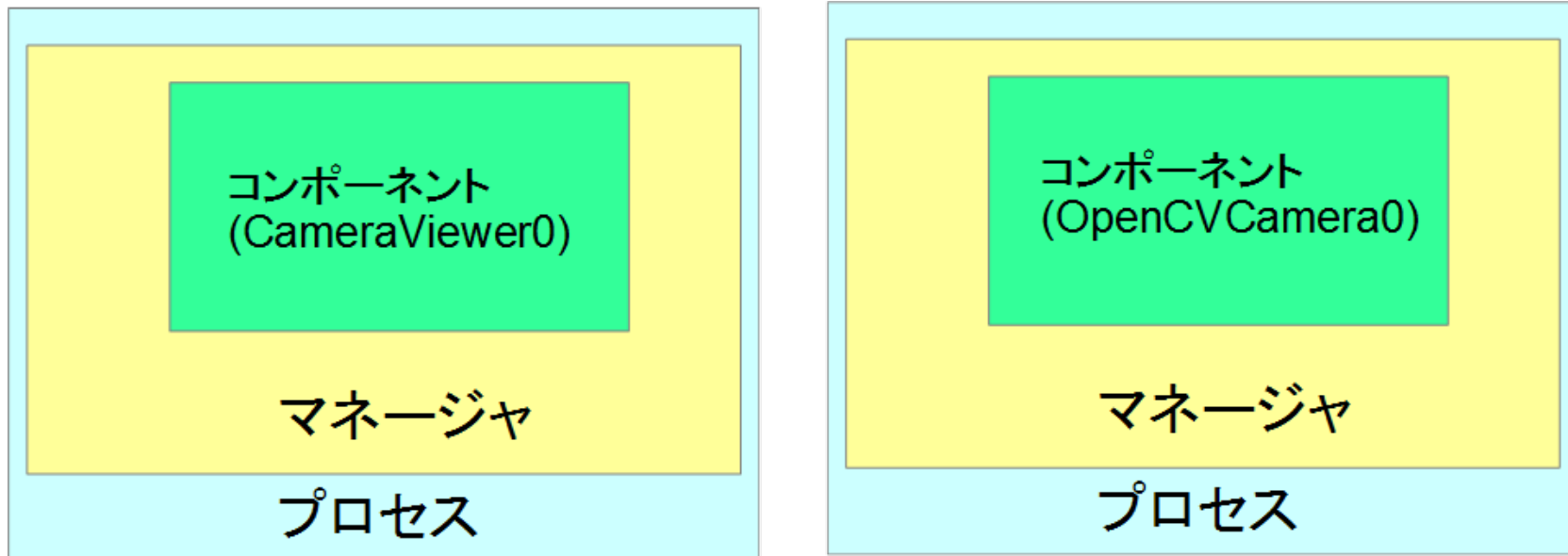
方法2



パラメータを編集する

マネージャの操作

- CameraViewerComp.exe、OpenCVCameraComp.exeのプロセスではマネージャが起動している
 - マネージャがコンポーネントを起動する



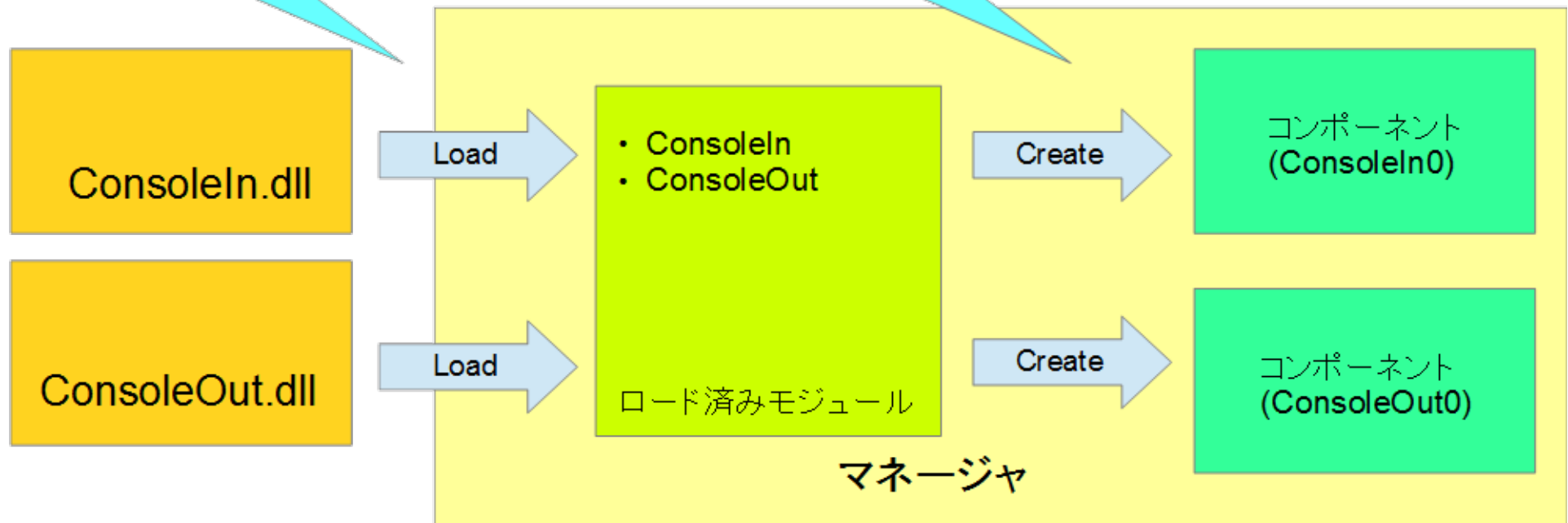
基本的にマネージャは各プロセスに1つ起動する。
マネージャがコンポーネントを起動する

マネージャの操作

1. モジュールをロードする

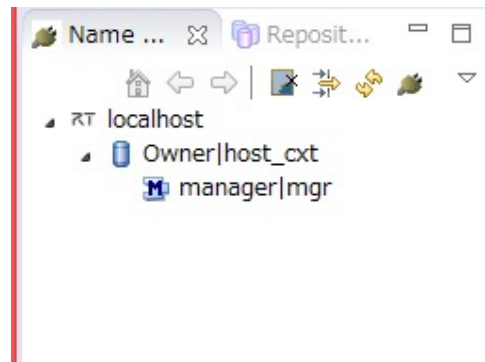
- C++: .dll, .so
- Python: .py
- Java: .jar

2. コンポーネントを起動する



マネージャの操作

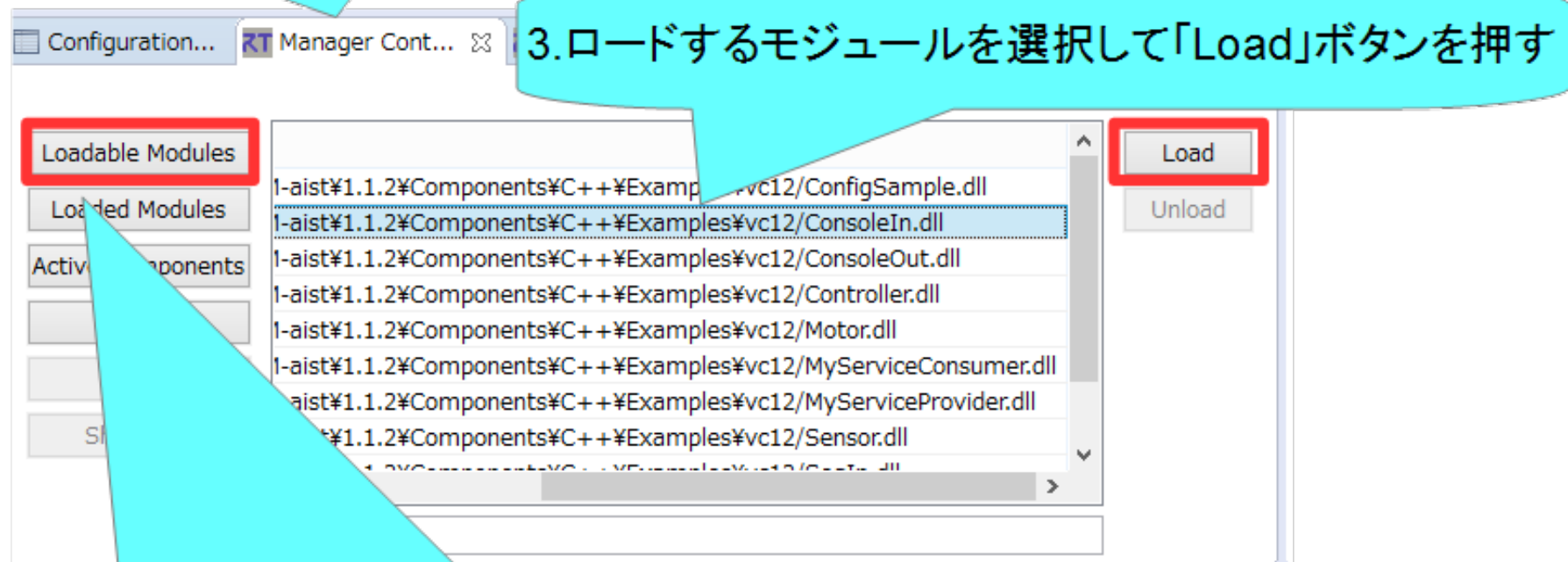
- マスターマネージャの起動、RT System Editorからの操作によるRTCの生成までの手順を説明する
 - rtc.confの設定
 - 「manager.is_master」を「YES」に設定して起動するマネージャをマスターに設定する
 - manager.is_master: YES
 - モジュール探索パスの設定
 - manager.modules.load_path: ., C:¥¥Program Files (x86)¥¥OpenRTM-aist¥¥1.2.2¥¥Components¥¥C++¥¥Examples¥¥vc12
 - 作成したrtc.confを設定ファイルの指定してrtcd.exeを起動する
 - rtcdはコマンドプロンプトからrtcd.exeを入力するか、OpenRTM-aistをインストールしたフォルダからコピーして使用する
 - rtcdはマネージャの起動のみを行う
 - ~Comp.exeは起動時に特定のコンポーネントの起動も行う
 - RT Syetem Editorのネームサービスビューにマネージャが表示される



マネージャの操作

- モジュールのロード

1.「Manager Control View」タブを選択



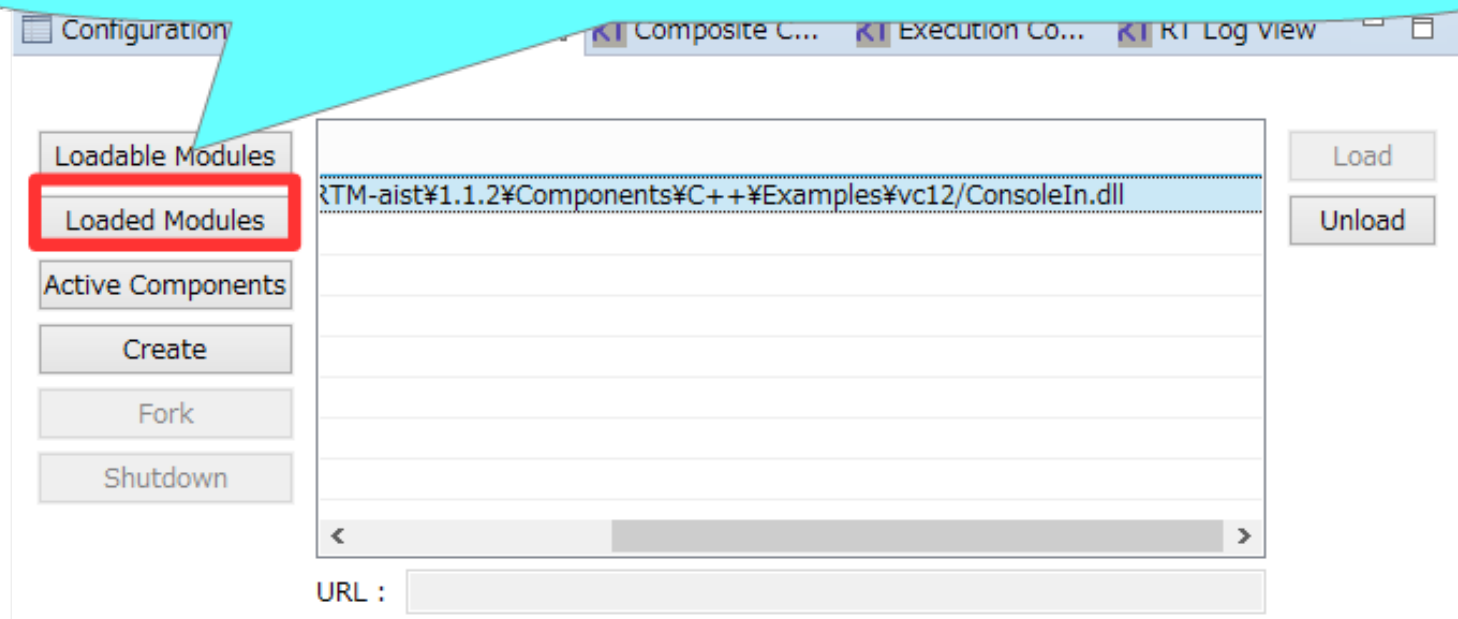
3.ロードするモジュールを選択して「Load」ボタンを押す

2.「Loadable Modules」ボタンを押すとロード可能なモジュール一覧表示

マネージャの操作

- モジュールのロード

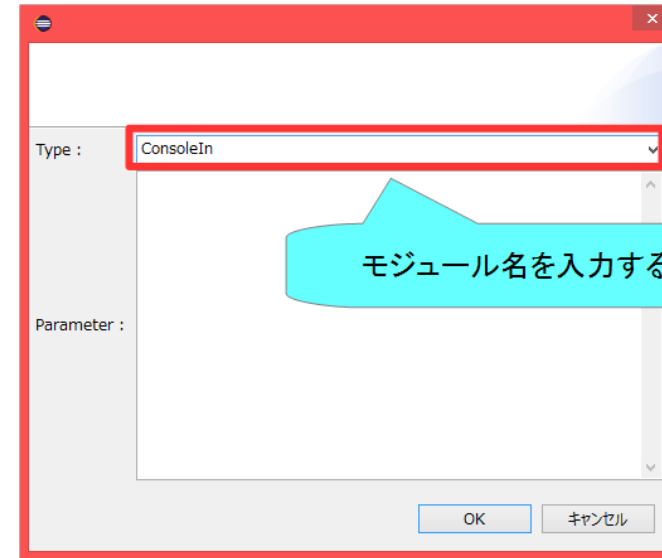
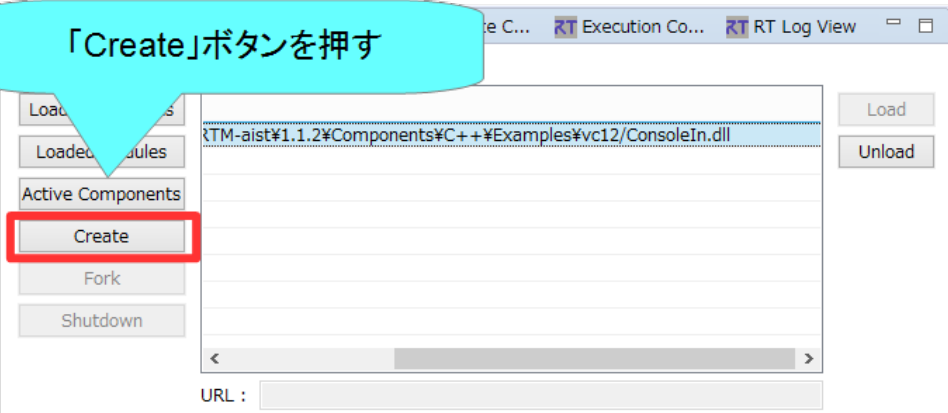
「Loaded Modules」ボタンを押すとロード済みのモジュール一覧表示



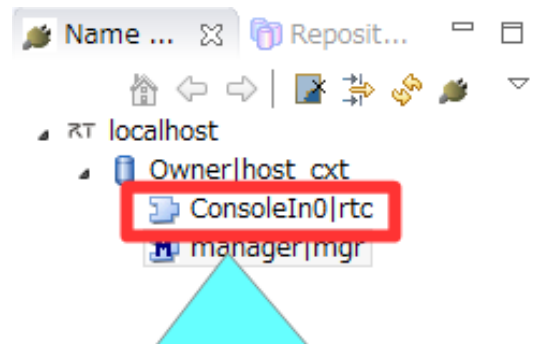
マネージャの操作

• RTCの生成

「Create」ボタンを押す



モジュール名を入力する



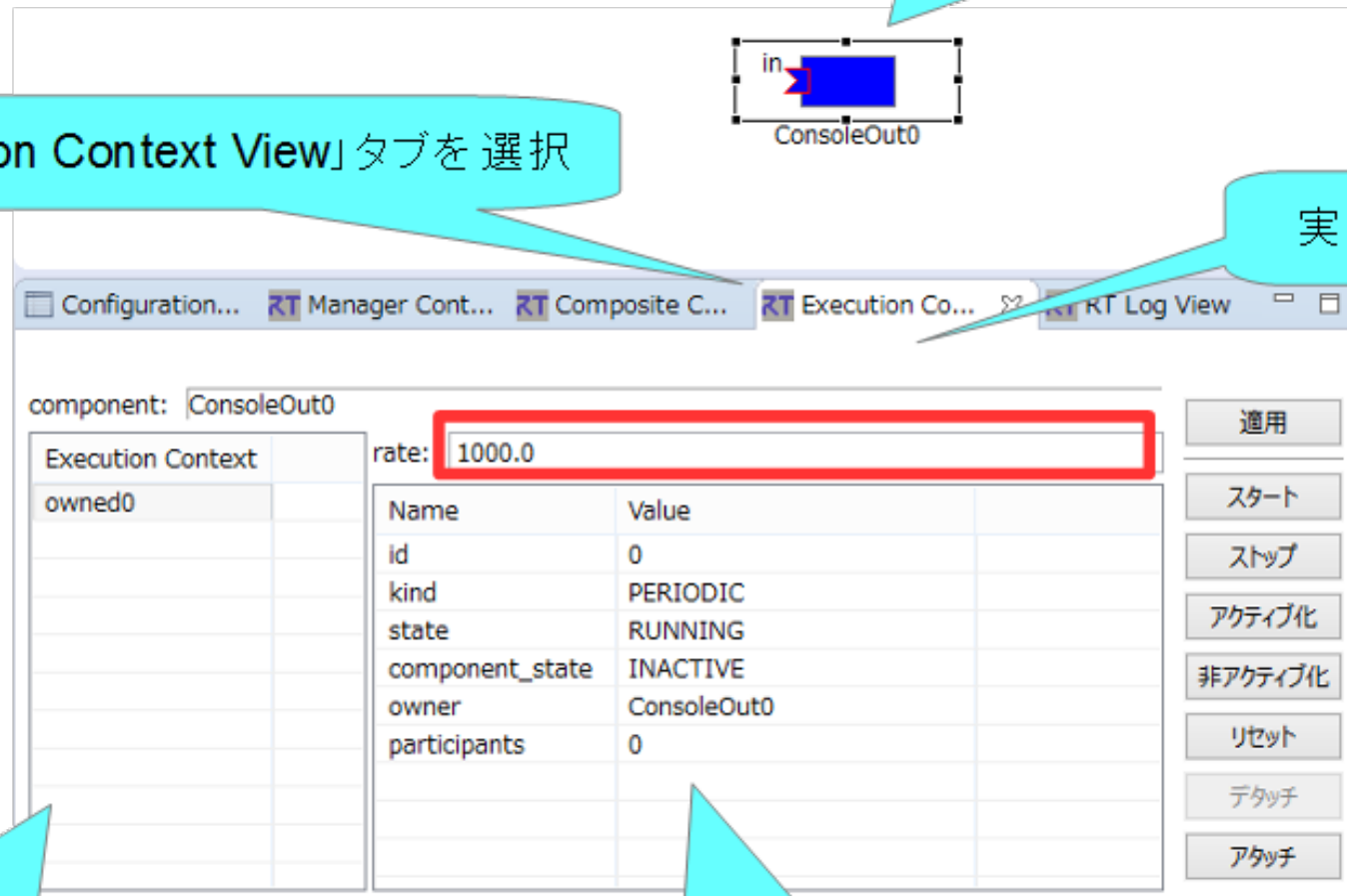
指定したRTCが起動する

実行コンテキストの操作

RTCをクリック

「Execution Context View」タブを選択

実行周期



component: ConsoleOut0

Execution Context		rate: 1000.0
owned0		
Name	Value	
id	0	
kind	PERIODIC	
state	RUNNING	
component_state	INACTIVE	
owner	ConsoleOut0	
participants	0	

Buttons: 適用, スタート, ストップ, アクティブ化, 非アクティブ化, リセット, デタッチ, アタッチ

関連付けている実行コンテキスト一覧

実行コンテキストの情報

実行コンテキストの操作

- 実行周期の設定

実行周期を変更

適用ボタンを押すと反映

Configuration... RT Manager Cont... Composite C... RT Execution Co... RT RT Log View

component: ConsoleOut0

Execution Context	owned0	rate: 10.0	Name	Value
			id	0
			kind	PERIODIC
			state	RUNNING
			component_state	INACTIVE
			owner	ConsoleOut0
			participants	0

適用

スタート

ストップ

アクティブ化

非アクティブ化

リセット

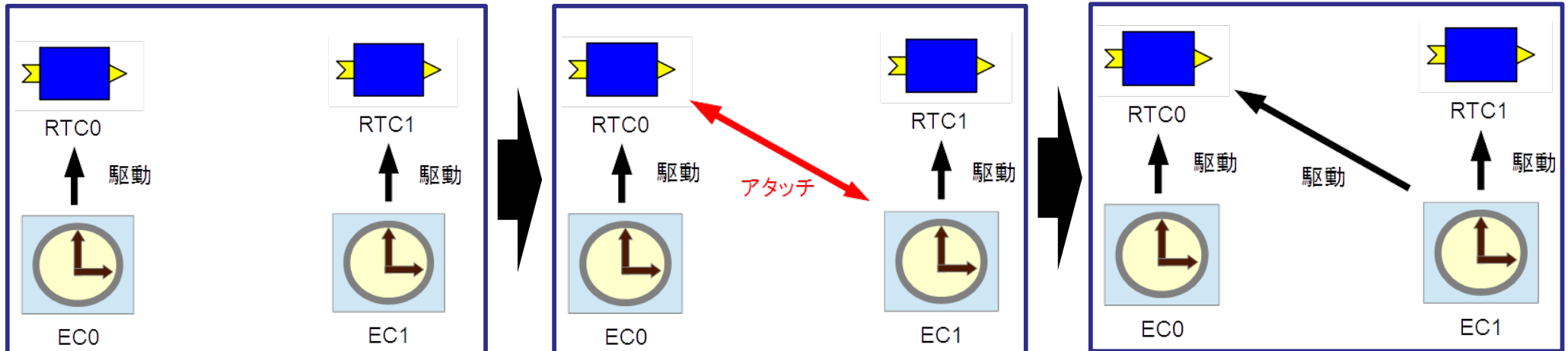
デタッチ

アタッチ

実行コンテキストの操作

実行コンテキストの関連付け

- RTC起動時に生成した実行コンテキスト以外の実行コンテキストと関連付け
 - 関連付けた実行コンテキストでRTCを駆動させる
- 他のRTCとの実行を同期させる



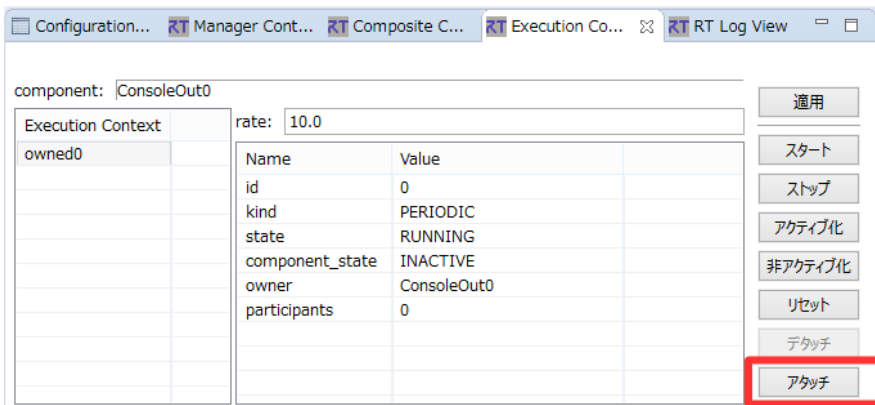
RTC0とRTC1は別々の
実行コンテキストで駆動

RTC0とEC1を関連付ける

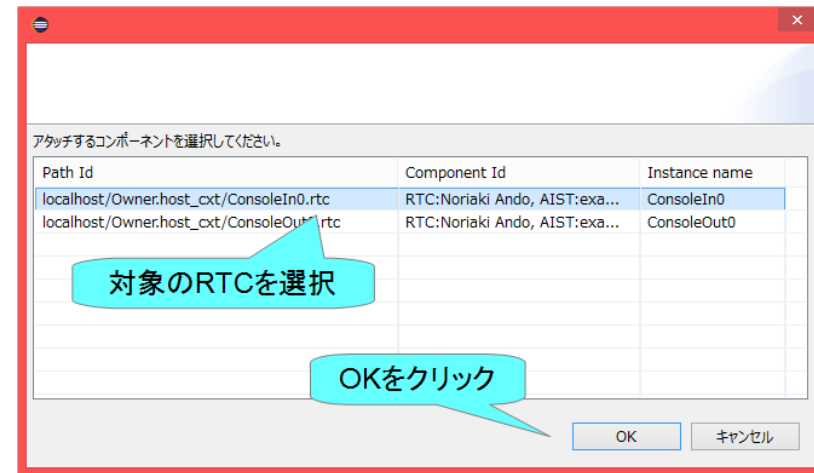
EC1がRTC0も駆動する

実行コンテキストの操作

- 実行コンテキストの関連付け



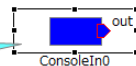
アタッチボタンを押す



対象のRTCを選択

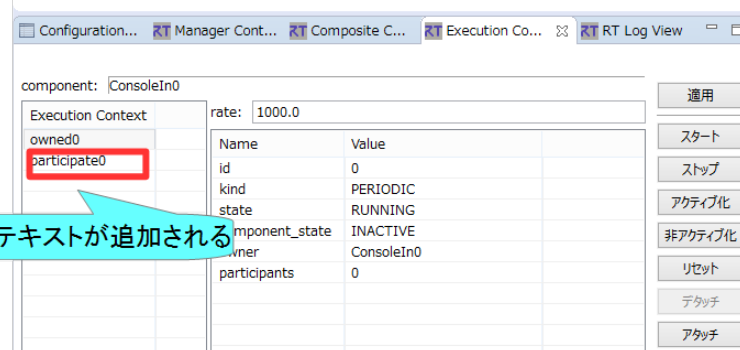
OKをクリック

対象のRTCをクリック



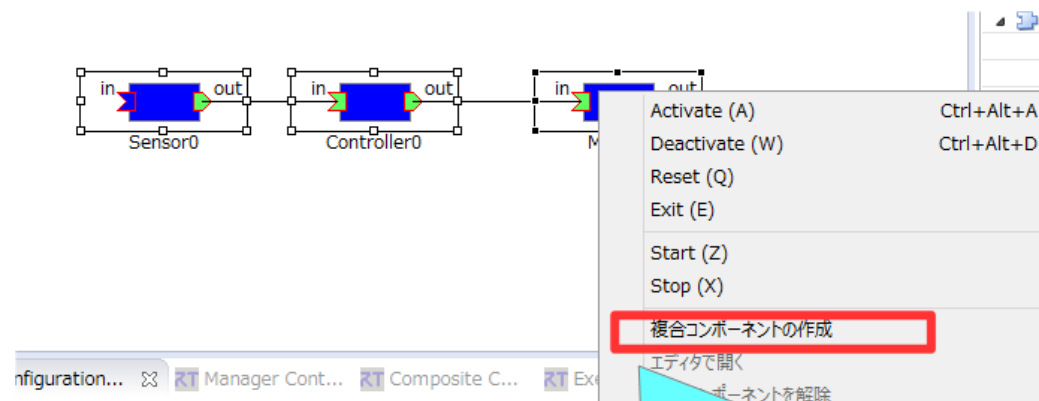
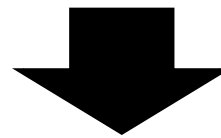
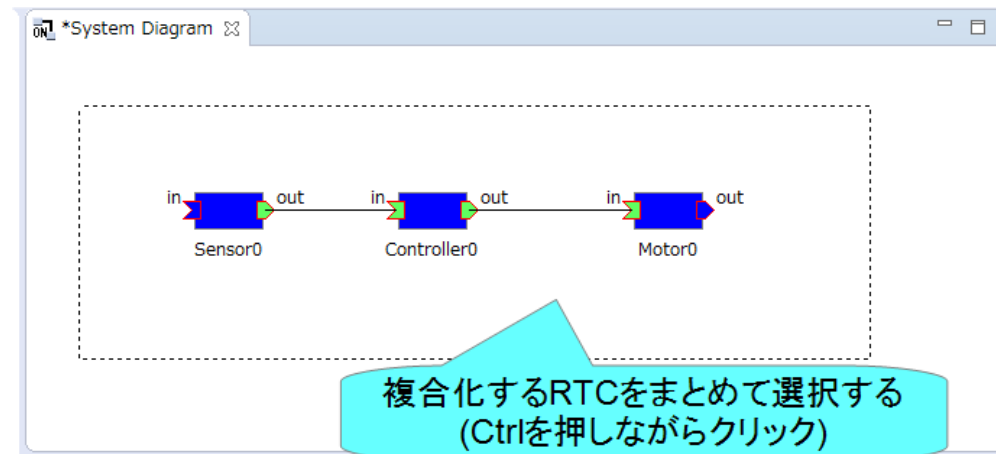
ConsoleOut0

実行コンテキストが追加される



複合コンポーネントの操作

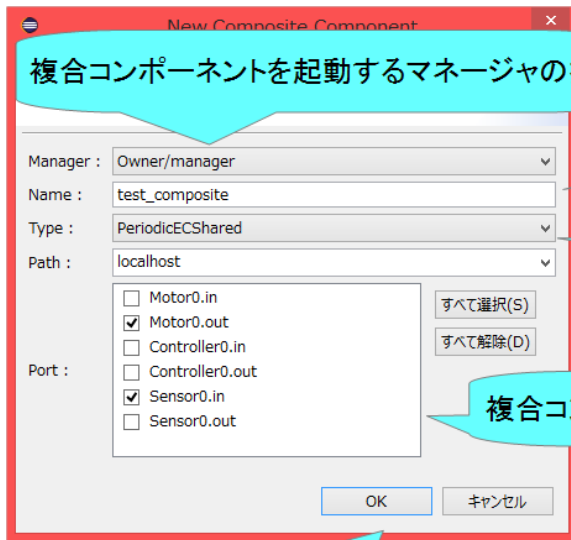
- 複合コンポーネントの生成



右クリックして「複合コンポーネントの作成」を選択

複合コンポーネントの操作

- 複合コンポーネントの生成



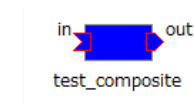
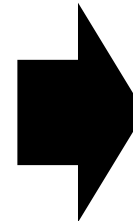
複合コンポーネントを起動するマネージャの指定

名前を設定

タイプを設定

複合コンポーネントに表示するポートを指定

OKをクリックする



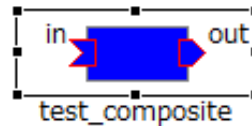
複合コンポーネントが生成される

- Type

- 以下の3種類から選択可能

- PeriodicECShared
 - 実行コンテキストの共有
 - PeriodicStateShared
 - 実行コンテキスト、状態の共有
 - Grouping
 - グループ化のみ

複合コンポーネントの操作



複合コンポーネントをクリック

「Composite Component View」タブを選択

Configuration... RT Manager Cont... RT Composite C... RT Execution Co... RT RT Log View

component: test_composite type: PeriodicECShared

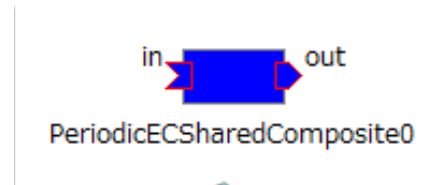
	component	port
☐	Controller0	Controller0.in
☐	Controller0	Controller0.out
☒	Sensor0	Sensor0.in
☐	Sensor0	Sensor0.out
☐	Motor0	Motor0.in
☒	Motor0	Motor0.out

適用
キャンセル

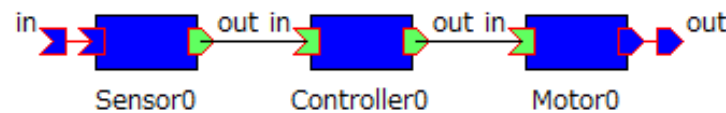
適用ボタンを押すと変更を反映

表示するポートの選択

複合コンポーネントの操作



RTCをダブルクリック

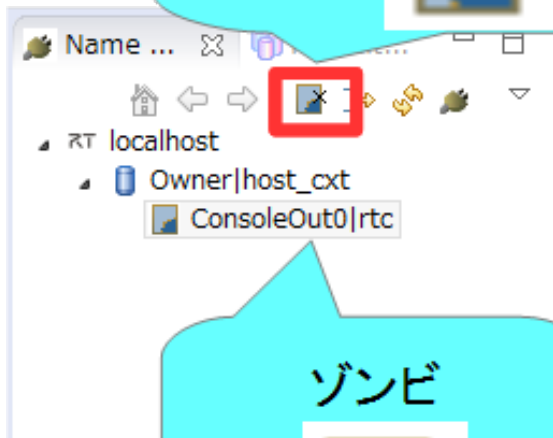


子コンポーネントを表示

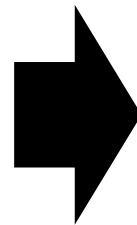
ゾンビの削除

- RTCのプロセスが異常終了する等してネームサーバーにゾンビが残った場合、以下の手順で削除する

ゾンビクリアボタンを押す



ゾンビ



ゾンビが消える

RT System Editorに関する設定

