2024年11月21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://openrtm.org/openrtm/ja/node/7295
  - 短縮URL
    - https://x.gd/utAFE

- 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。

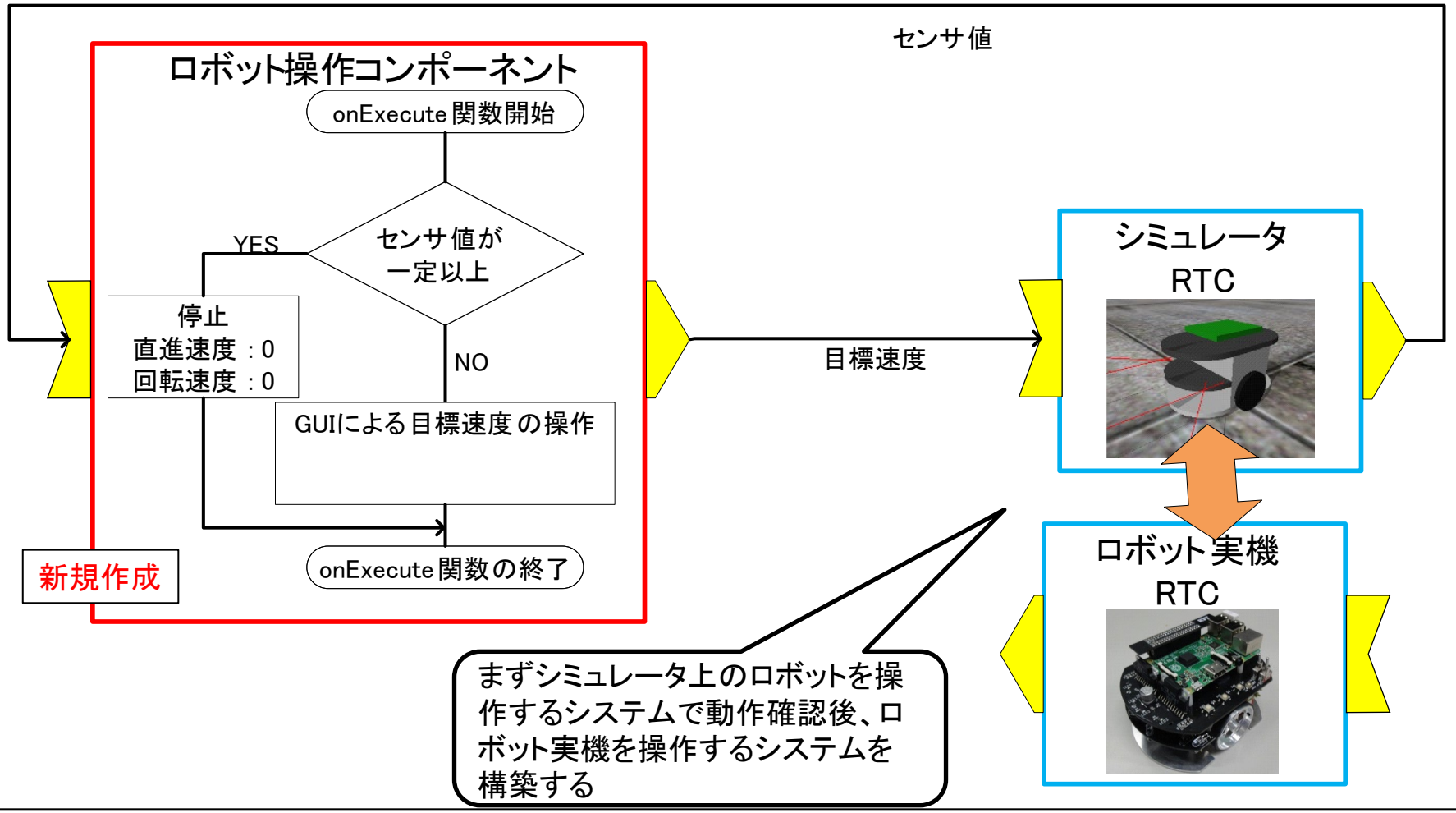
概要

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. シミュレータロボットと接続してテスト
4. 実機ロボットと接続してテスト

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と他のRTCでシステムを構築する方法を学ぶ。
４．シミュレータ内ロボットRTCに接続して動作させる方法を学ぶ。
５．実機ロボットRTCに接続してシミュレータ同様に動作可能であることを確認する。

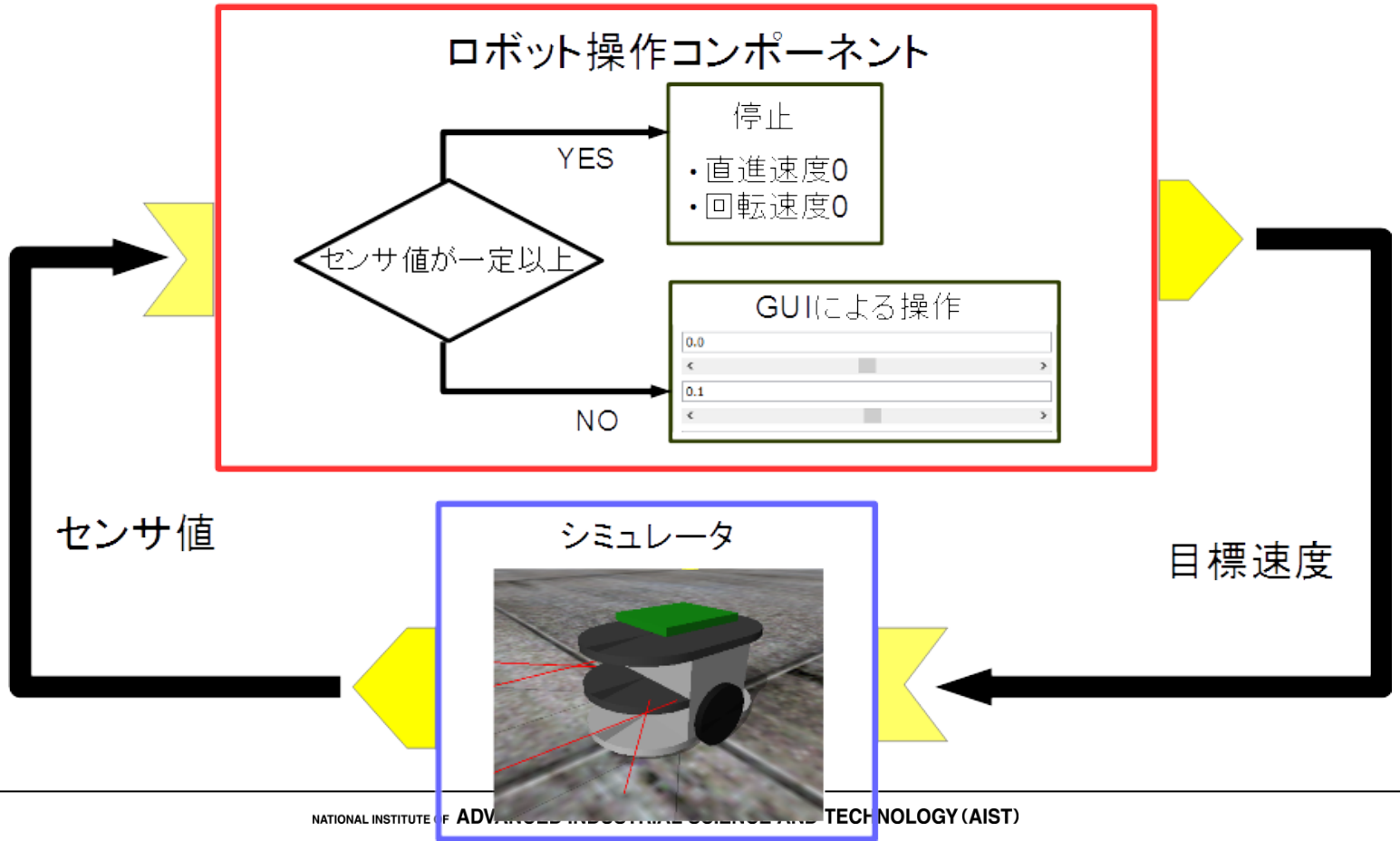
実習内容

- 小型移動ロボットを操作するRTCの作成



実習内容（1）

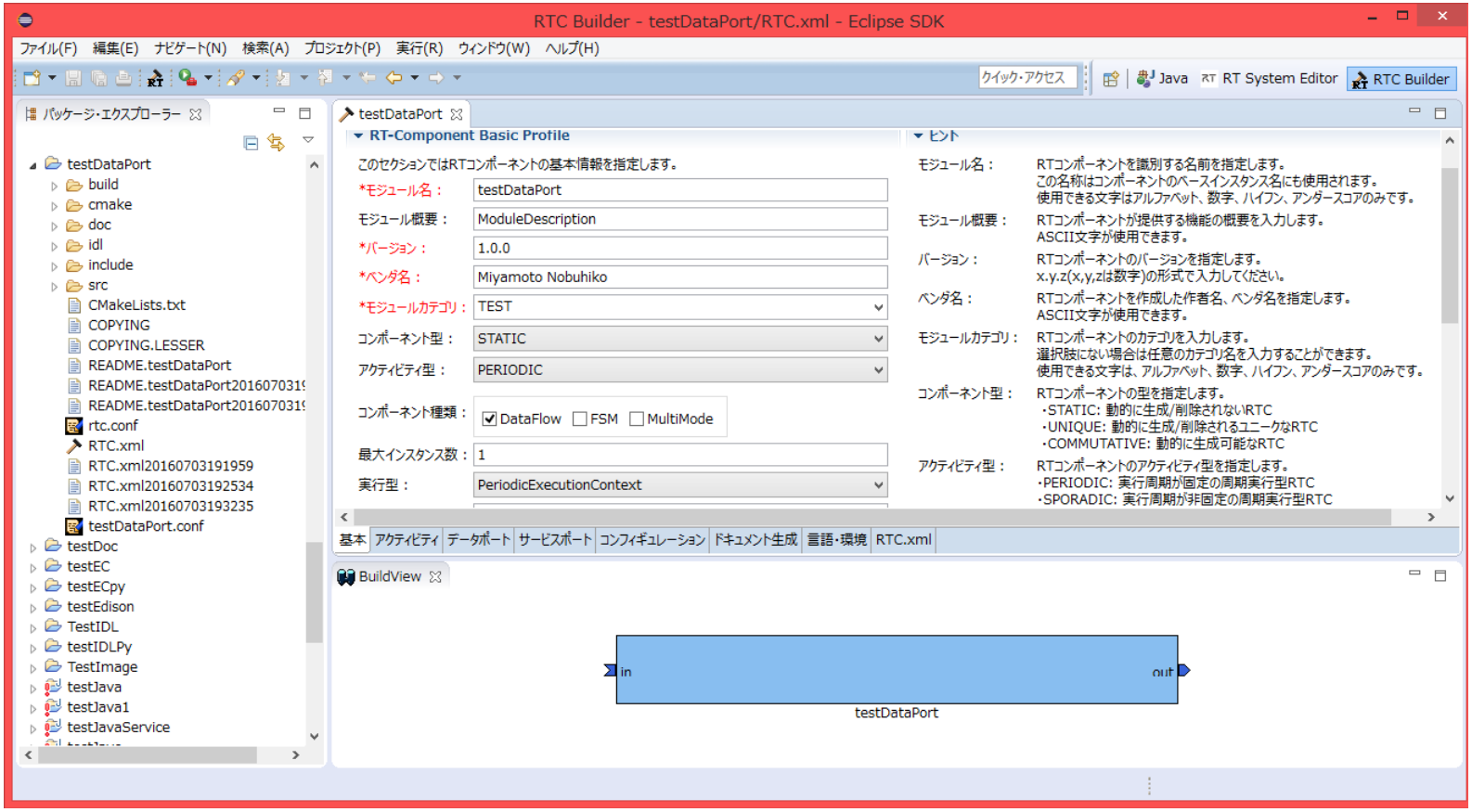
- シミュレータ上の車輪型移動ロボット(Raspberry Piマウス)の操作を行うコンポーネントの作成
  - GUIにより目標速度入力
  - センサ値が一定以上の場合に停止



（１）RTCBuilderによるひな形コードの生成

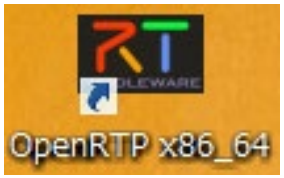
RTC Builder

- 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
  - デスクトップのショートカットをダブルクリック

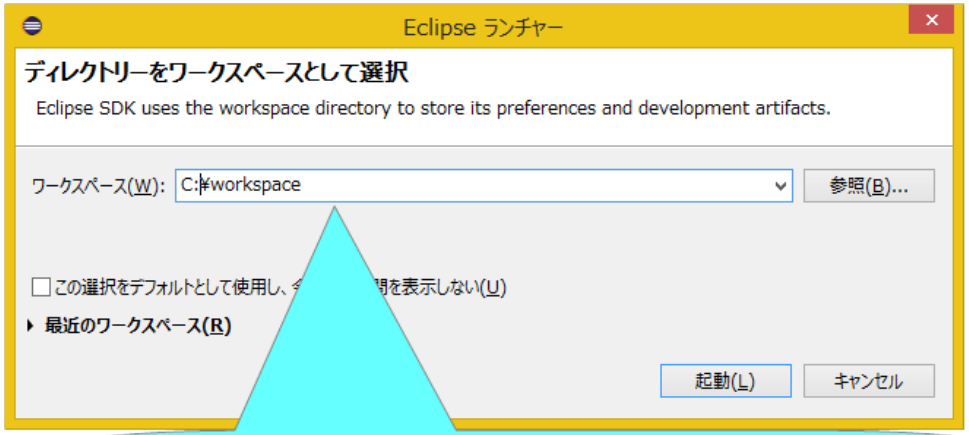


RTC Builderの起動(参考)

- Windows 10

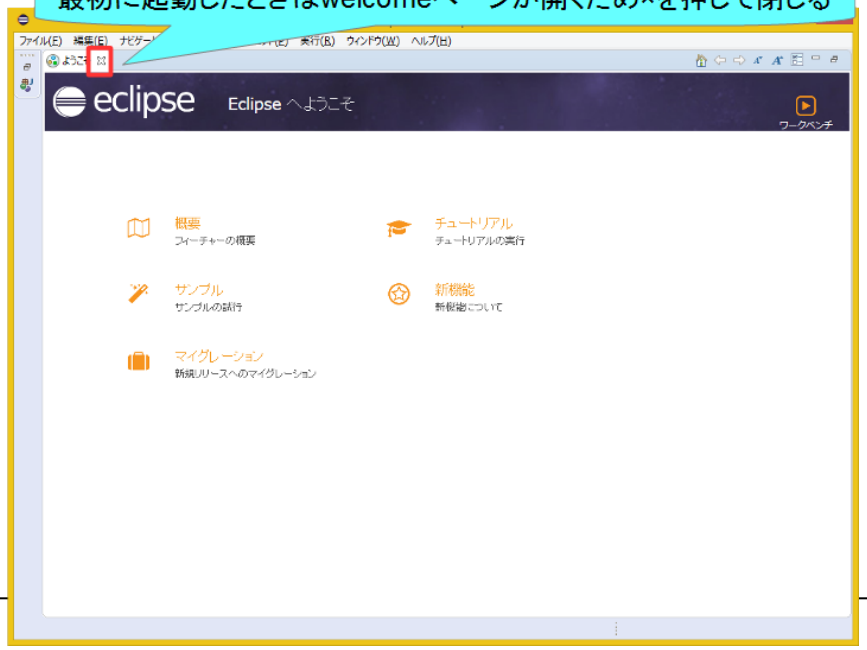
左下の「ここに入力して検索」に「OpenRTP」と入力

RTC Builderの起動

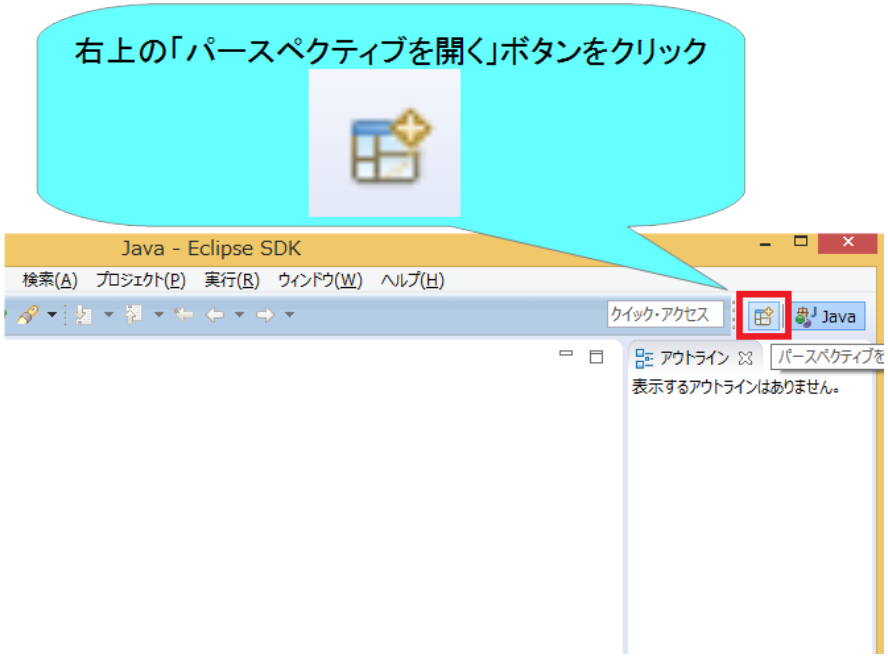


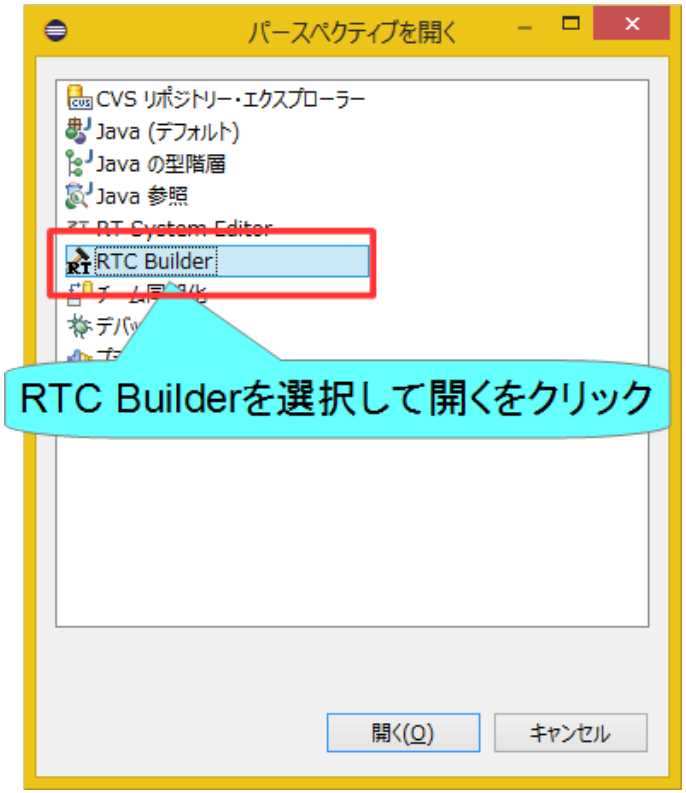
ワークスペースに適当な場所を指定して起動をクリックする

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



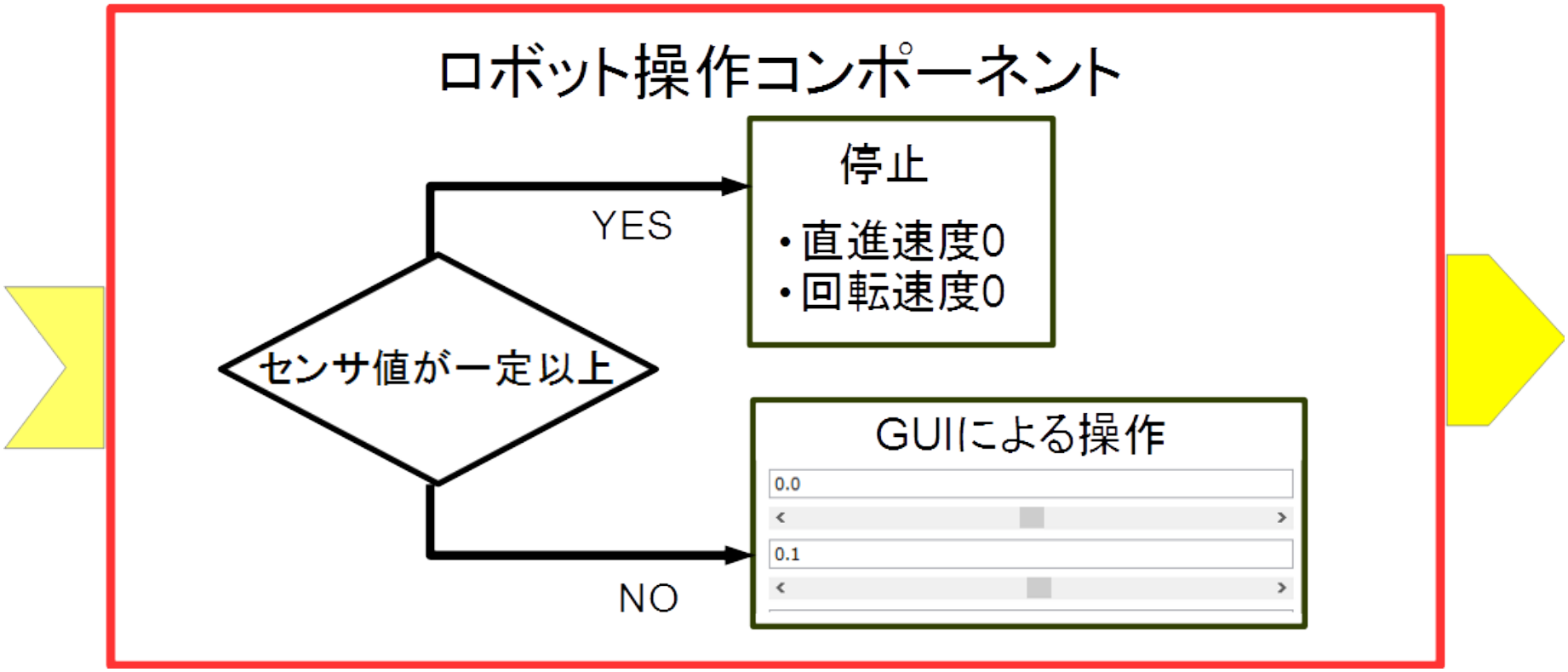
RTC Builderの起動





プロジェクト作成

- RobotControllerコンポーネントのスケルトンコードを作成する。
  - 車輪型移動ロボット操作コンポーネント
    - GUIでロボットを操作
    - センサ値が一定以上の場合に停止



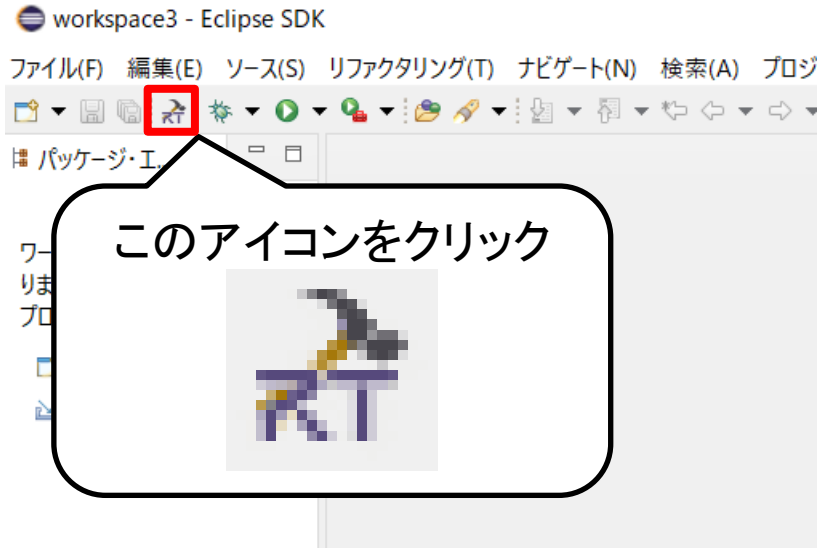
資料

- WEBページ
  - https://openrtm.org/openrtm/ja/node/6550
  - 短縮URL
    - https://x.gd/3EiRi
- シミュレータ
  - https://github.com/OpenRTM/RTM_Tutorial/releases/download/tutorial20241121/RTM_Tutorial.zip
  - 短縮URL
    - https://x.gd/x6bar

プログラム

	11月21日（木）
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:** 241121-01.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:** 241121-02.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）シミュレータロボットと接続してテスト （4）実機ロボットと接続してテスト ・チュートリアル(RTコンポーネントの作成入門、Raspberry Pi Mouse、Windows) **資料:** 241121-03.pdf **シミュレータ:** RTM_Tutorial.zip

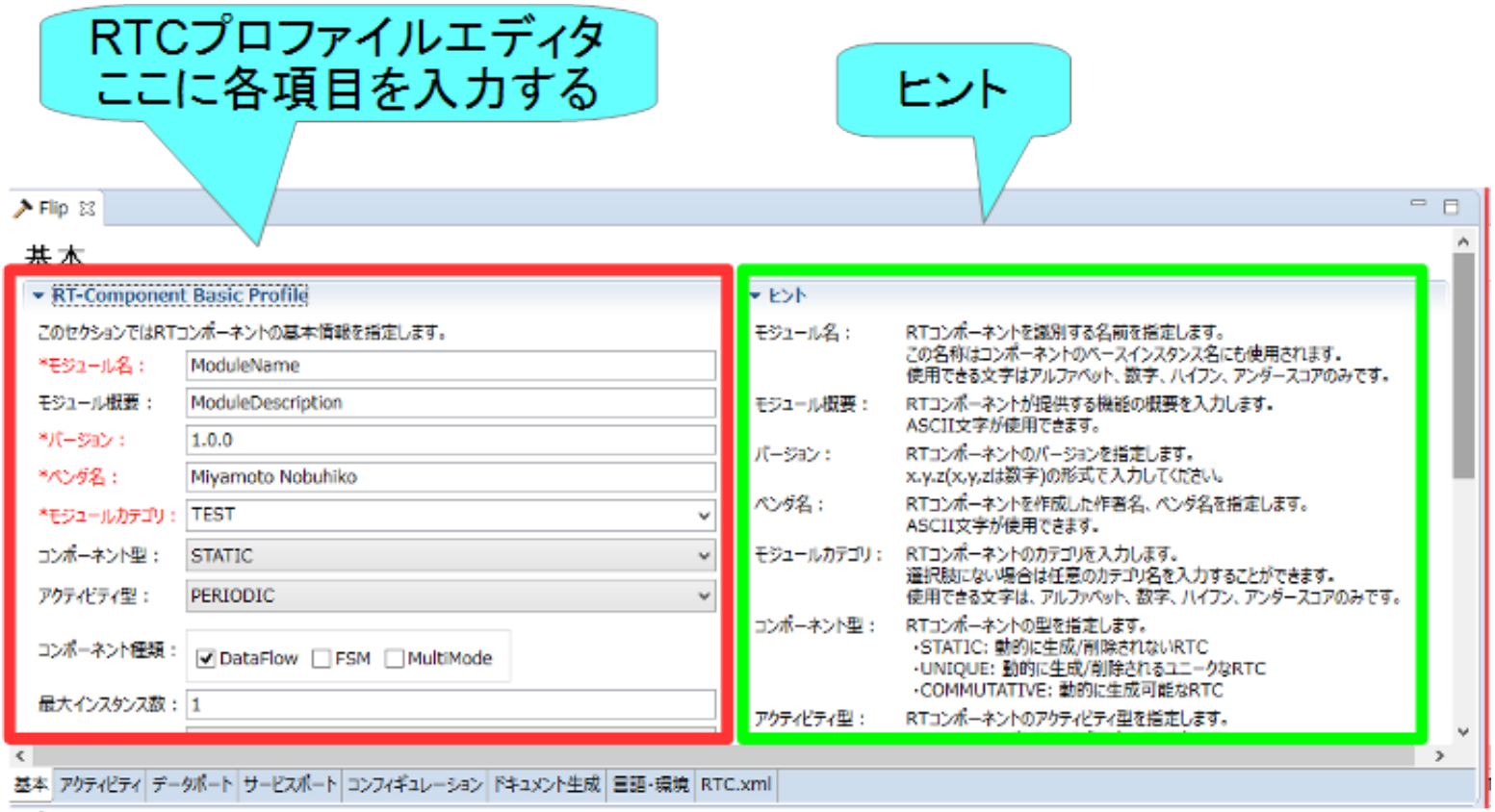
プロジェクト作成



- Eclipse起動時にワークスペースに指定したディレクトリに「RobotController」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

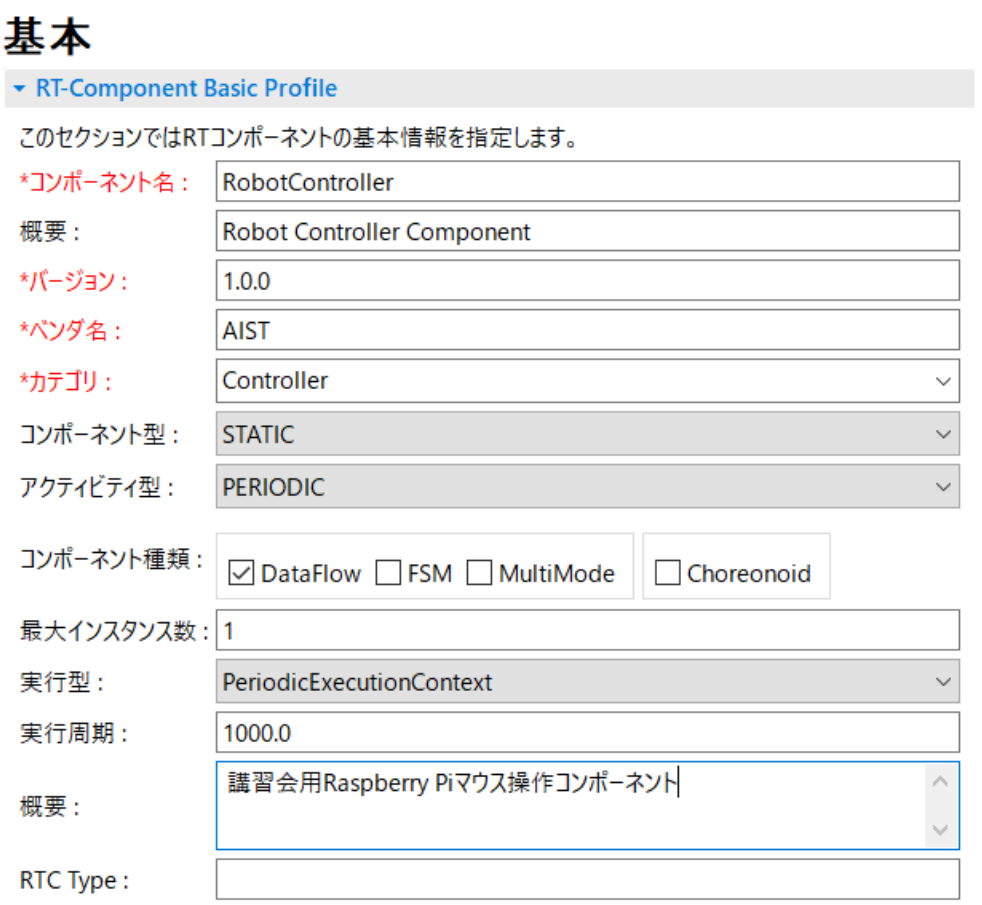
基本プロファイルの入力

- 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



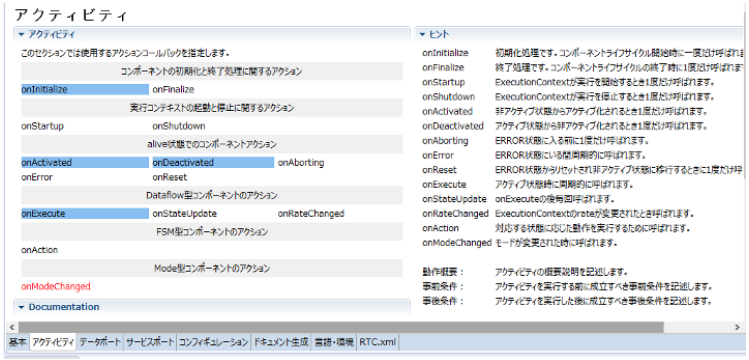
基本プロファイルの入力

- **コンポーネント名**
  - **RobotController**
- モジュール概要
  - 任意(Robot Controller 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(Controller)
- 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



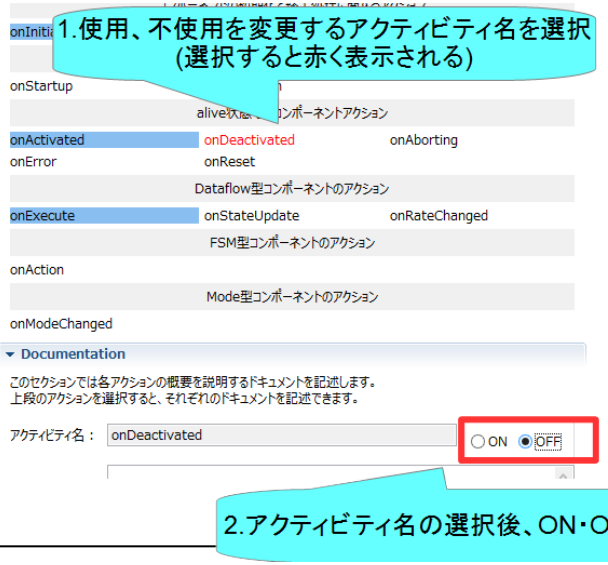
アクティビティの設定

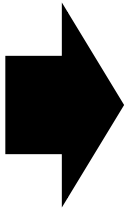
- 使用するコールバック関数を設定する

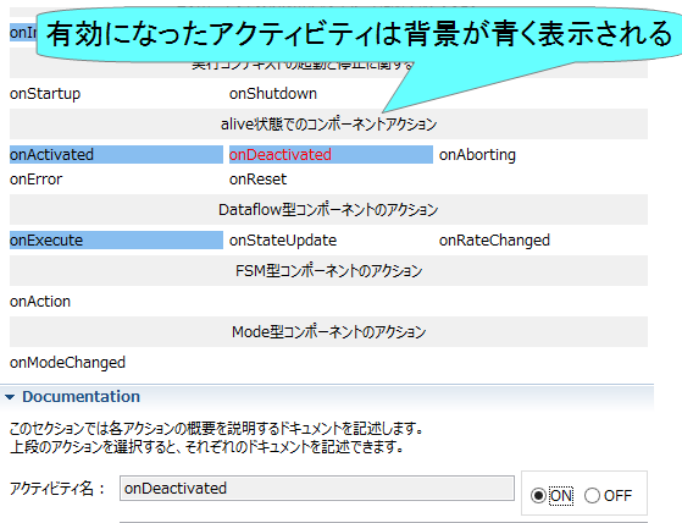


「アクティビティ」タブを選択

- 指定アクティビティを有効にする手順





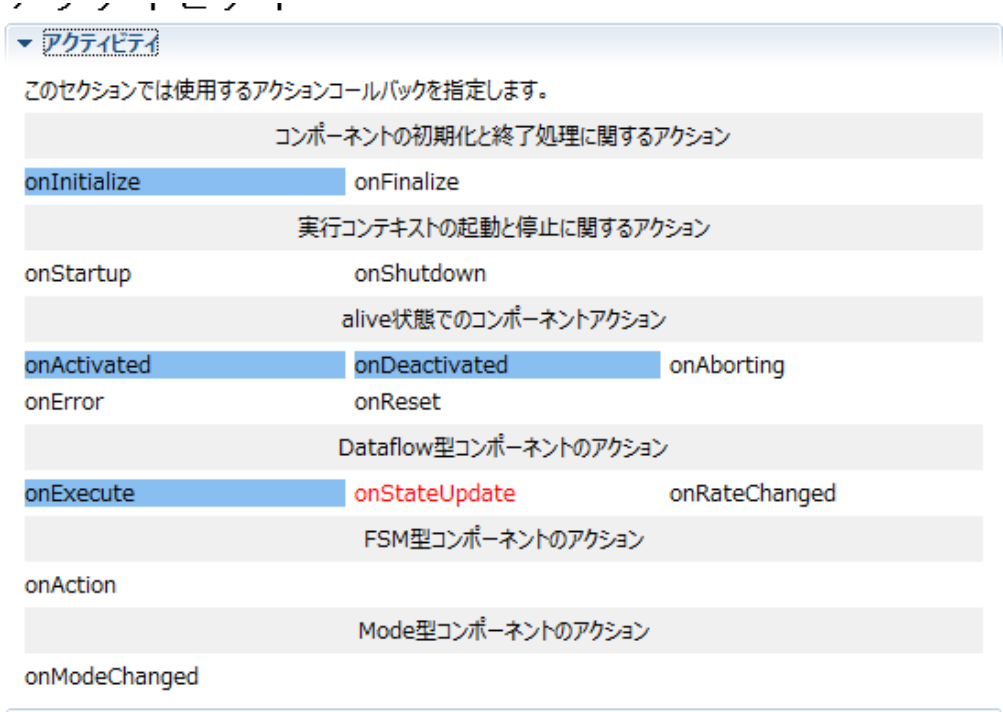


アクティビティの設定

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

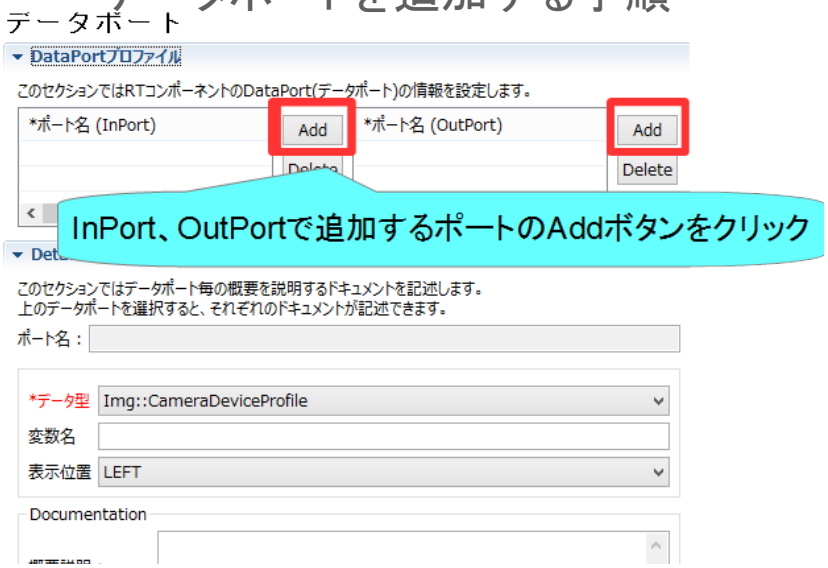


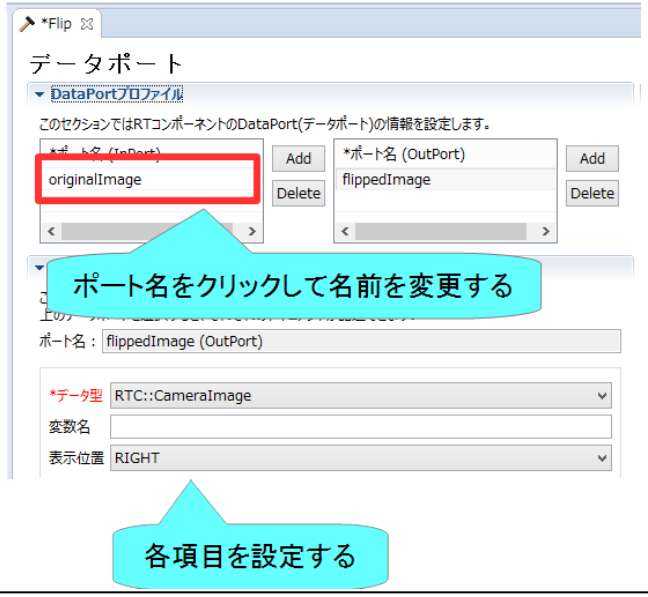
データポートの設定

- 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

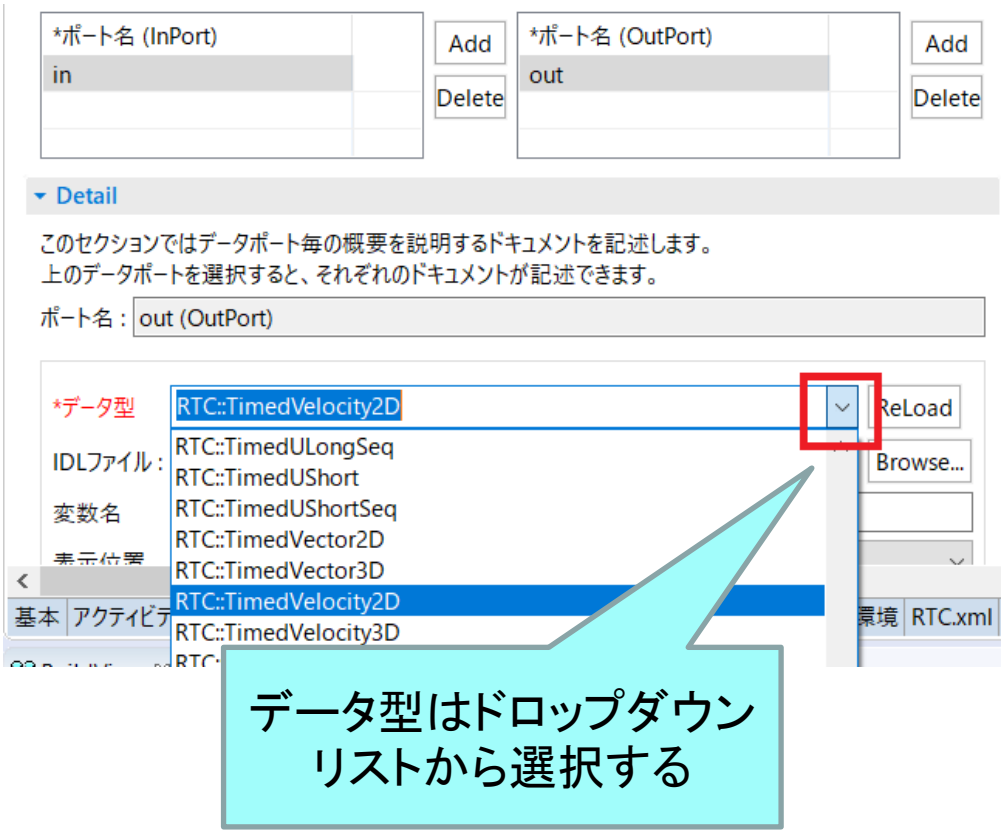




データポートの設定

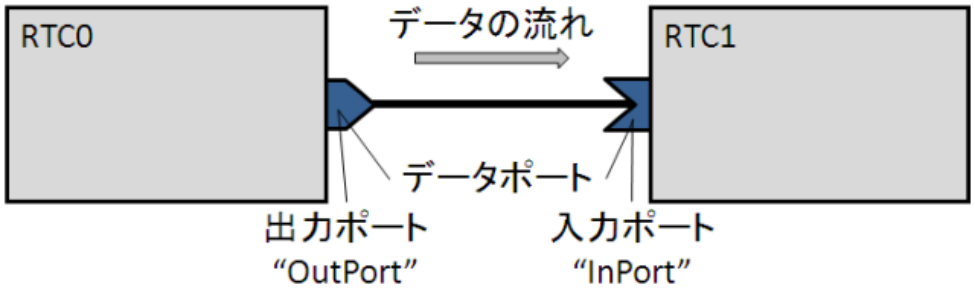
- 以下のInPortを設定する
  - **in**
    - データ型：
      RTC::TimedShortSeq
    - 他の項目は任意
    - ※TimedShort型と間違えないようにしてください。

- 以下のOutPortを設定する
  - **out**
    - データ型：
      RTC::TimedVelocity2D
    - 他の項目は任意
    - ※TimedVelocity3D型、TimedVector2Dと間違えないようにしてください

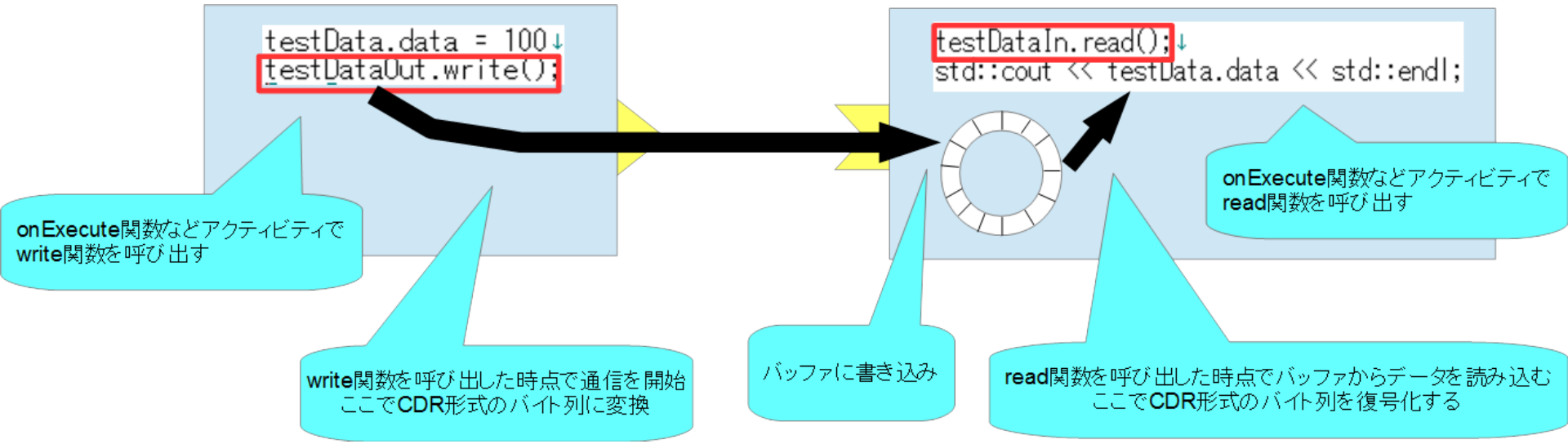


データポートについて

- 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

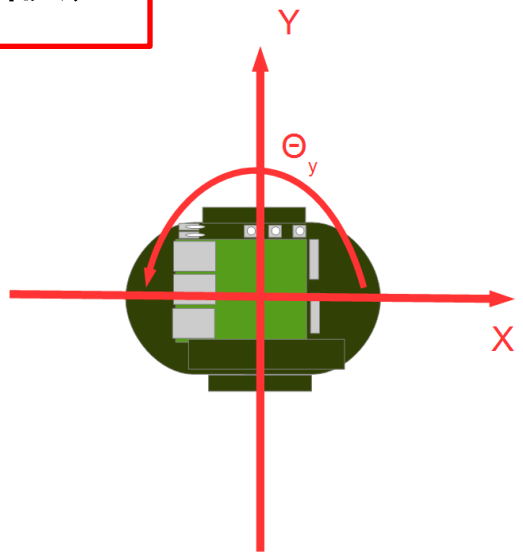


RTC::TimedVelocity2D型について

- ExtendedDataTypes.idlで定義されている**移動ロボットの速度**を表現するためのデータ型
  - **vx**: X軸方向の速度
  - **vy**: Y軸方向の速度(車輪が横滑りしないと仮定すると0)
  - **va**: Z軸周りの角速度

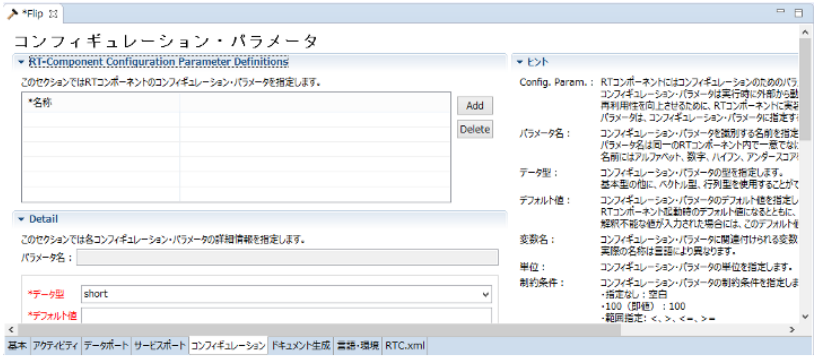
vxで直進速度、vaで回転速度を設定

```
struct Velocity2D
{
    /// Velocity along the x axis in metres per second.
    double vx;
    /// Velocity along the y axis in metres per second.
    double vy;
    /// Yaw velocity in radians per second.
    double va;
};
```



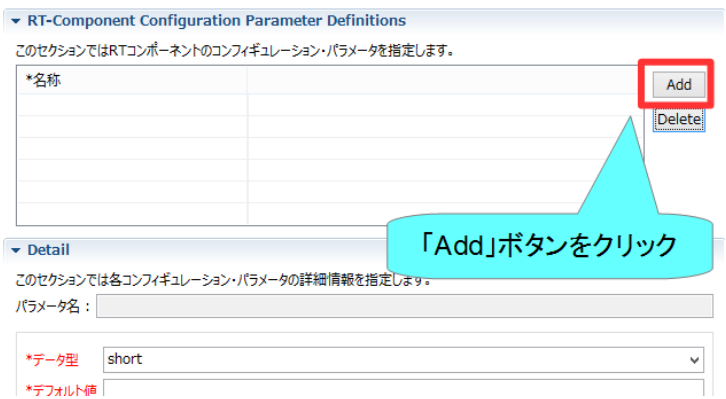
コンフィギュレーションの設定

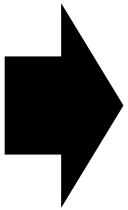
- コンフィギュレーションパラメータの追加、設定を行う

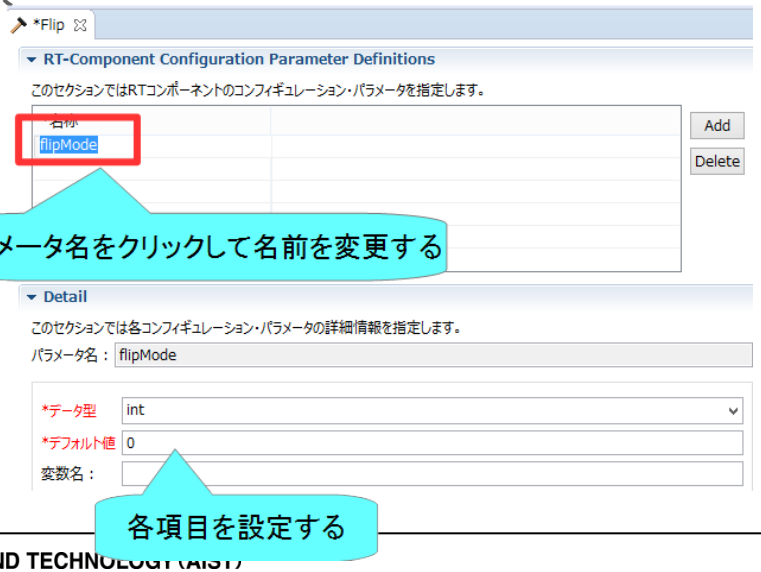


「コンフィギュレーション」タブを選択

- コンフィギュレーションパラメータを追加する手順



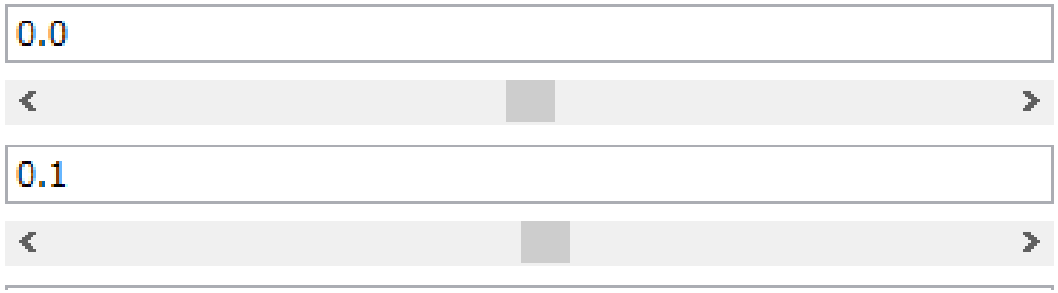




コンフィギュレーションの設定

- 以下のコンフィギュレーションパラメータを設定する
  - **speed_x**
    - データ型：double
    - デフォルト値： 0.0
    - 制約条件：-1.5<x<1.5
    - Widget：slider
    - Step: 0.01
    - 他の項目は任意
  - **speed_r**
    - データ型：double
    - デフォルト値： 0.0
    - 制約条件：-2.0<x<2.0
    - Widget：slider
    - Step: 0.01
    - 他の項目は任意

GUI(スライダー)による移動ロボット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

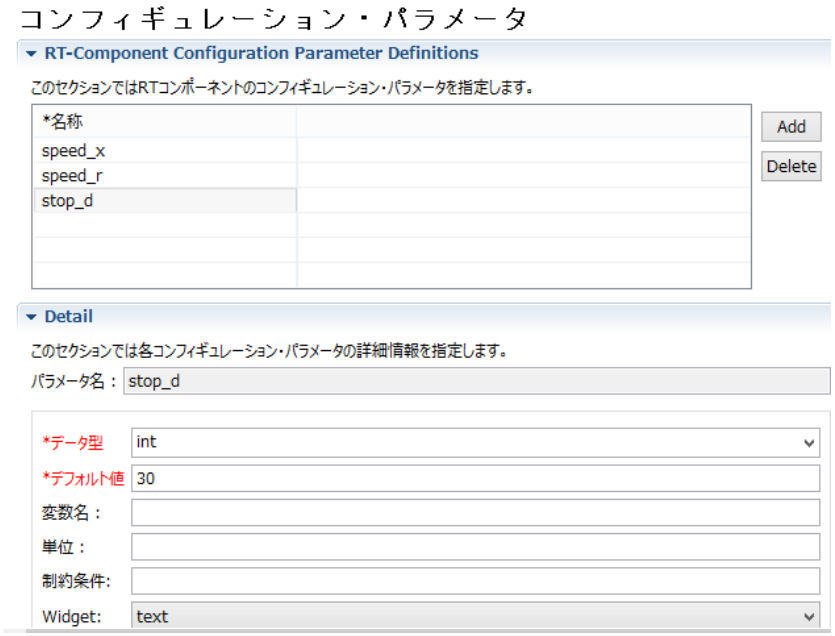
- 制約条件：(0,1,2,3)
- Widget：checkbox

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

コンフィギュレーションの設定

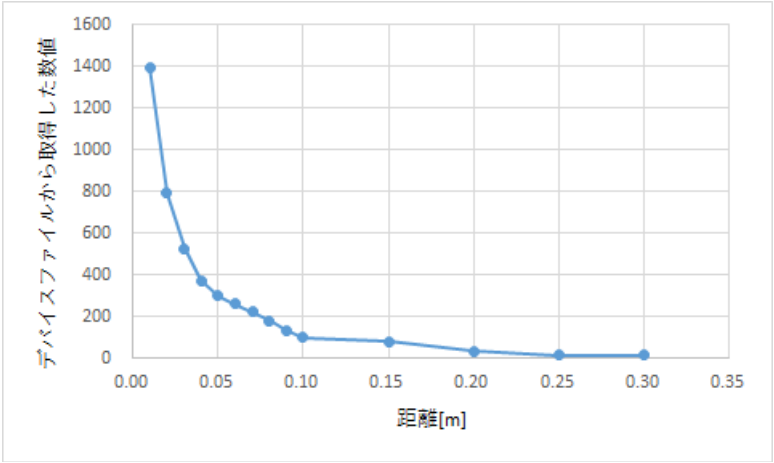
- 以下のコンフィギュレーションパラメータを追加
  - **stop_d**
    - データ型：int
    - デフォルト値：30
    - 他の項目は任意

センサ値がこの値以上の場合に停止



Raspberry Piマウスの距離センサ

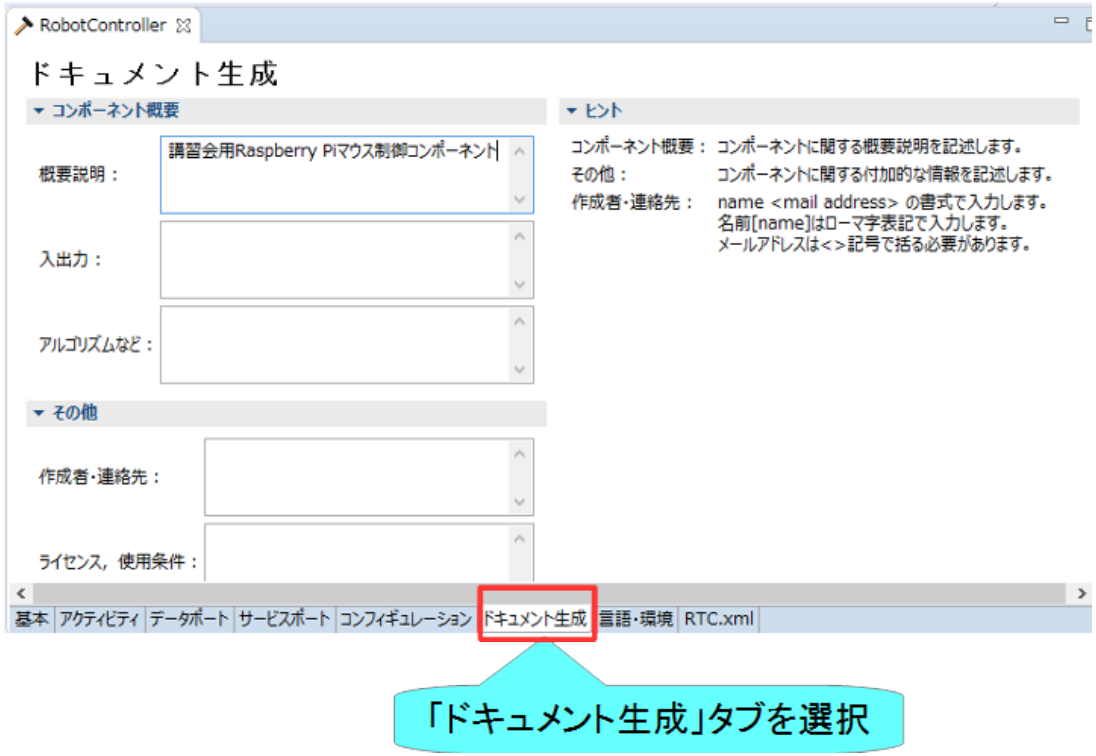
- Raspberry Piマウス実機には距離センサが搭載されている
  - 計測した値は物体までの**距離が近いほど大きな値**となる



- シミュレータでもこのデータに近い値を計算して出力している

ドキュメントの設定

- 各種ドキュメント情報を設定



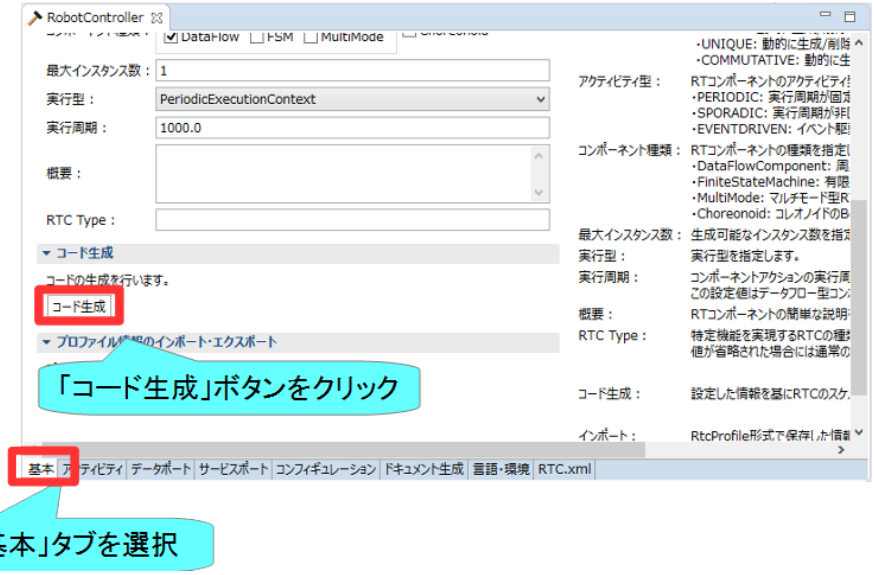
- 今回は適当に設定しておいてください。
  - 空白でも大丈夫です

言語の設定

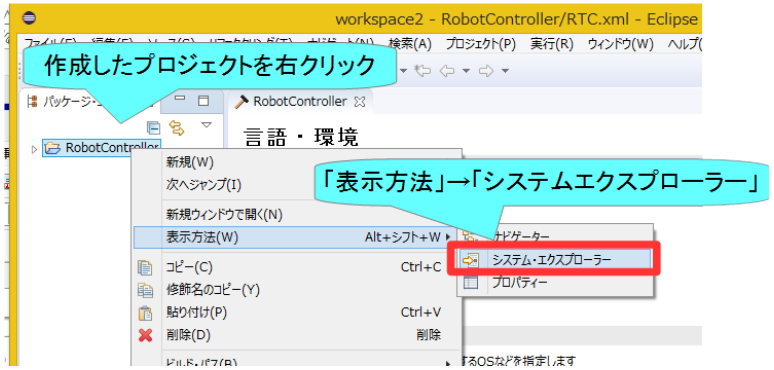
- 実装する言語，動作環境に関する情報を設定

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¥RobotController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、RobotController.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



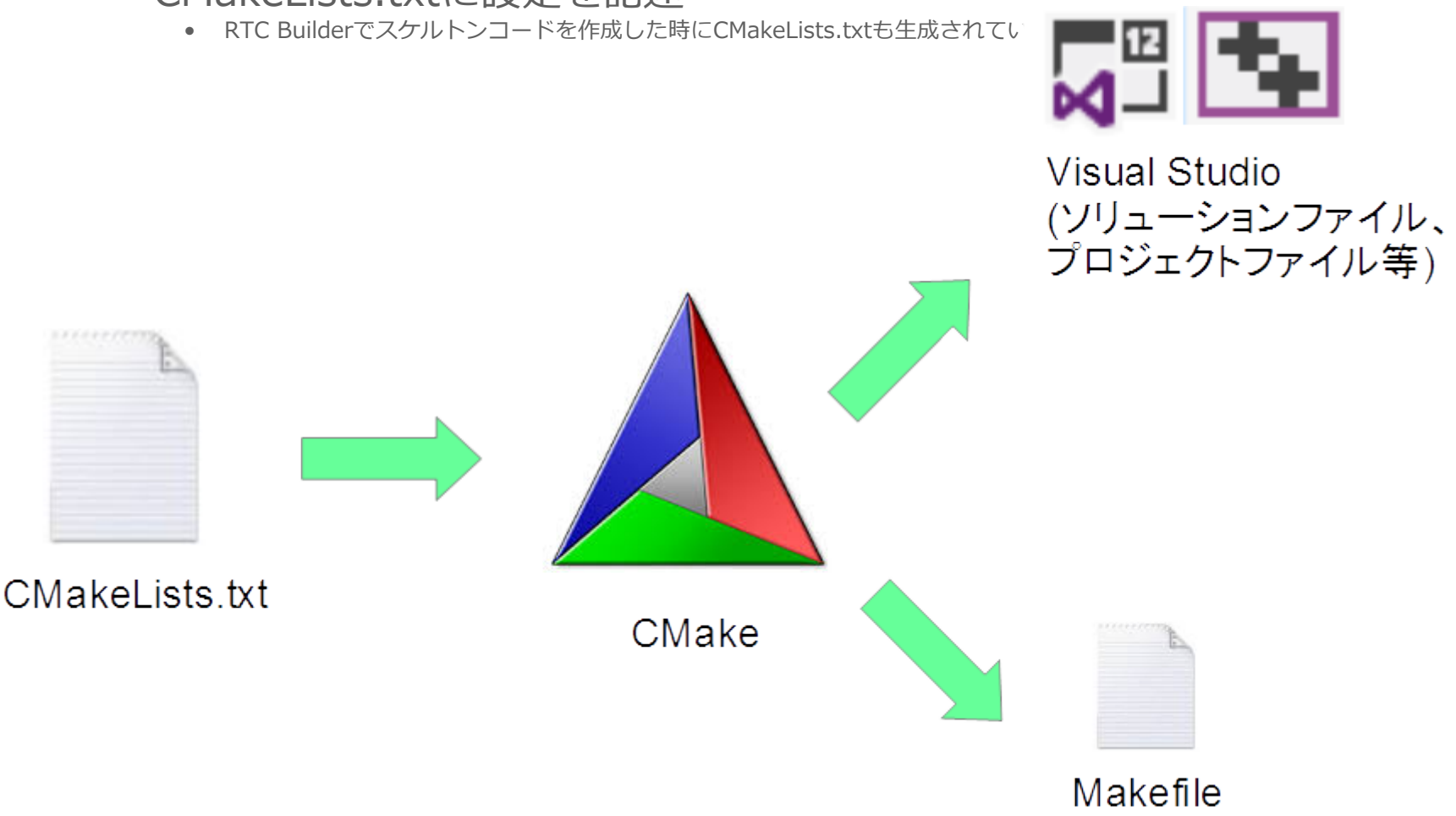
ソースコードの編集、RTCのビルド

手順

- 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
  - RobotController.hの編集
  - RobotController.cppの編集
- ビルド
  - Windows： Visual Studio
  - Ubuntu： Code::Blocks

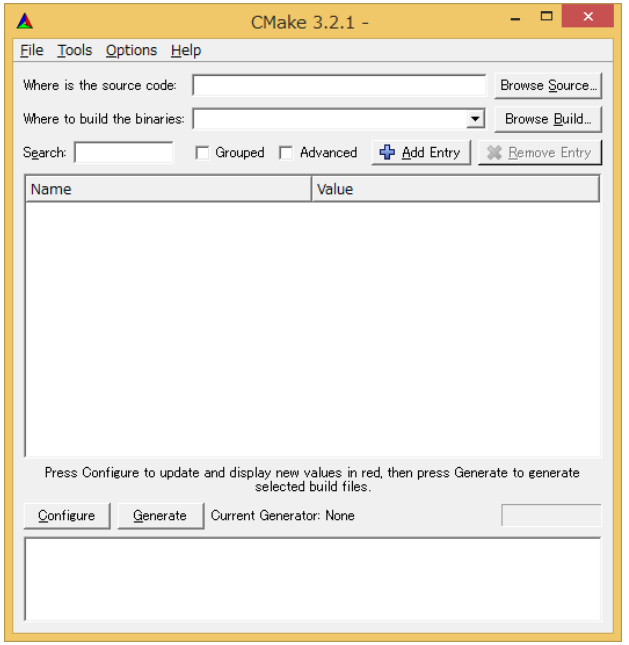
CMake

- 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



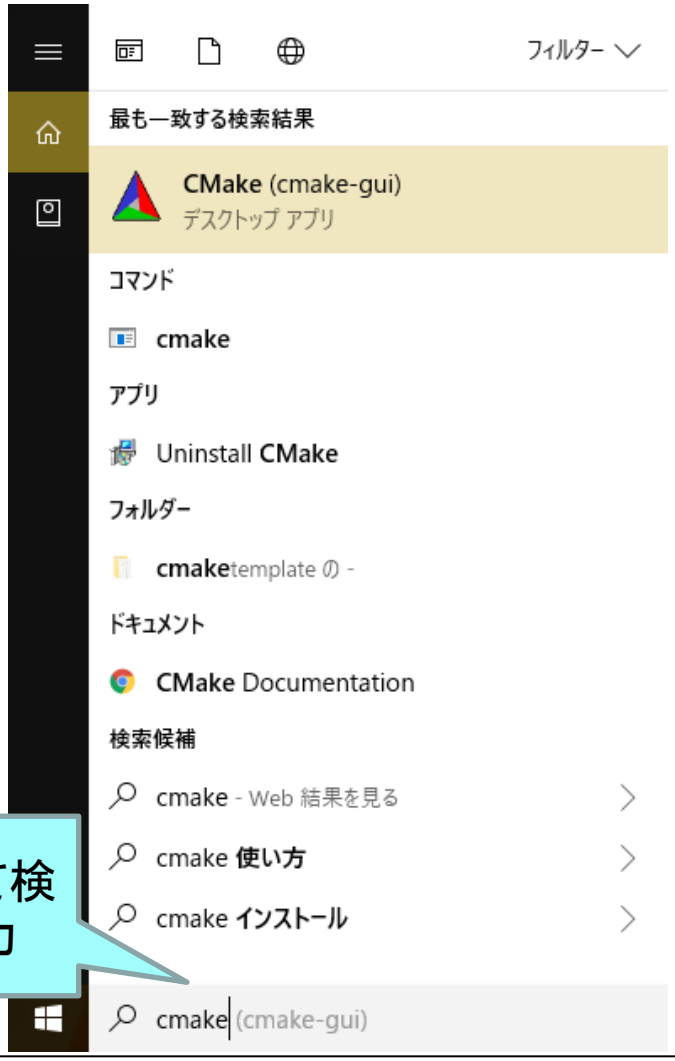
ビルドに必要なファイルの生成

- CMakeを使用する
  - 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



cmake-guiの起動

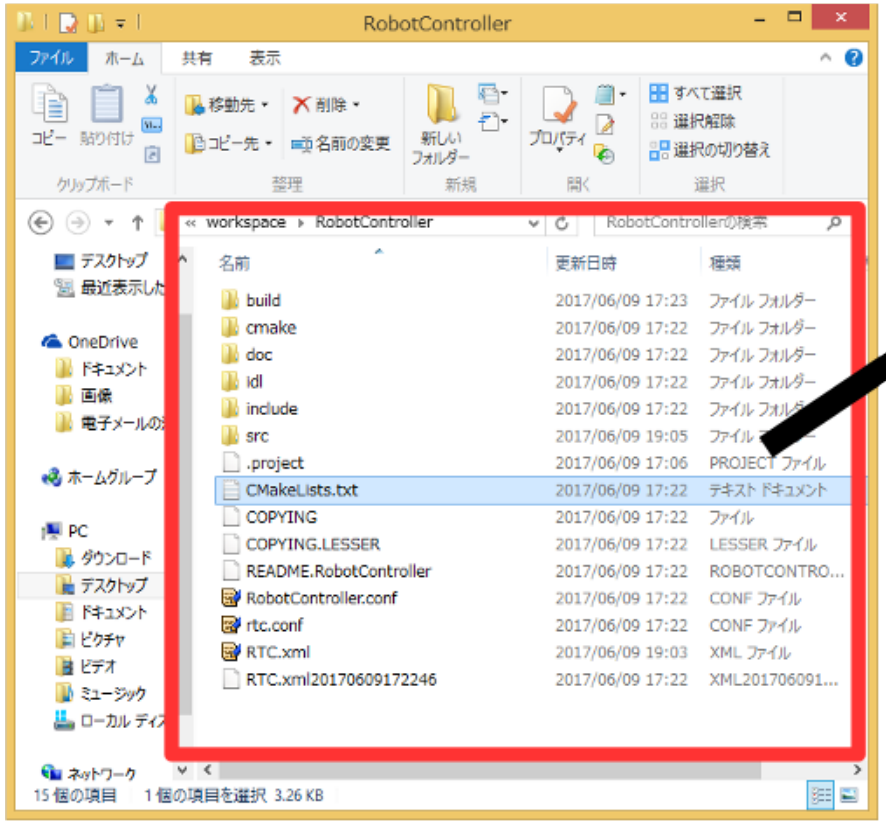
- Windows 10

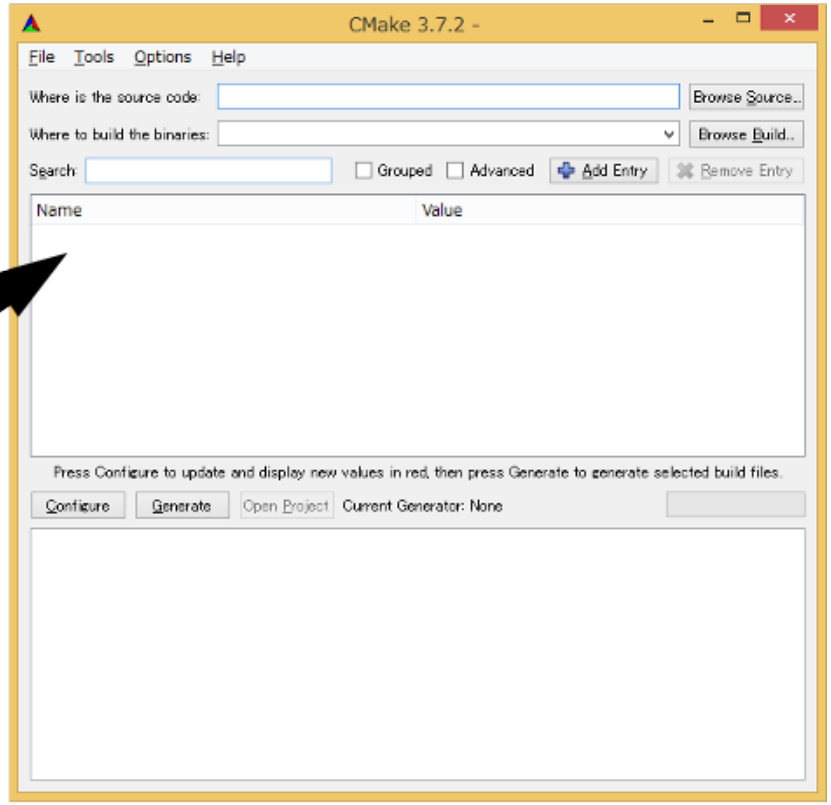


左下の「ここに入力して検索」に「cmake」と入力

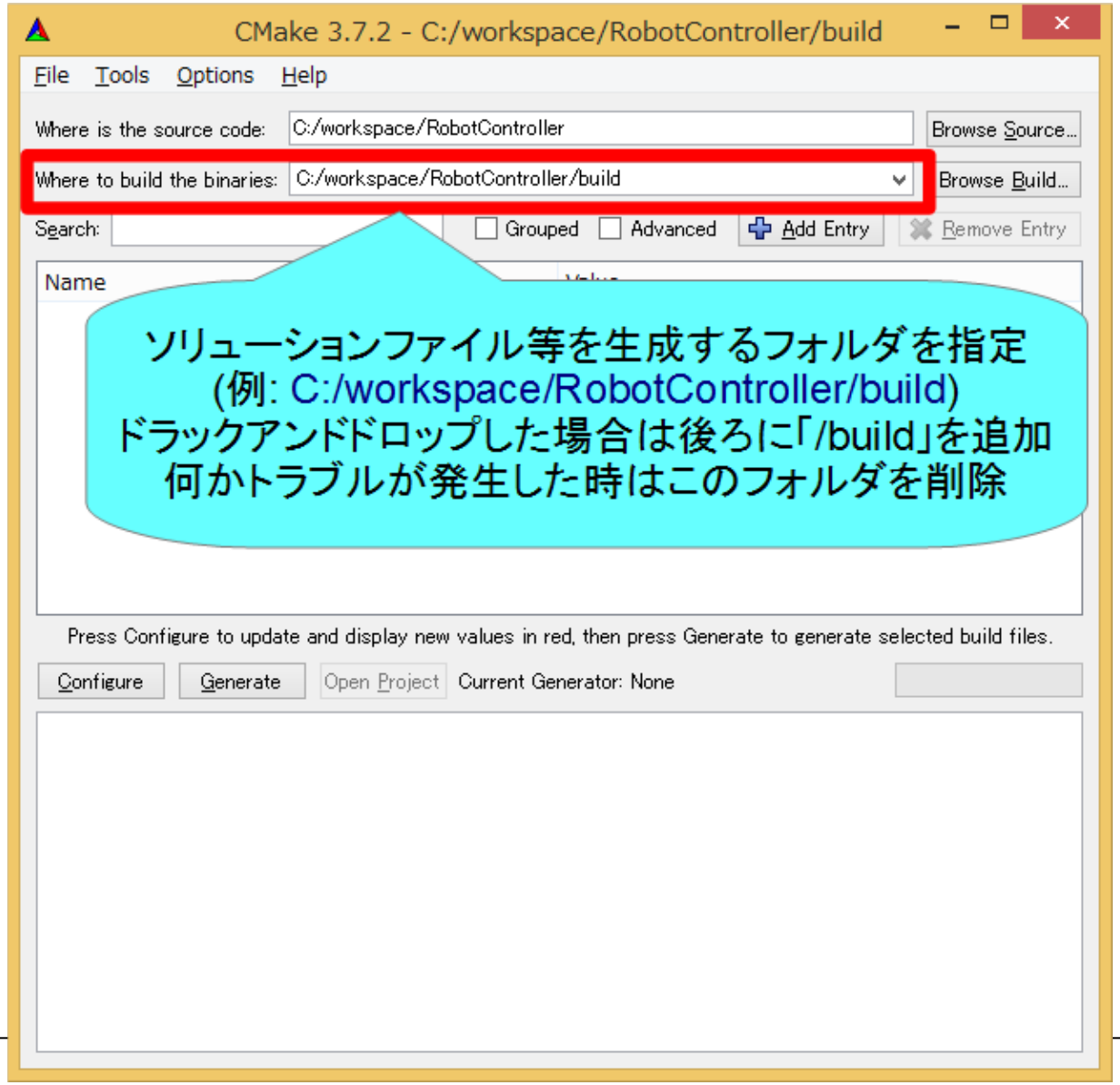
ビルドに必要なファイルの生成

- **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¥RobotController)

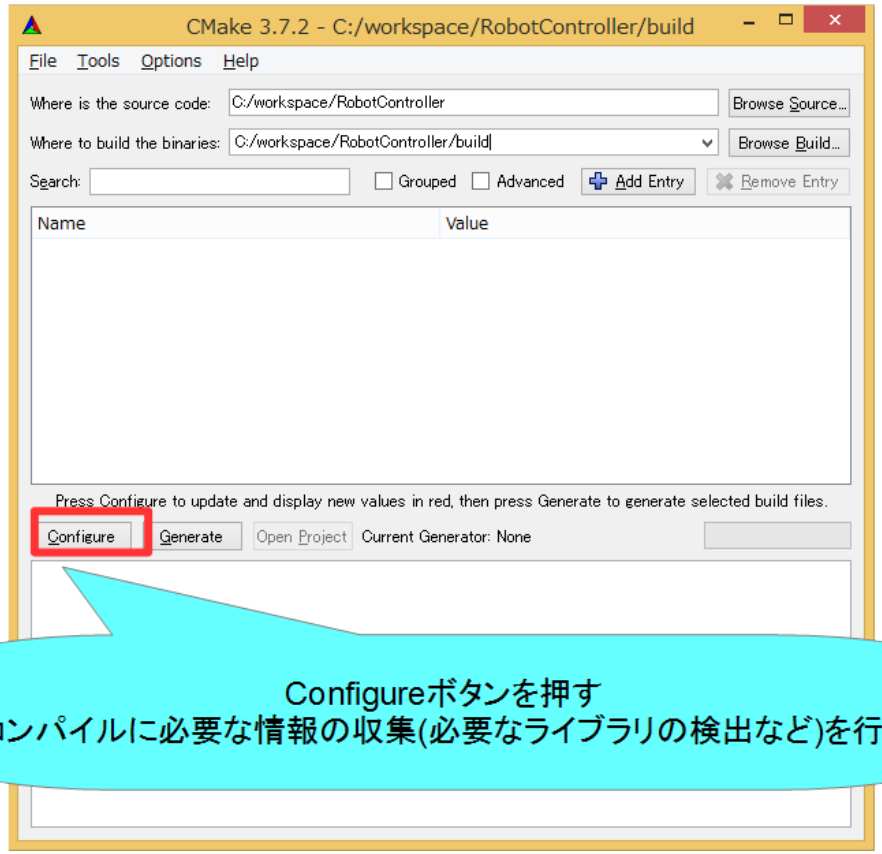




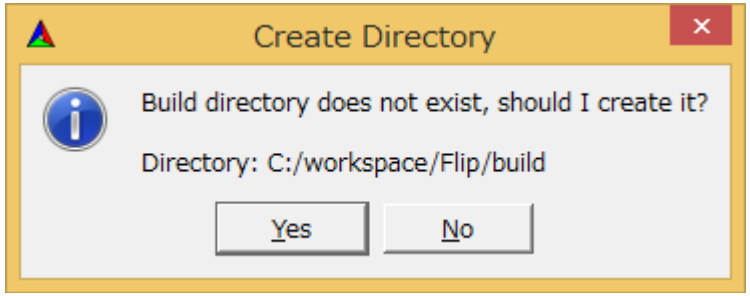
ビルドに必要なファイルの生成



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

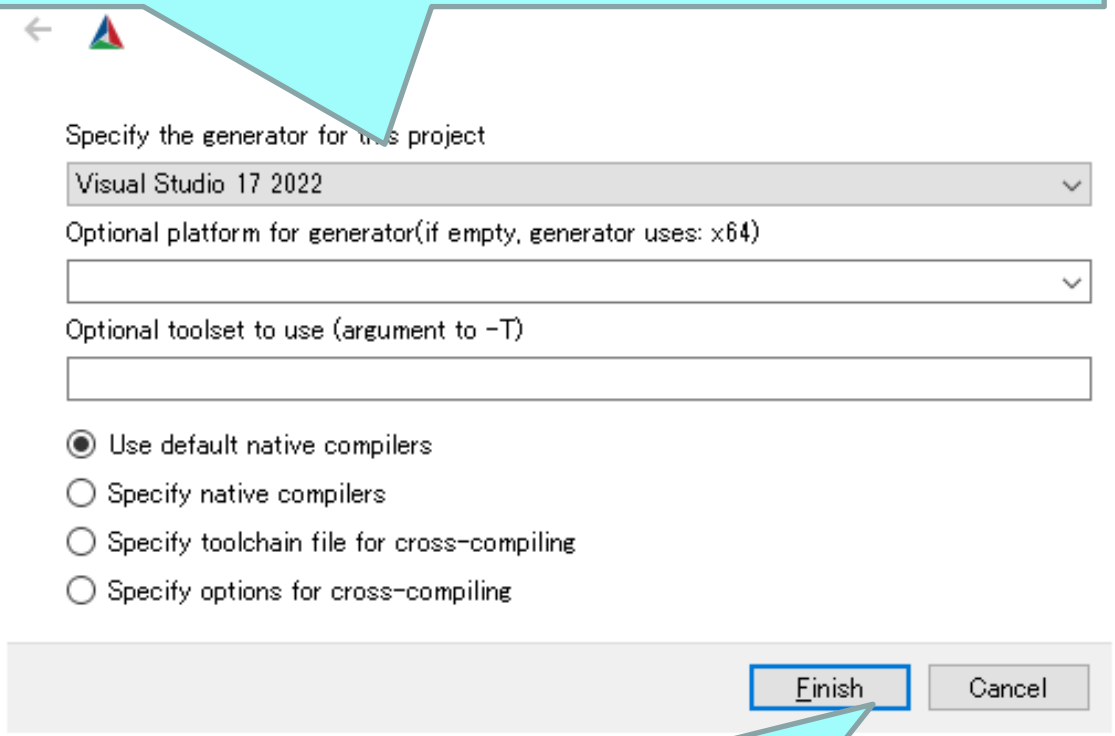


buildフォルダが存在しない場合は
作成するかどうかきかれるため「Yes」を選択

ビルド環境の設定

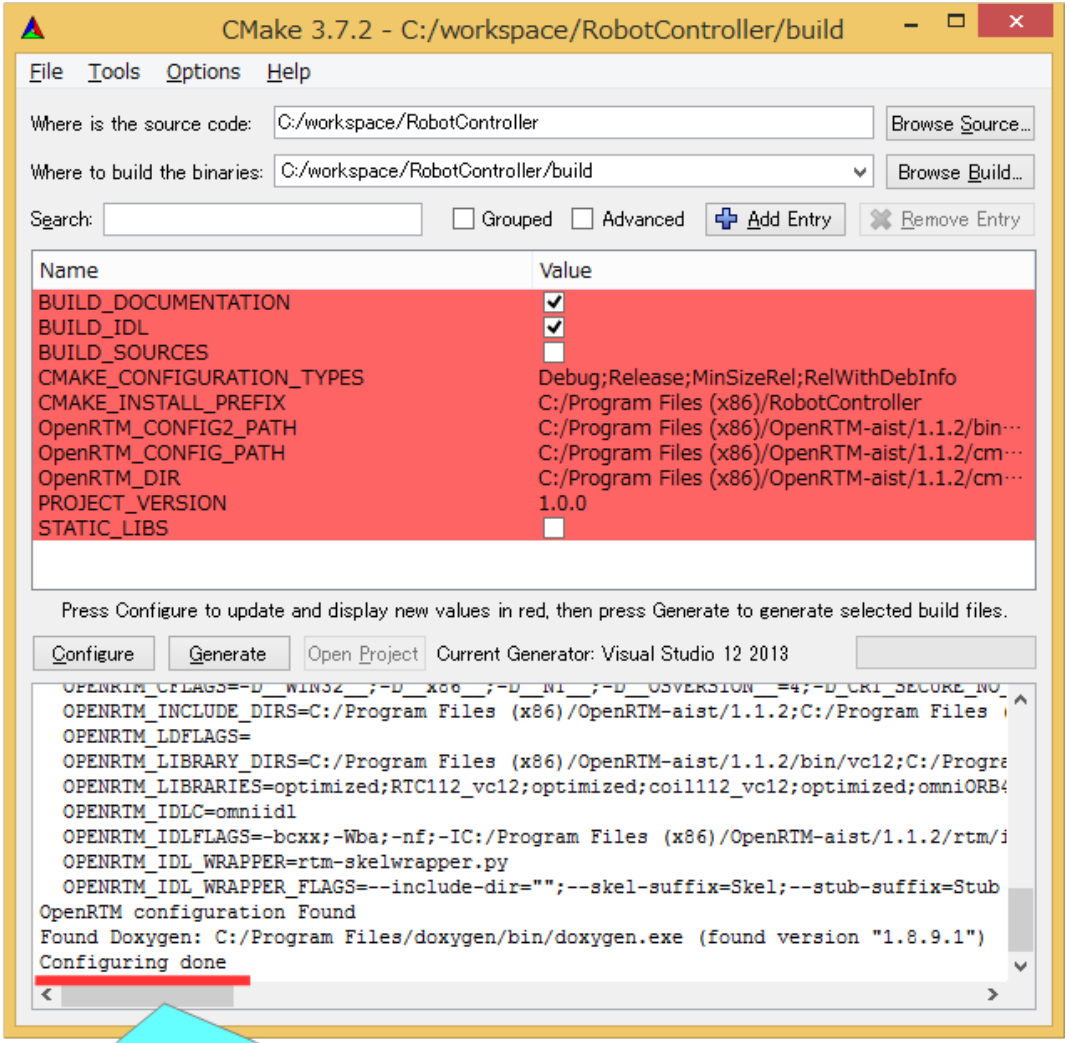
<u>Visual Studio 2022 → Visual Studio 17 2022</u>
Visual Studio 2019 → Visual Studio 16 2019
Code::Blocks → CodeBlocks-Unix Makefiles

※貸し出したPCでは「Visual Studio 17 2022」を指定



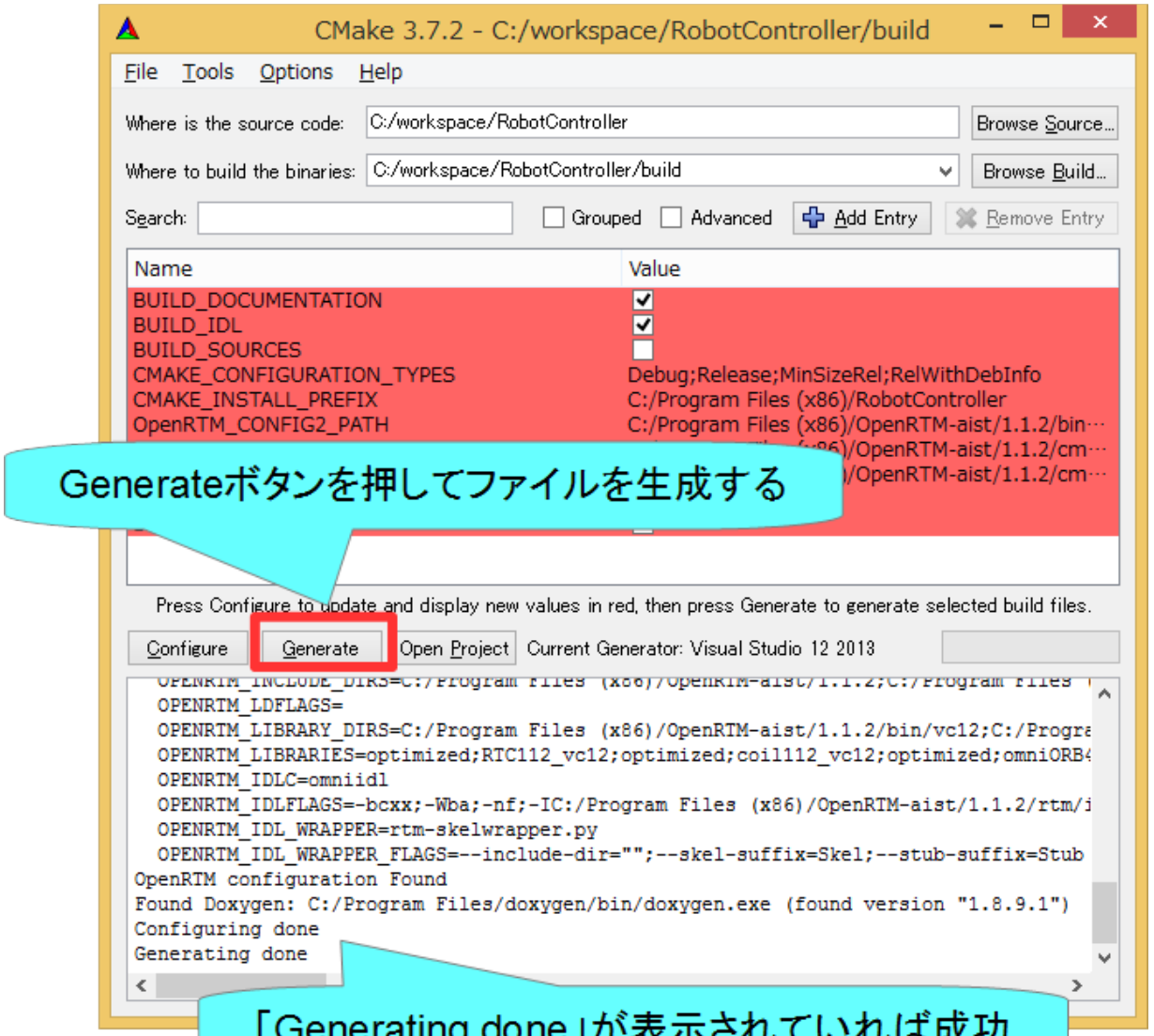
設定後、Finishボタンを押す

ビルドに必要なファイルの生成

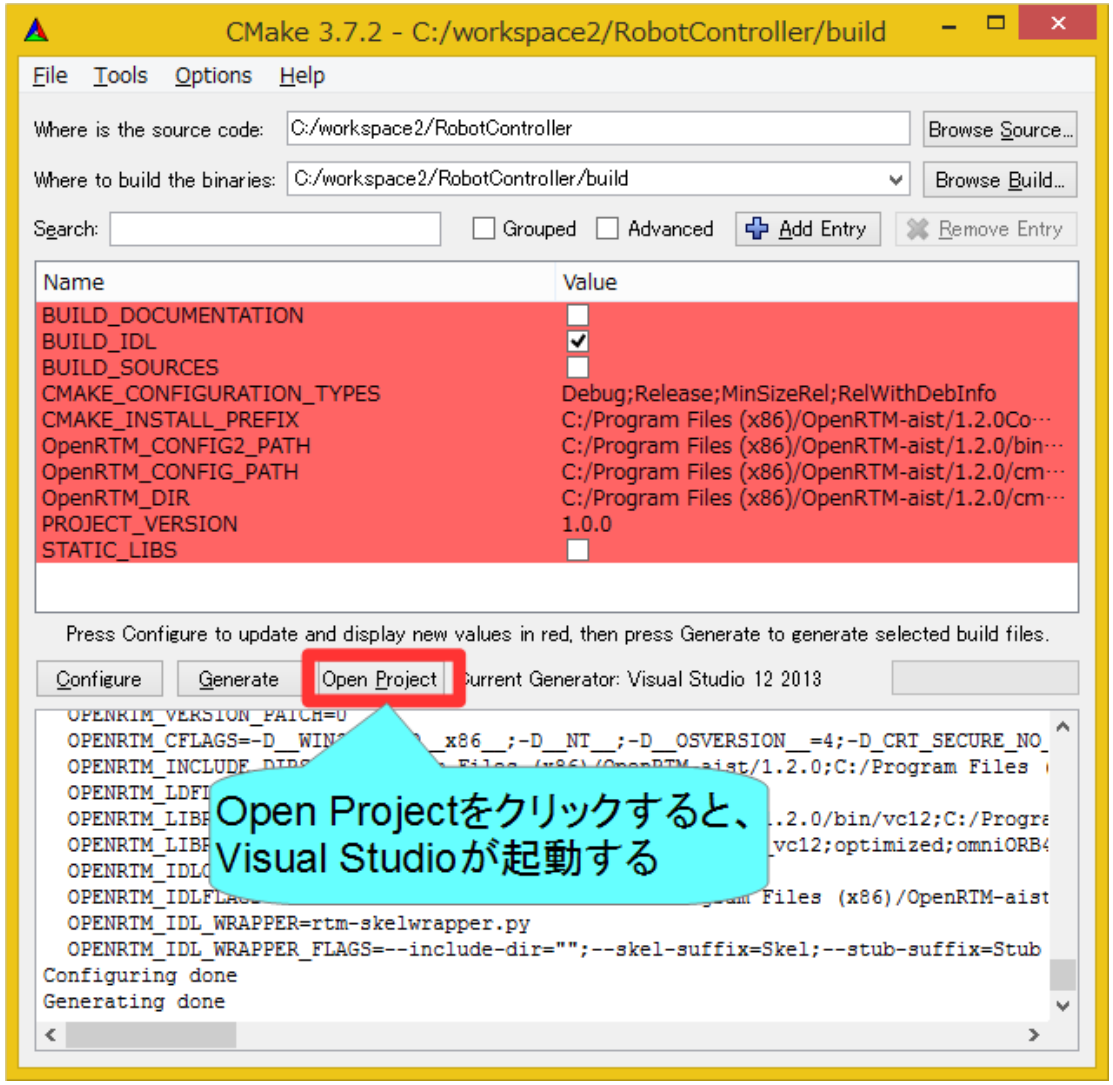


「Configure done」が表示されていれば成功

ビルドに必要なファイルの生成

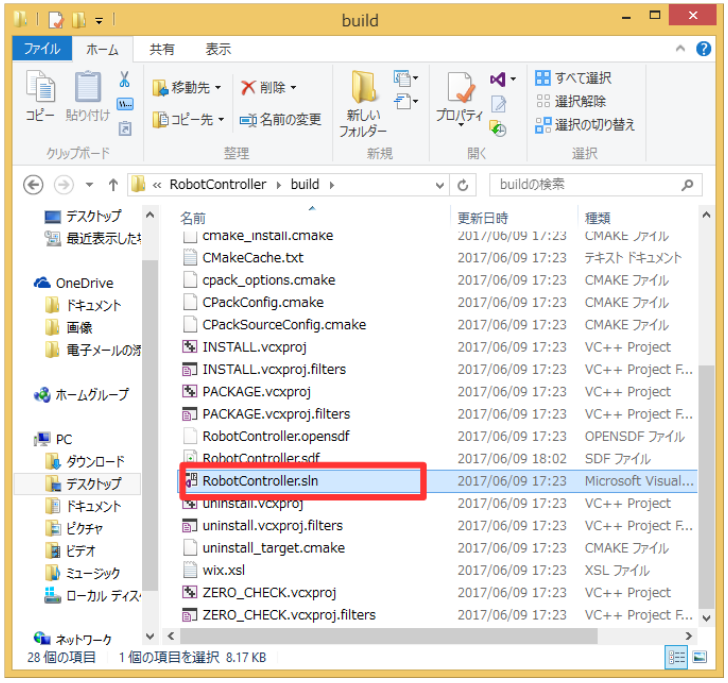


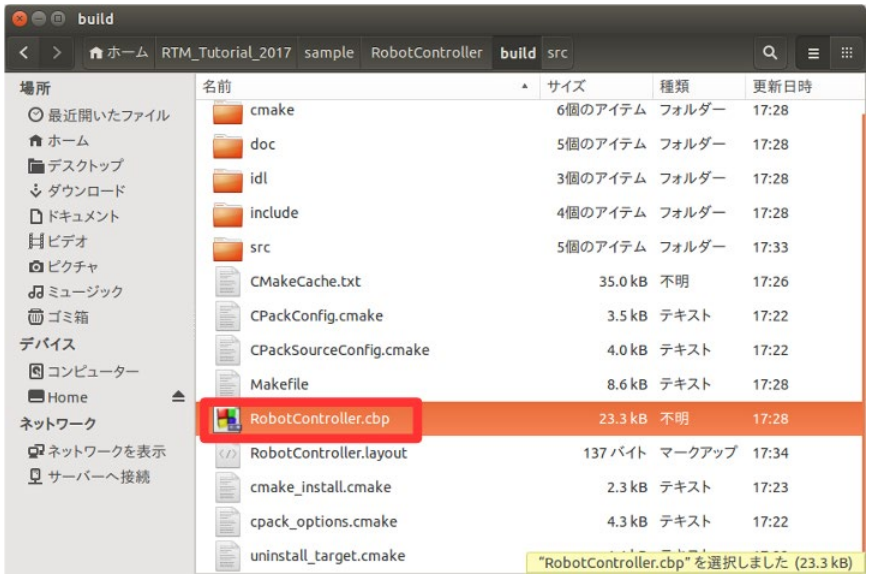
ソースコードの編集


CMake 3.7.2 - C:/workspace2/RobotController/build
File Tools Options Help
Where is the source code: C:/workspace2/RobotController
Browse Source...
Where to build the binaries: C:/workspace2/RobotController/build
Browse Build...
Search:
Grouped
Advanced
Add Entry
Remove Entry
Name
Value
BUILD_DOCUMENTATION
BUILD_IDL
BUILD_SOURCES
CMAKE_CONFIGURATION_TYPES
Debug;Release;MinSizeRel;RelWithDebInfo
CMAKE_INSTALL_PREFIX
C:/Program Files (x86)/OpenRTM-aist/1.2.0Co…
OpenRTM_CONFIG2_PATH
C:/Program Files (x86)/OpenRTM-aist/1.2.0/bin…
OpenRTM_CONFIG_PATH
C:/Program Files (x86)/OpenRTM-aist/1.2.0/cm…
OpenRTM_DIR
C:/Program Files (x86)/OpenRTM-aist/1.2.0/cm…
PROJECT_VERSION
1.0.0
STATIC_LIBS
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 12 2013
OPENRTM_VERSION_PATCH=0
OPENRTM_CFLAGS=-D__WIN
_x86__;-D__NT__;-D__OSVERSION__=4;-D_CRT_SECURE_NO_
OPENRTM_INCLUDE_DIR
OPENRTM_LDFI
OPENRTM_LIBR
OPENRTM_LIBR
OPENRTM_IDLC
OPENRTM_IDLFLA
OPENRTM_IDL_WRAPPER=rtm-skelwrapper.py
OPENRTM_IDL_WRAPPER_FLAGS=--include-dir="";--skel-suffix=Skel;--stub-suffix=Stub
Configuring done
Generating done
Open Projectをクリックすると、
Visual Studioが起動する

ソースコードの編集

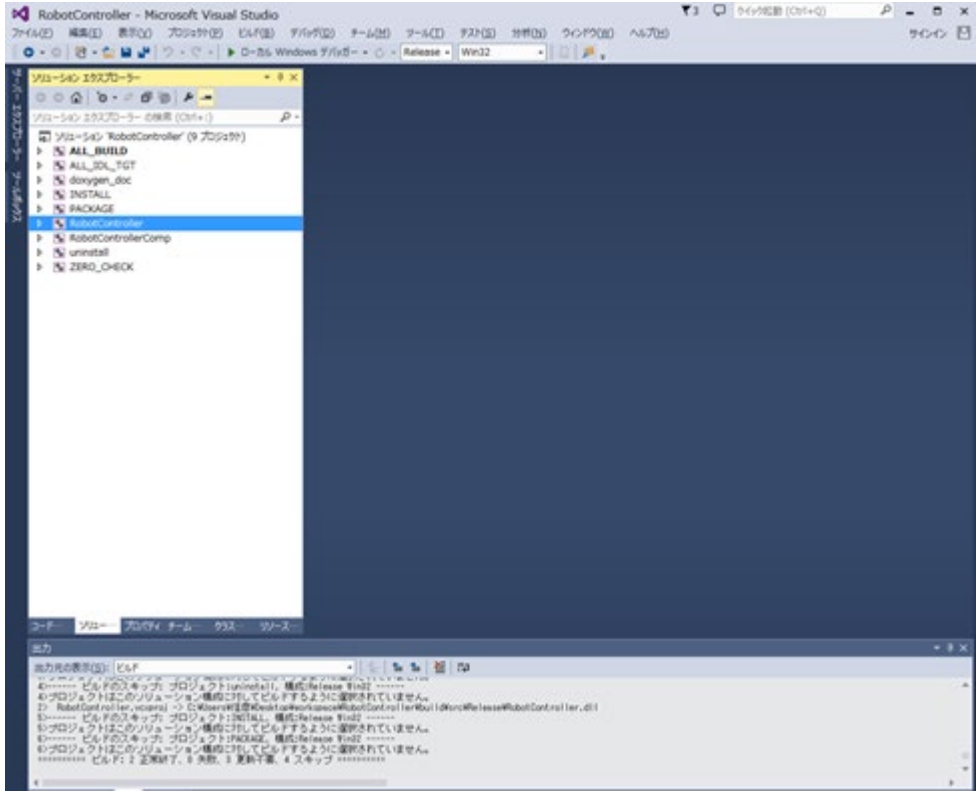
- CMake-guiのバージョンが古い場合は「Open Project」ボタンがないため、ファイルをダブルクリックして開く
  - Windows
    - buildフォルダの「RobotController.sln」をダブルクリックして開く
  - Ubuntu
    - buildフォルダの「RobotController.cbp」をダブルクリックして開く

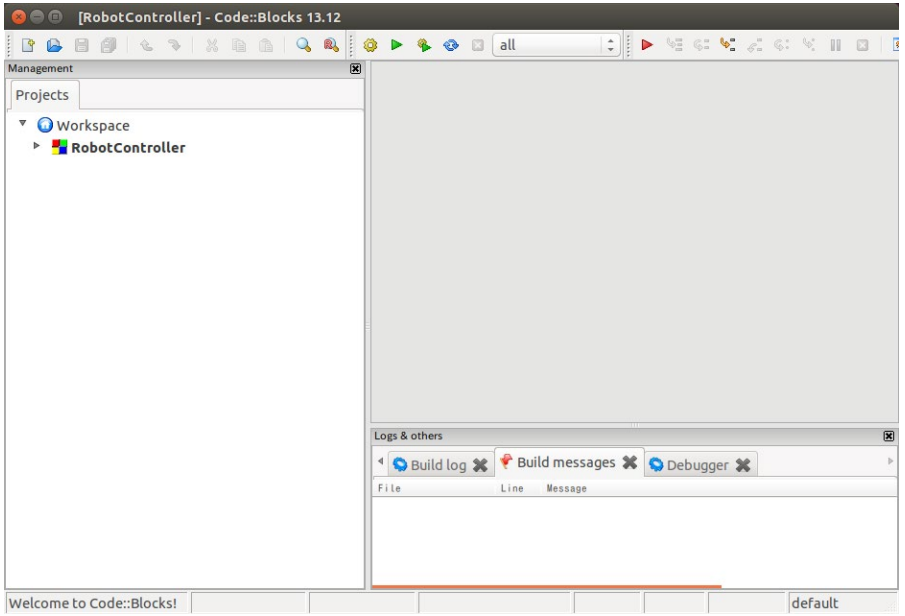




ソースコードの編集

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

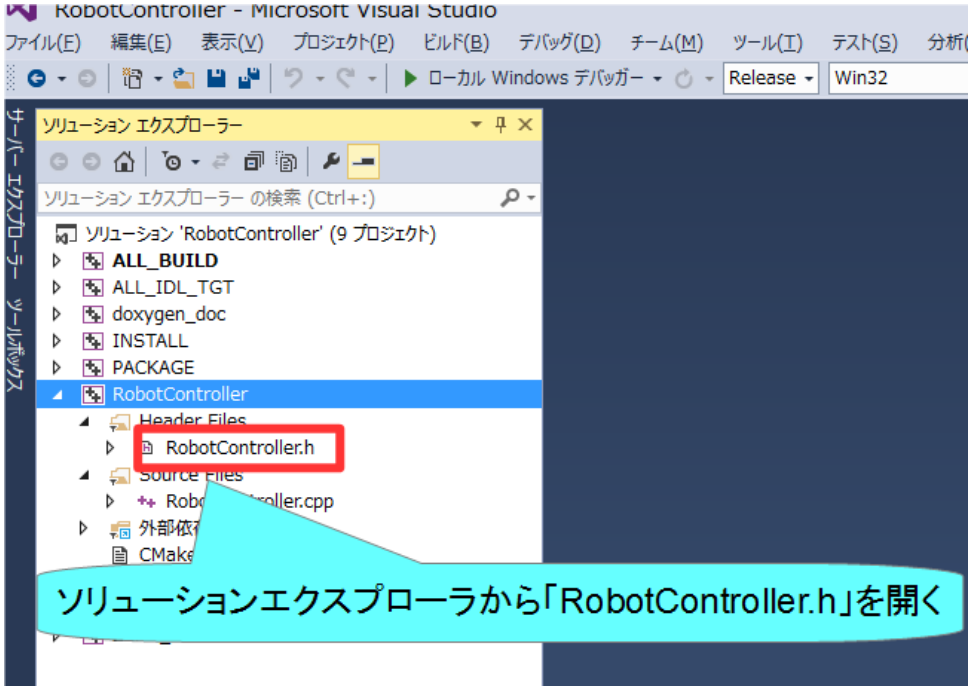




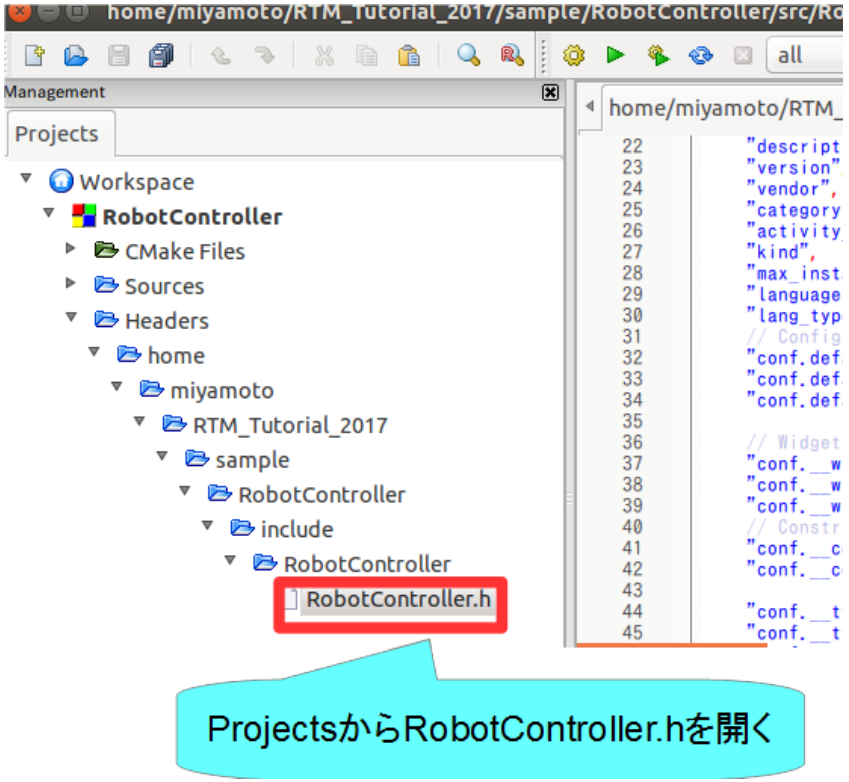
ソースコードの編集

- RobotController.hの編集

Visual Studio

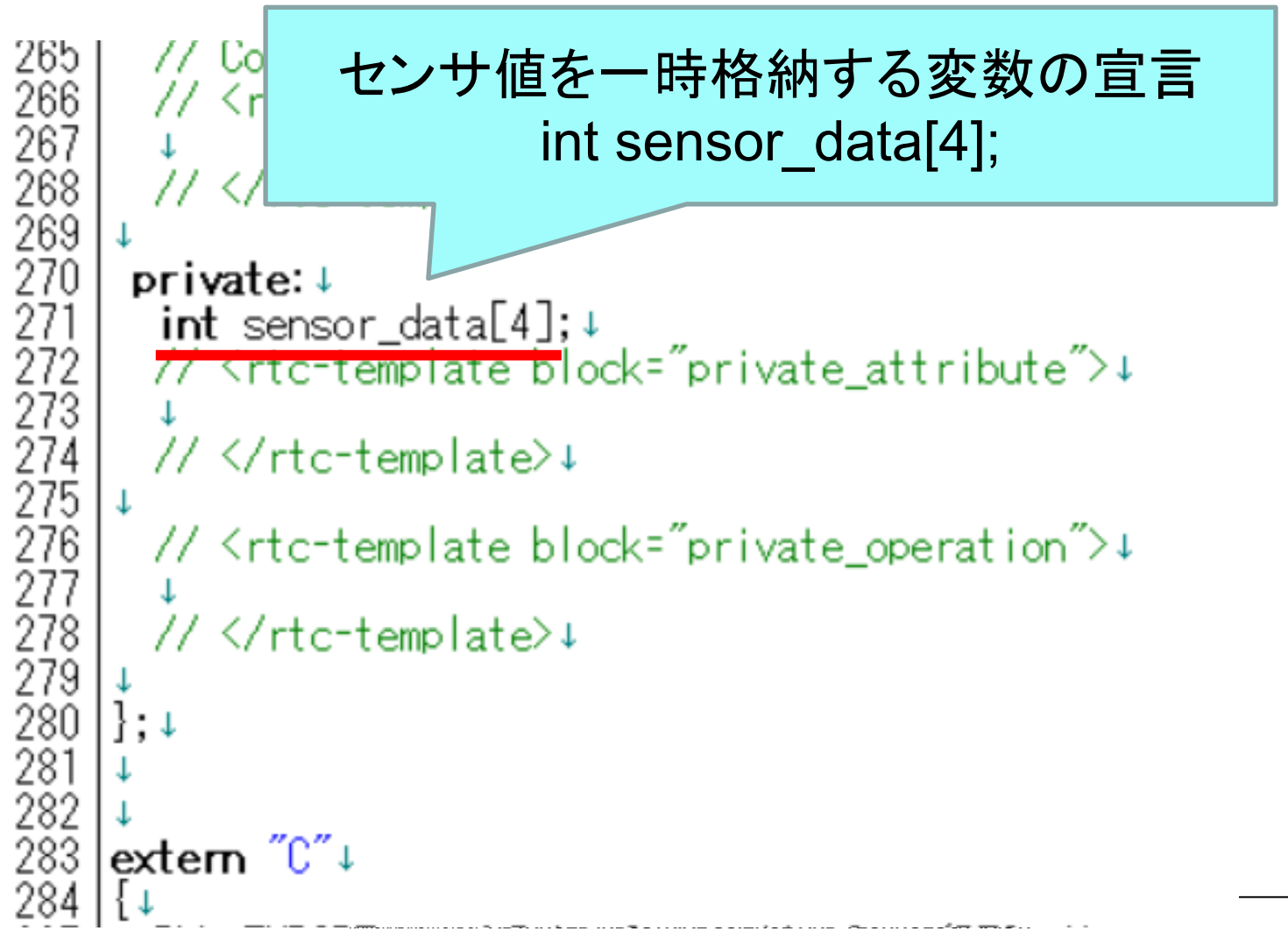


Code::Blocks



ソースコードの編集

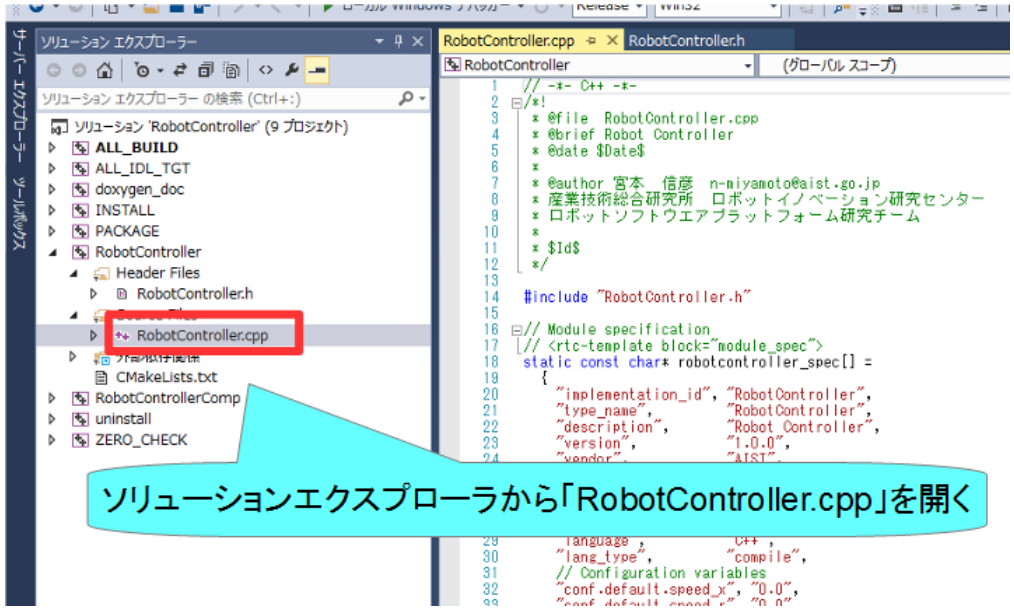
- RobotController.hの編集



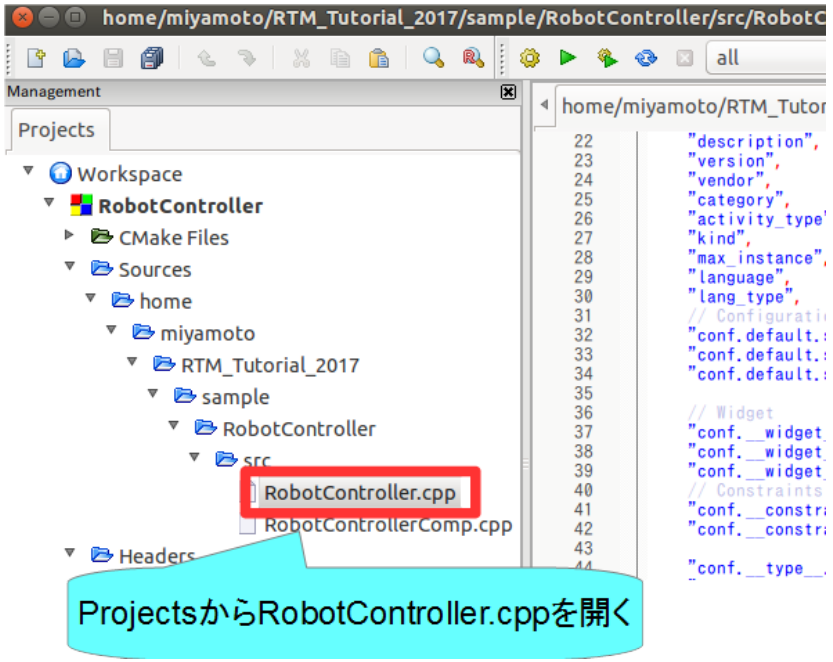
ソースコードの編集

- RobotController.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

- RobotController.cppの編集

```
125 RTC::ReturnCode_t RobotController::onActivated(RTC::UniqueId ec_id)
126 {
127
128     //センサ値初期化
129     for (int i = 0; i < 4; i++)
130     {
131         sensor_data[i] = 0;
132     }
133
134     return RTC::RTC_OK;
135 }
```

onActivateに追加

```
138 RTC::ReturnCode_t RobotController::onDeactivated(RTC::UniqueId ec_id)
139 {
140     //ロボットを停止する
141     m_out.data.vx = 0;
142     m_out.data.va = 0;
143     m_outOut.write();
144
145     return RTC::RTC_OK;
146 }
```

onDeactivateに追加

ソースコードの編集

- RobotController.cppの編集

```
157 RTC::ReturnCode_t RobotController::onExecute(RTC::UniqueId ec_id)
158 {
159     //入力データの存在確認
160     if (m_inIn.isNew())
161     {
162         //入力データ読み込み
163         m_inIn.read();
164         //この時点で入力データがm_inに格納される
165         for (int i = 0; i < m_in.data.length(); i++)
166         {
167             //入力データを別変数に格納
168             if (i < 4)
169             {
170                 sensor_data[i] = m_in.data[i];
171             }
172         }
173     }
174
175     //前進するときのみ停止するかを判定
176     if (m_speed_x > 0)
177     {
178         for (int i = 0; i < 4; i++)
179         {
180             //センサ値が設定値以上か判定
181             if (sensor_data[i] > m_stop_d)
182             {
183                 //センサ値が設定値以上の場合は停止
184                 m_out.data.vx = 0;
185                 m_out.data.va = 0;
186                 m_outOut.write();
187                 return RTC::RTC_OK;
188             }
189         }
190     }
191     //設定値以上の値のセンサが無い場合はコンフィギュレーションパラメータの値で操作
192     m_out.data.vx = m_speed_x;
193     m_out.data.va = m_speed_r;
194     m_outOut.write();
195     return RTC::RTC_OK;
196 }
```

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

```
//入力データの存在確認
if (m_inIn.isNew())
{
    //入力データ読み込み
    m_inIn.read();
    //この時点で入力データがm_inに格納される
    for (int i = 0; i < m_in.data.length(); i++)
    {
        //入力データを別変数
        if (i < 4)
        {
            sensor_data[i] = m_in.data[i];
        }
    }
}
```

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_inにデータが格納される

補足：TimedShortSeq型は配列のように
複数のデータを保持している。

ソースコードの編集

- データを書き込む手順

```
175     //前進するときのみ停止するかを判定
176     if (m_speed_x > 0)
177     {
178         for (int i = 0; i < 4; i++)
179         {
180             //センサ値が設定値以上か判定
181             if (sensor_data[i] > m_stop_d)
182             {
183                 //センサ値が設定値以上の場合は停止
184
185
186
187
188             }
189         }
190     }
191     //設定値以上の値のセンサが無い場合はコンフィギュレーションパラメータの値で操作
192     m_out.data.vx = m_speed_x;
193     m_out.data.va = m_speed_r;
194     m_outOut.write();
```

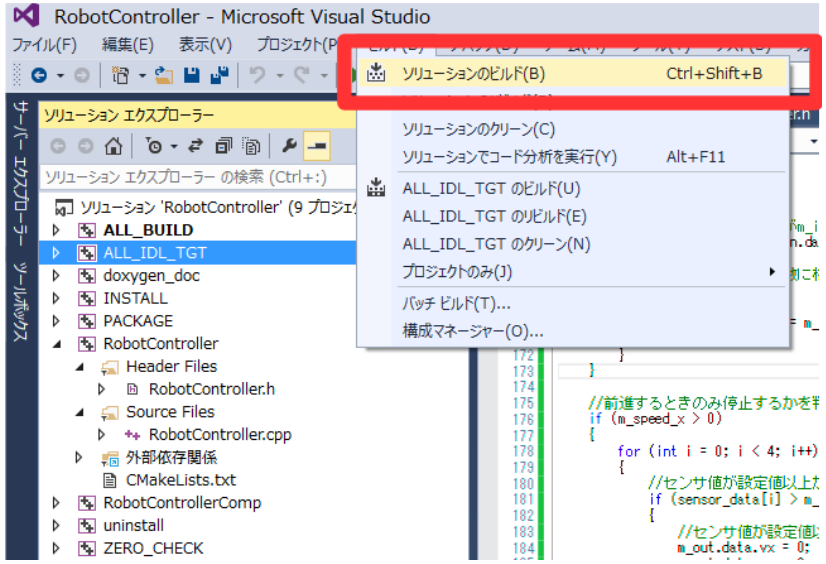
変数m_outにデータを格納する
TimedVelocity2D型のため、vxに直進速度、
vaに回転速度を格納する。

write関数でデータの書き込み

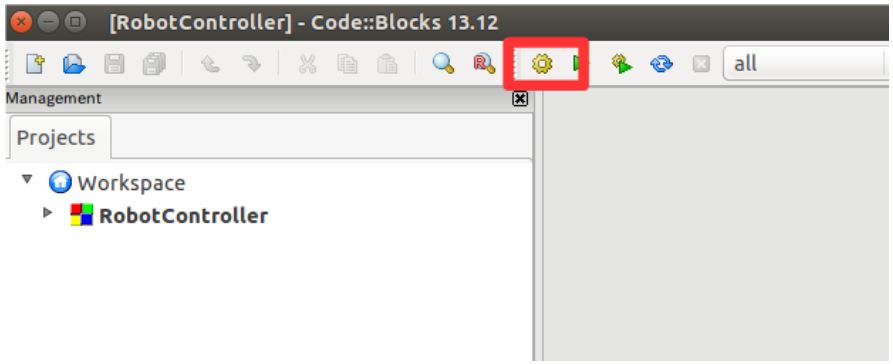
補足:コンフィギュレーションパラメータは変更すると
対応する変数(m_speed_x、m_speed_r、m_stop_d)
に値が格納される

ソースコードのコンパイル

Visual Studio



Code::Blocks

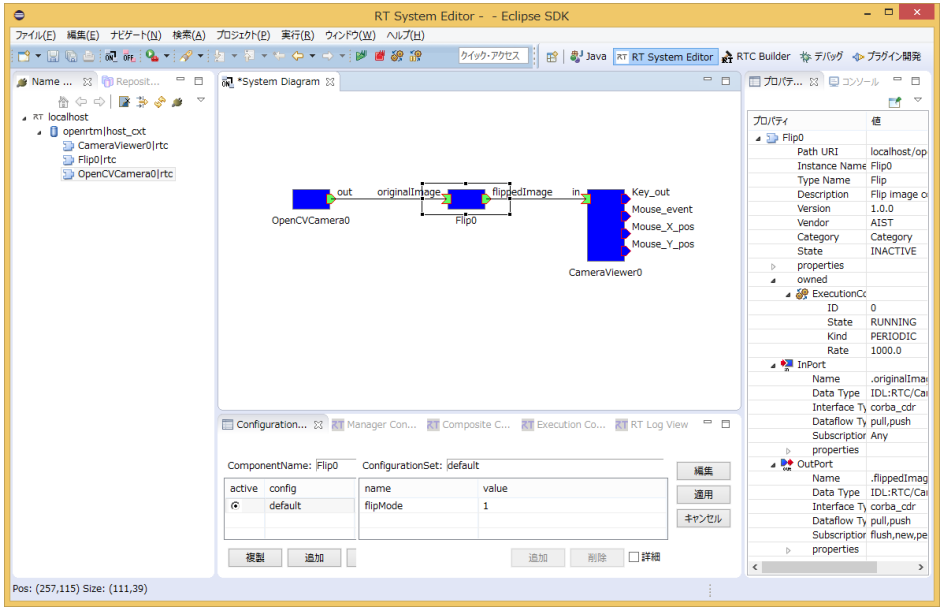


- 成功した場合、実行ファイルが生成される
  - Windows
    - **build¥src**フォルダの**Release**(もしくは**Debug**)フォルダ内にRobotControllerComp.exeが生成される
  - Ubuntu
    - **build/src**フォルダにRobotControllerCompが生成される

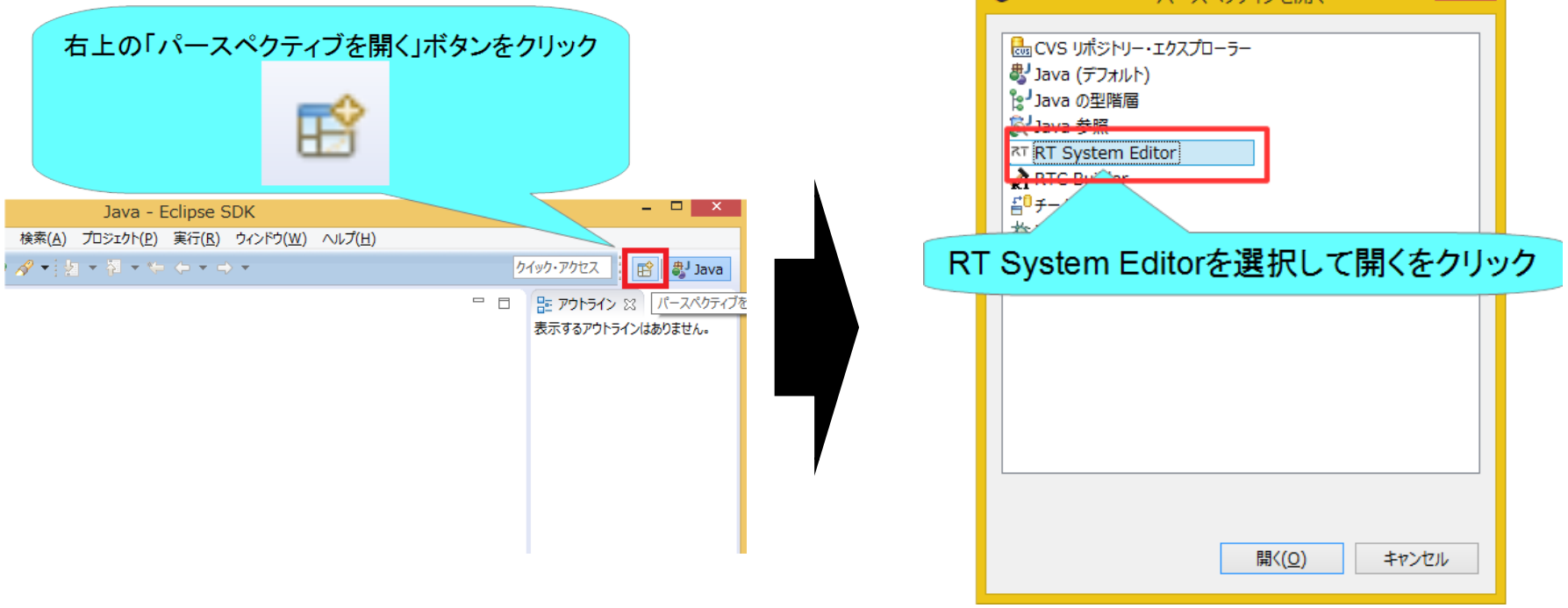
システム構築支援ツール
RT System Editorについて

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



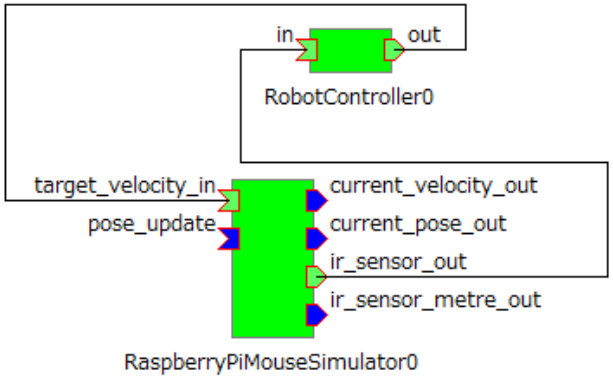
RT System Editorの起動



RT System Editorの画面構成

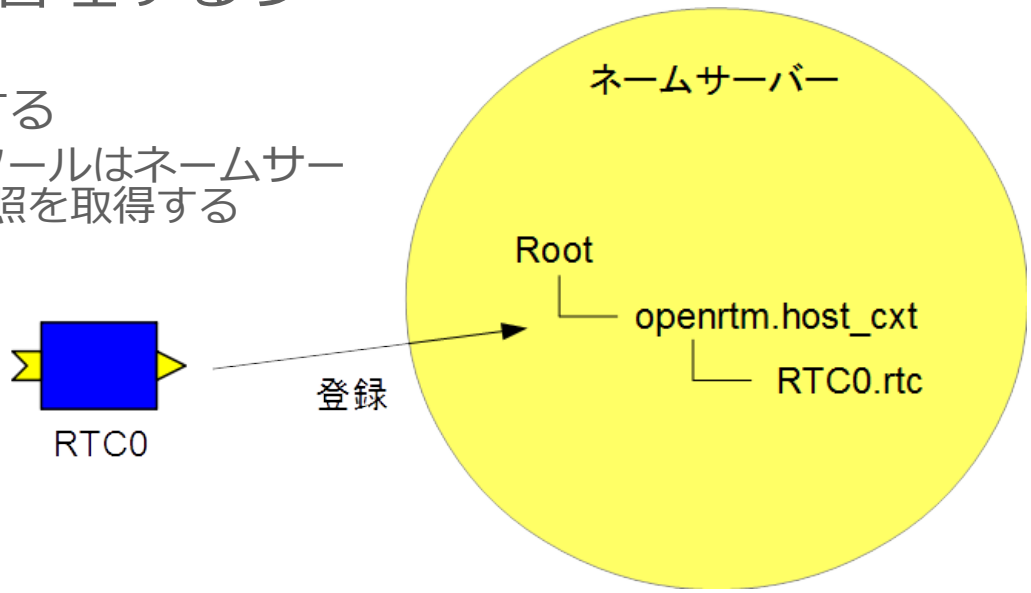
RobotControllerコンポーネントの動作確認

- シミュレータコンポーネントと接続してシミュレータ上のロボットを操作するRTシステムを作成する
  - ネームサーバーを起動する
  - RaspberryPiMouseSimulatorコンポーネントを起動する
    - 展開したZIPファイルのEXEフォルダ内「 RaspberryPiMouseSimulatorComp.exe 」をダブルクリック
  - RobotControllerコンポーネント起動
  - RaspberryPiMouseSimulatorコンポーネントとRobotControllerコンポーネントを接続して「All Activate」を行う

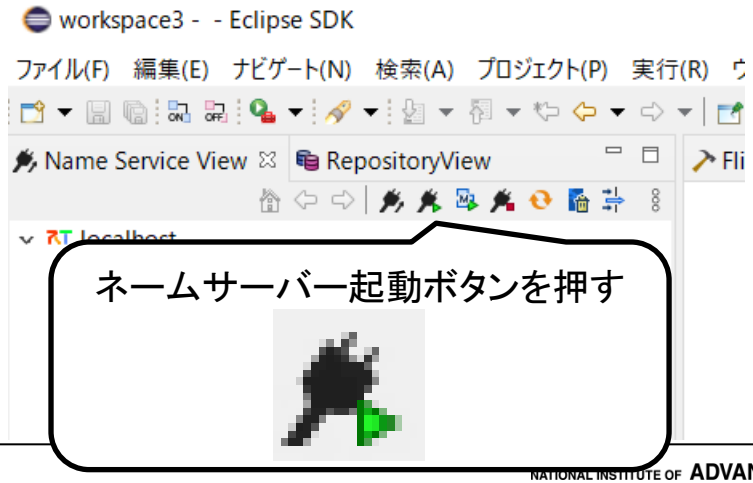


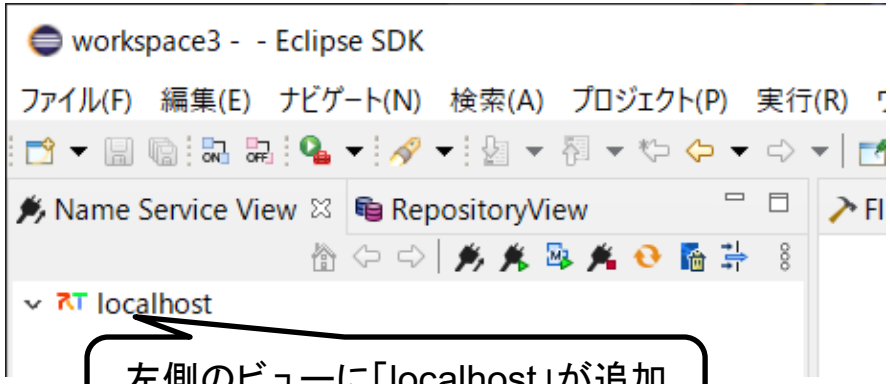
ネームサーバーの起動

- 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

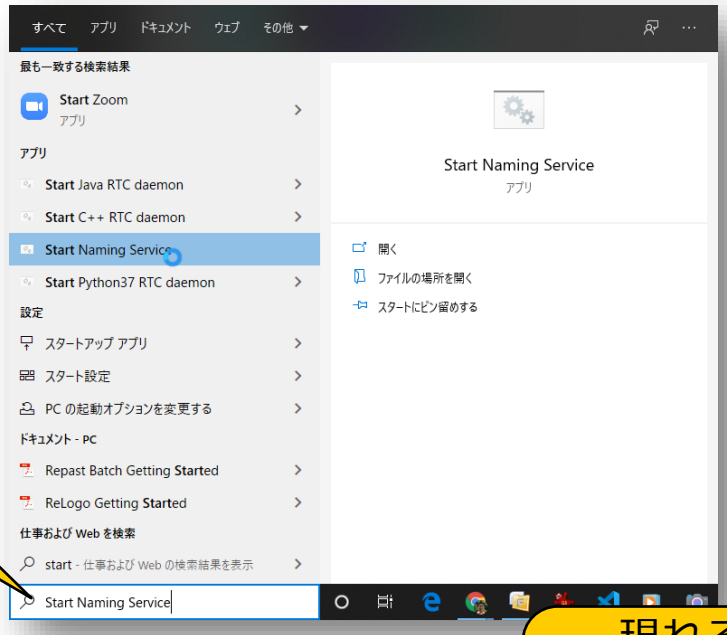




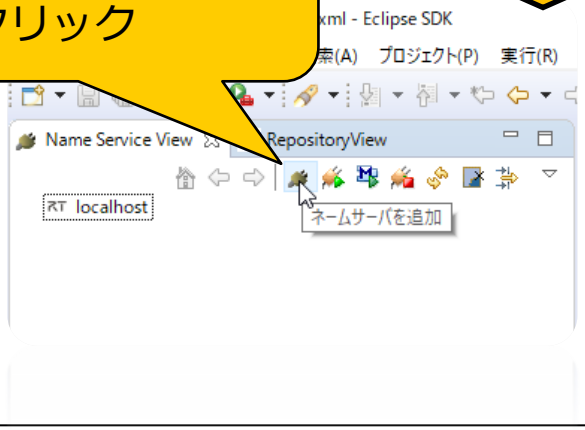
（参考）ネームサーバー起動・接続を個別に行う

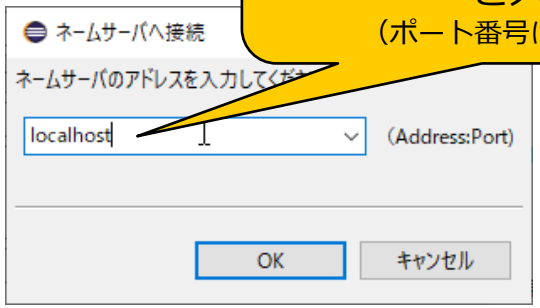
- Windows 10

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



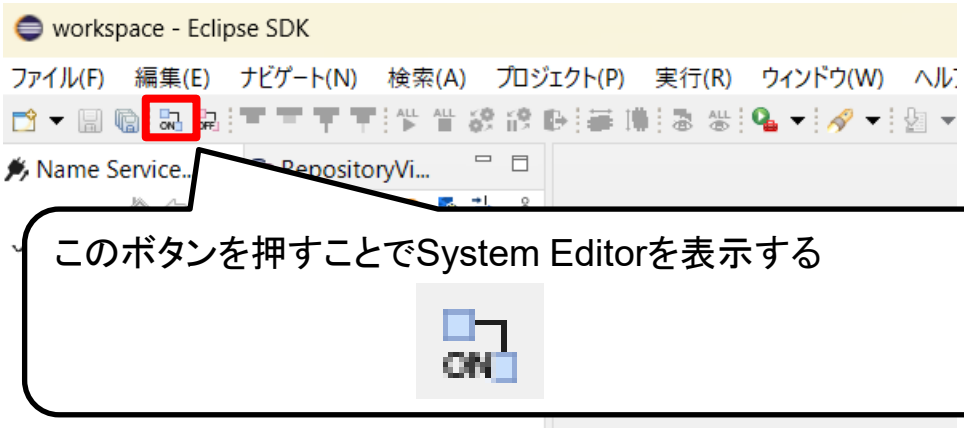


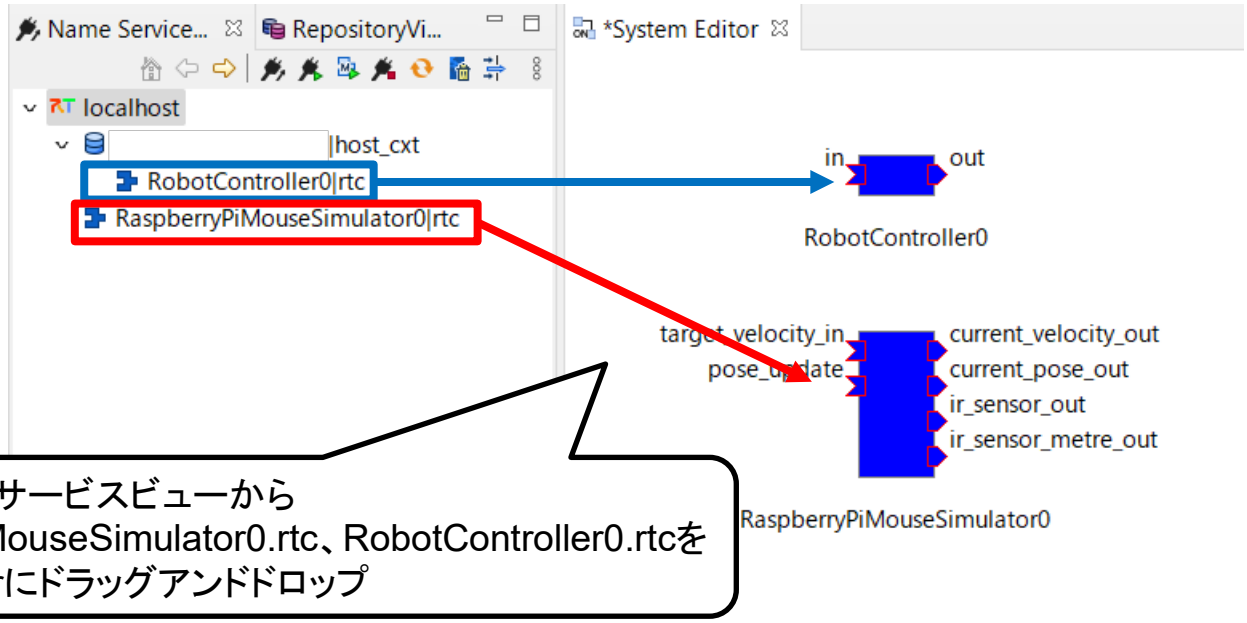
現れるダイアログに
「ホスト名：ポート番号」
を入力
（ポート番号は省略可能）

RobotControllerコンポーネントの動作確認

- シミュレータコンポーネントと接続してシミュレータ上のロボットを操作するRTシステムを作成する
  - ネームサーバーを起動する
  - RaspberryPiMouseSimulatorコンポーネントを起動する
    - RTMTutorialフォルダのEXEフォルダ内「 RaspberryPiMouseSimulatorComp.exe 」をダブルクリック
  - RobotControllerコンポーネント起動
    - **build¥src**フォルダの**Release**(もしくは**Debug**)フォルダ内にRobotControllerComp.exeが生成されているためこれを起動する
  - RobotControllerコンポーネント、RasPiMouseSimulatorコンポーネントを接続して「All Activate」を行う

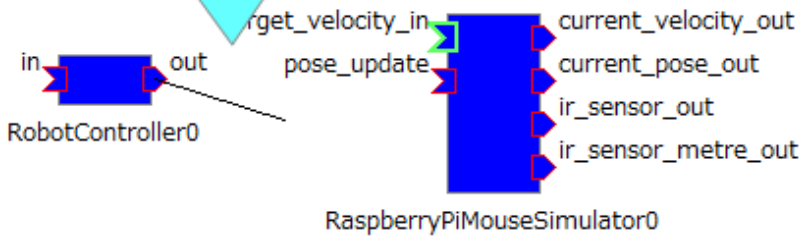
データポートの接続

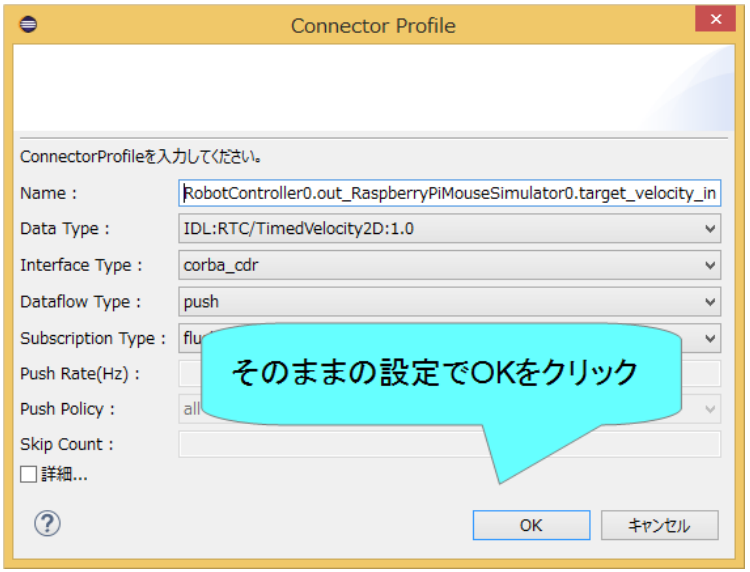

workspace - Eclipse SDK
ファイル(F)
編集(E)
ナビゲート(N)
検索(A)
プロジェクト(P)
実行(R)
ウィンドウ(W)
Name Service...
RepositoryVi...
このボタンを押すことでSystem Editorを表示する
ON

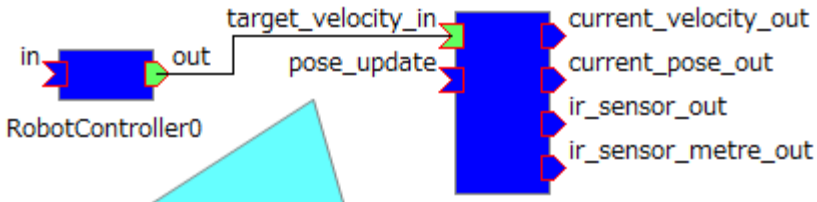

Name Service...
RepositoryVi...
*System Editor
localhost
host_cxt
RobotController0.rtc
RaspberryPiMouseSimulator0.rtc
in
out
RobotController0
target_velocity_in
pose_update
current_velocity_out
current_pose_out
ir_sensor_out
ir_sensor_metre_out
RaspberryPiMouseSimulator0
左側のネームサービスビューから
RaspberryPiMouseSimulator0.rtc、RobotController0.rtcを
System Editorにドラッグアンドドロップ

データポートの接続

RobotController0の「out」を選択して、
RaspberryPiMouseSimulator0の
「target_velocity_in」にドラッグアンドドロップ

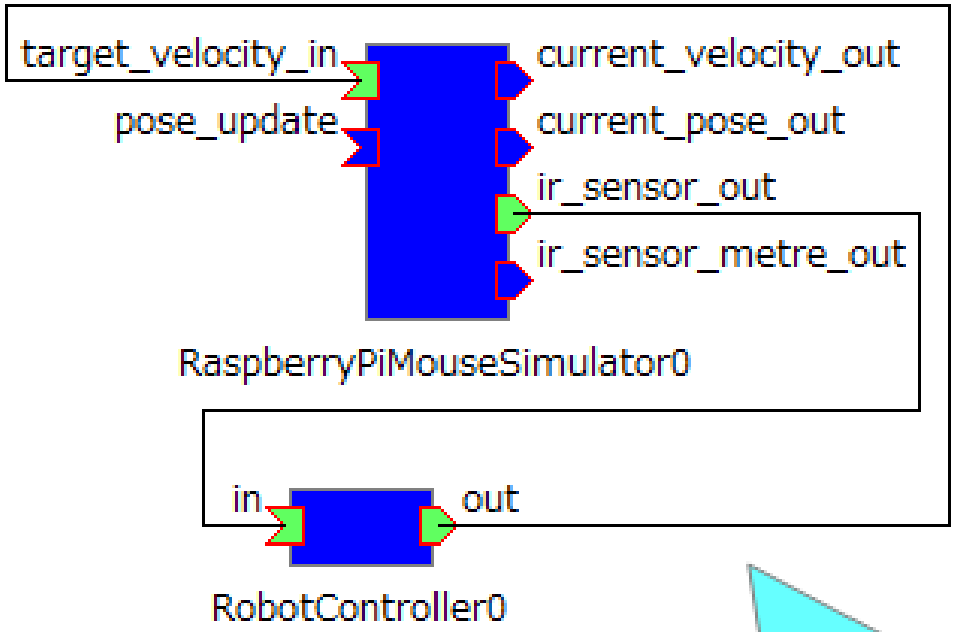






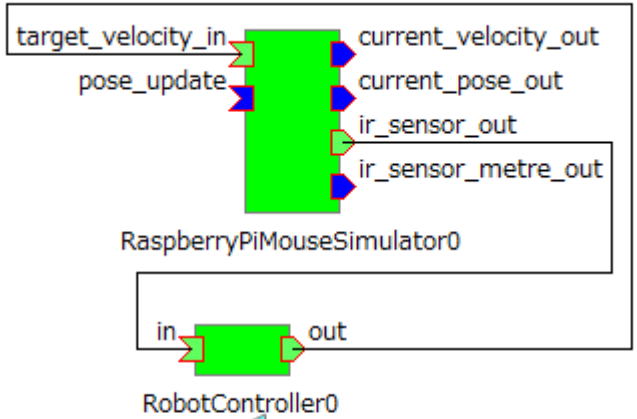
1. ポートの間に線が表示される
2. InPort、OutPortが緑色で表示される

データポートの接続



RaspberryPiMouseSimulator0の「ir_sensor_out」と
RobotController0の「in」を接続する

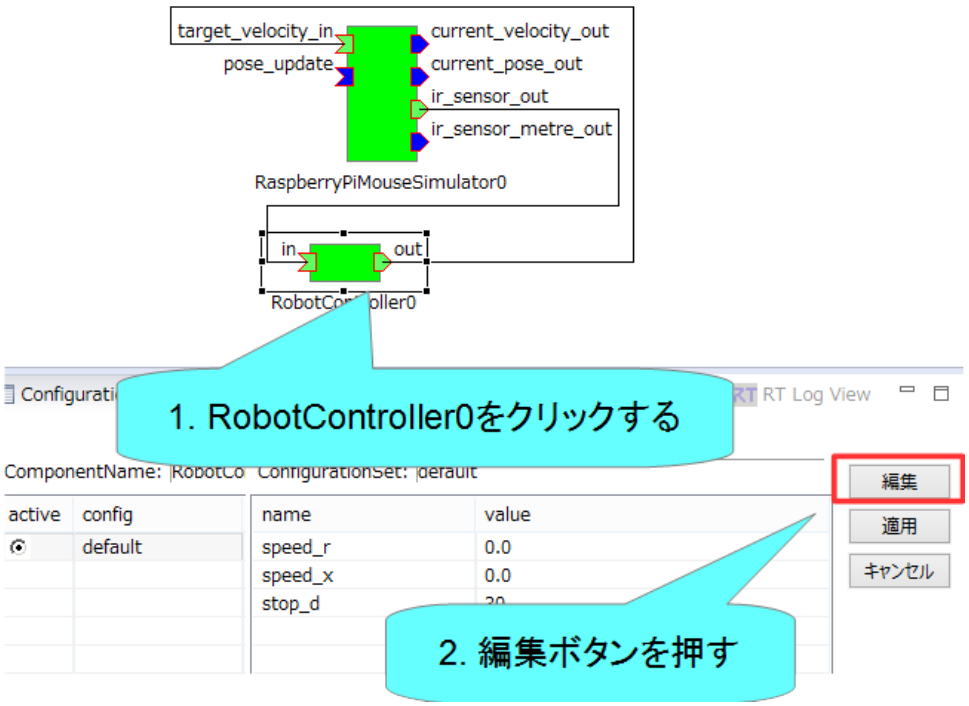
アクティブ化

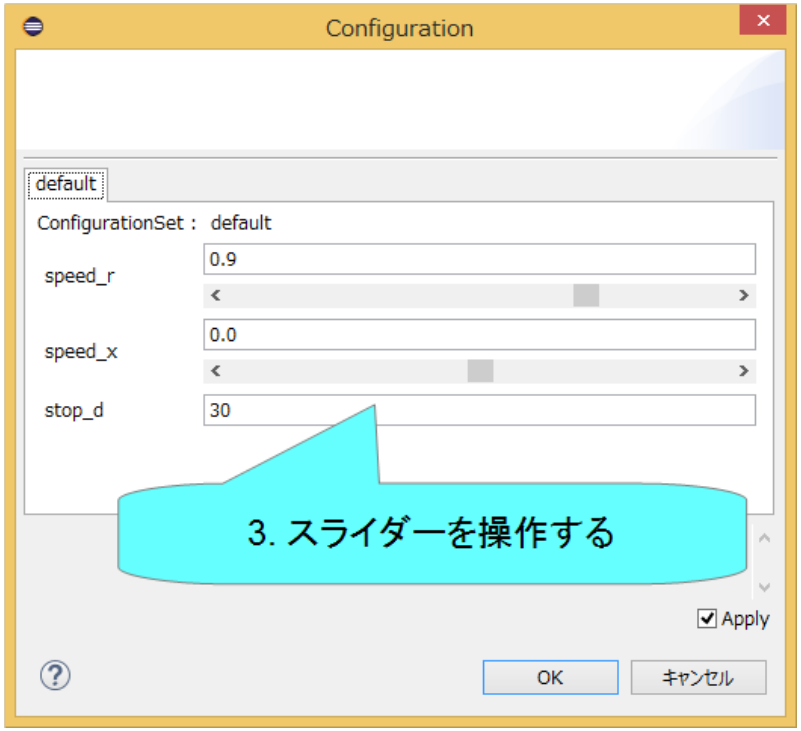


RTCが緑色になればアクティブ化成功

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

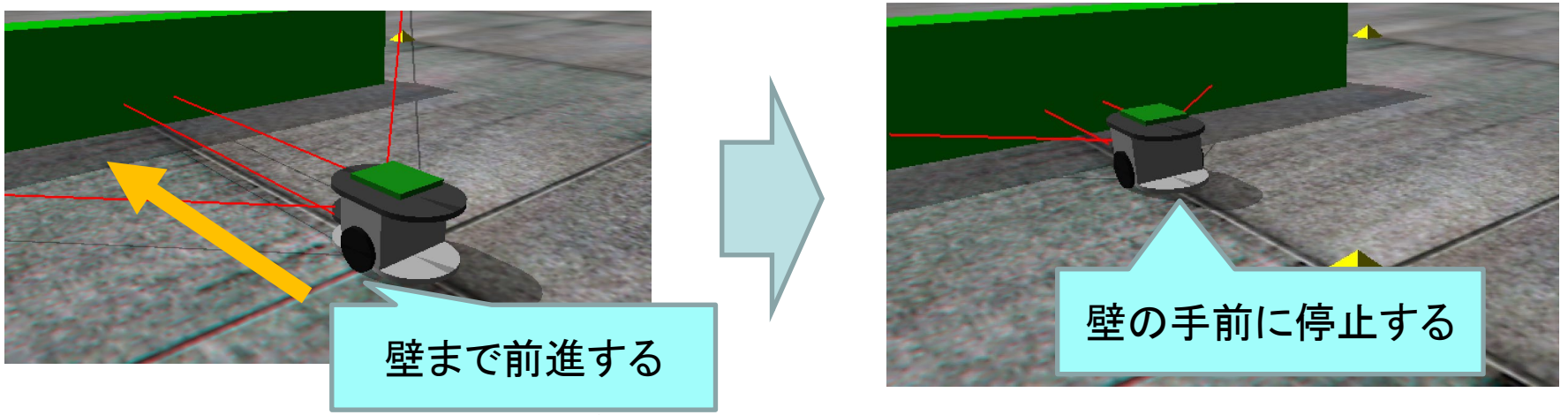




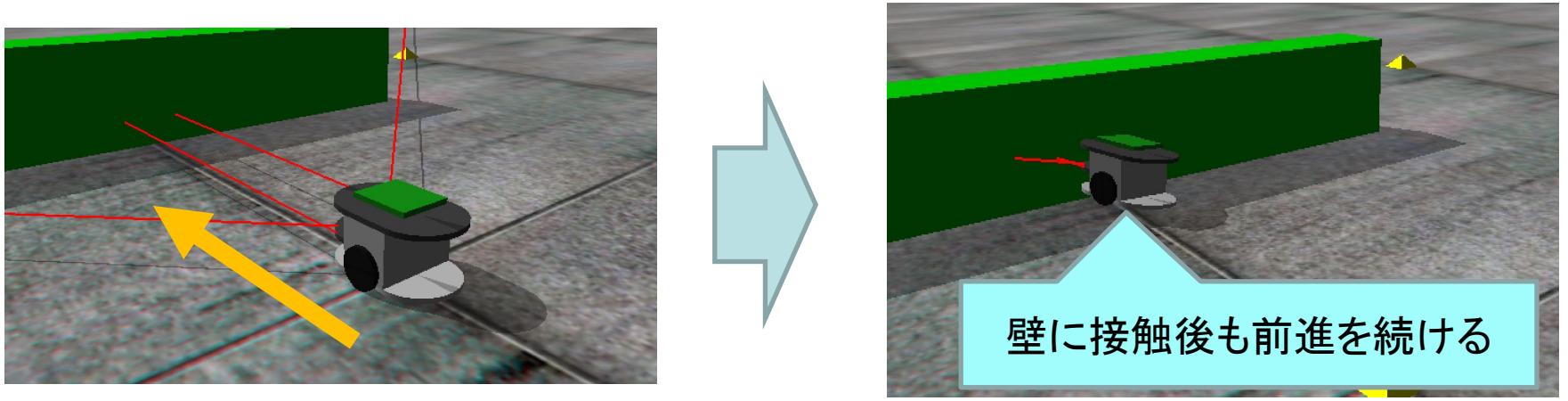
- 以下の動作ができるか確認
  - シミュレータ上のロボットがスライダーで操作できるか？
  - ロボットが障害物に近づくと停止するか？

動作確認

- 距離センサに壁が近づくと停止する場合

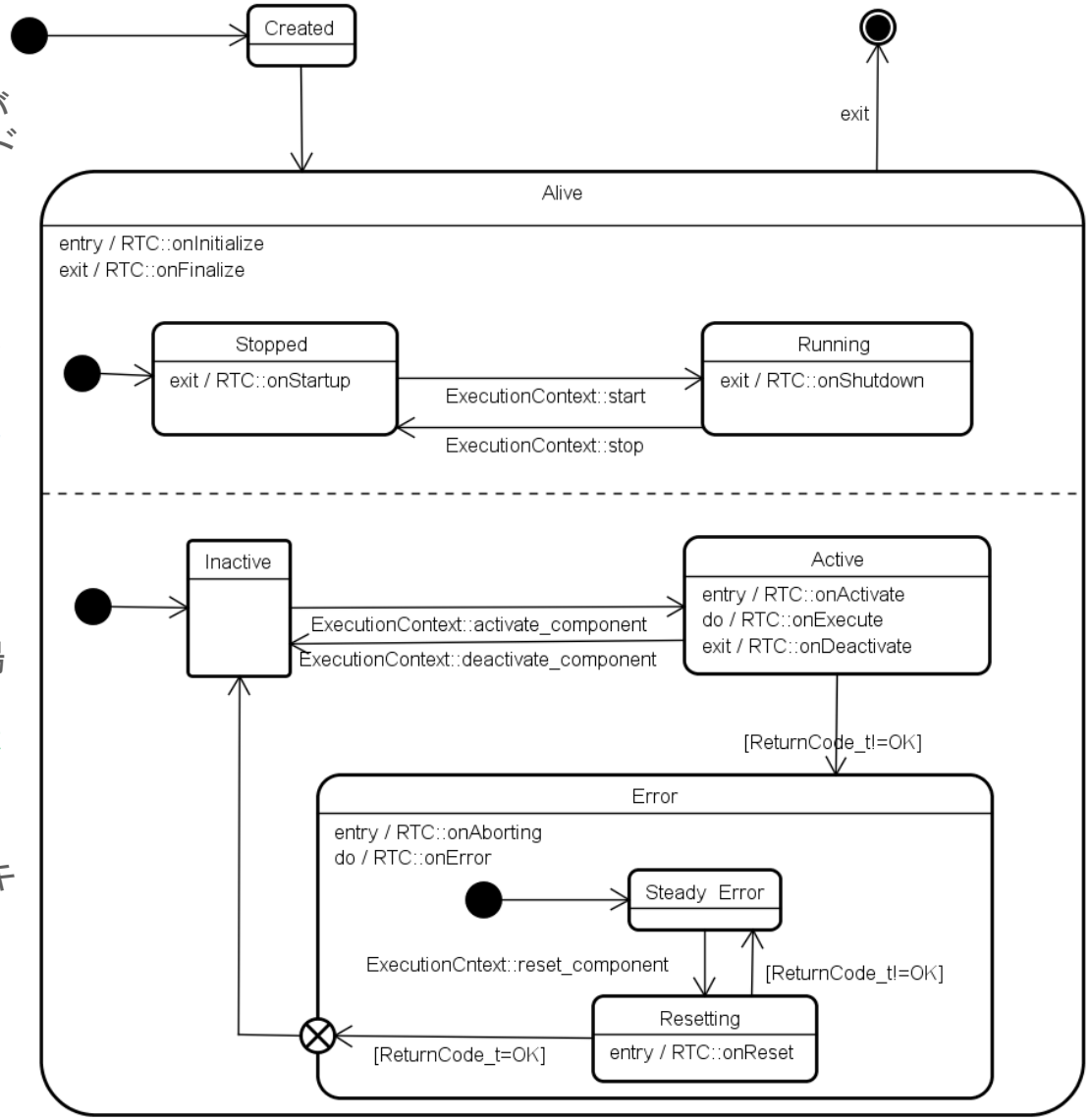


- 距離センサに壁が近づいても停止しない場合



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

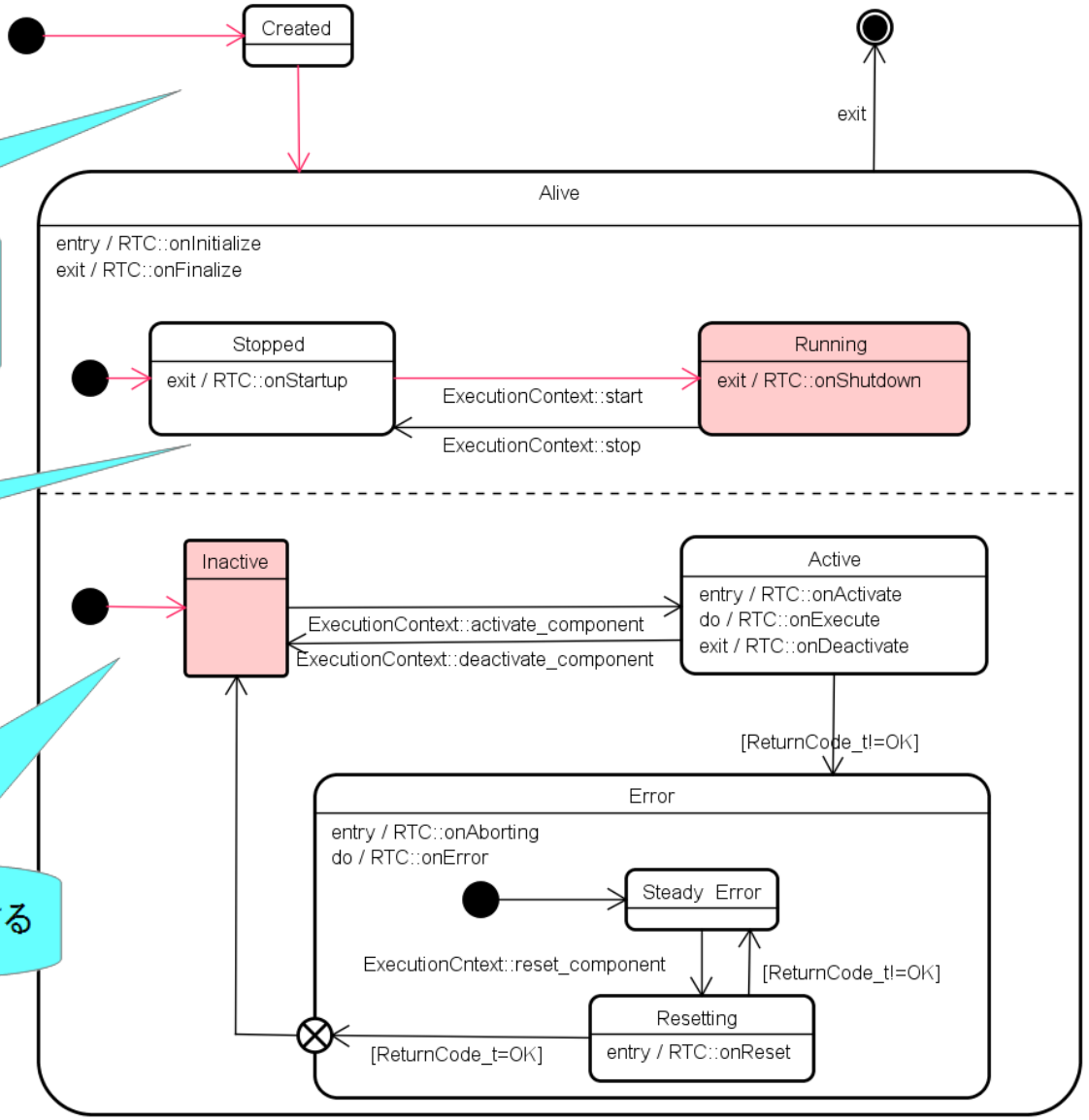


RTコンポーネントの状態遷移(生成直後)

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

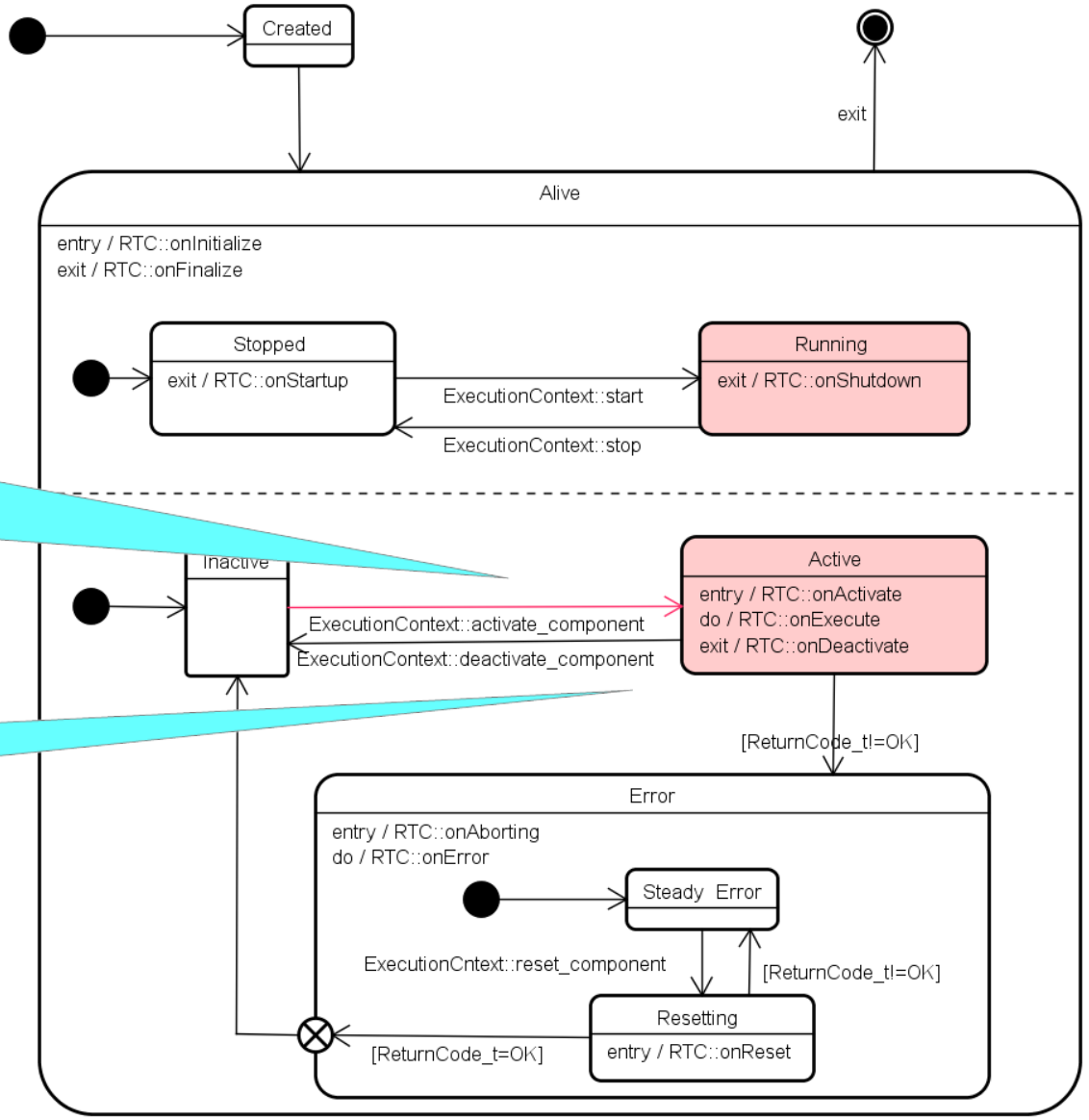
Created状態の次にInactive状態に遷移する



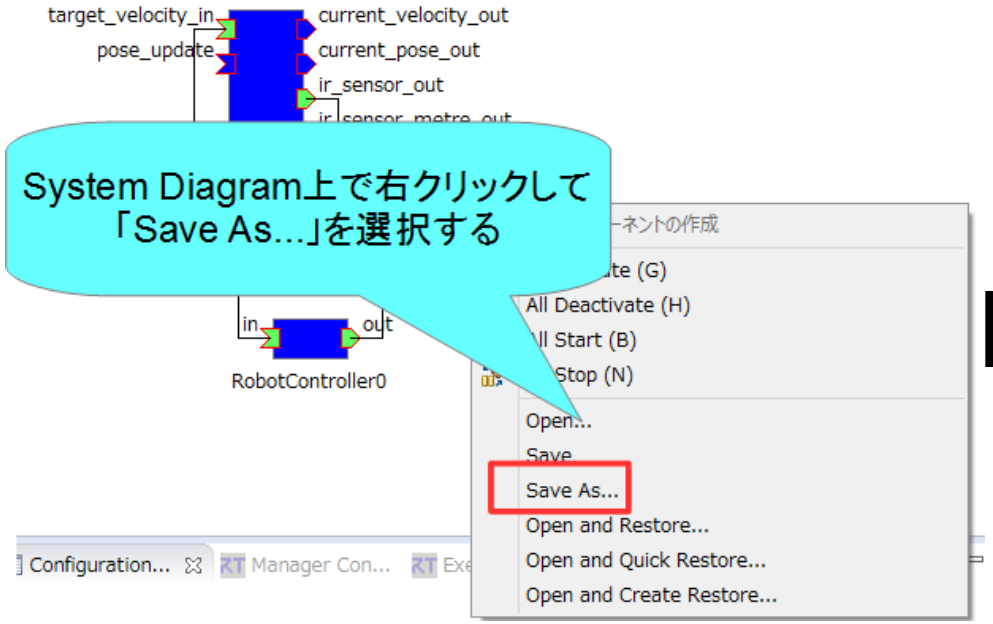
RTコンポーネントの状態遷移(アクティブ化)

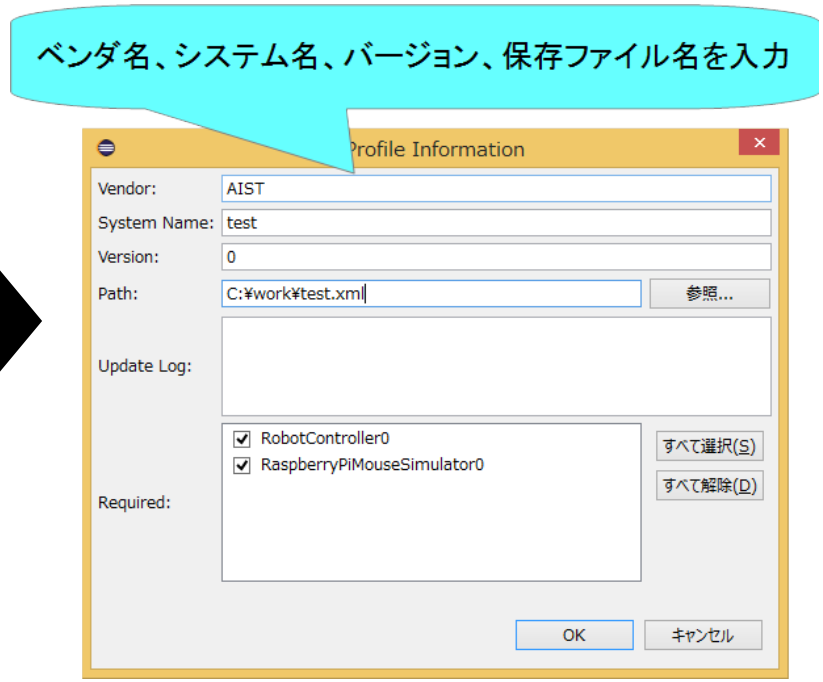
RTシステムエディタの操作によりRTコンポーネントのアクティブ化を行うとactivate_componentメソッドが呼び出される。
activate_componentメソッドによりコンポーネントがActive状態に遷移する。
この時onActivatedコールバックが実行される

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。

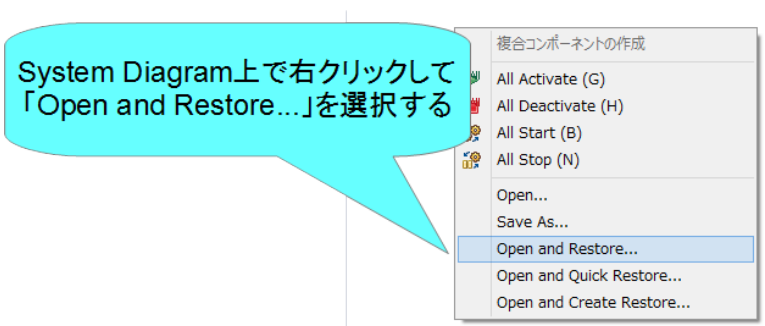


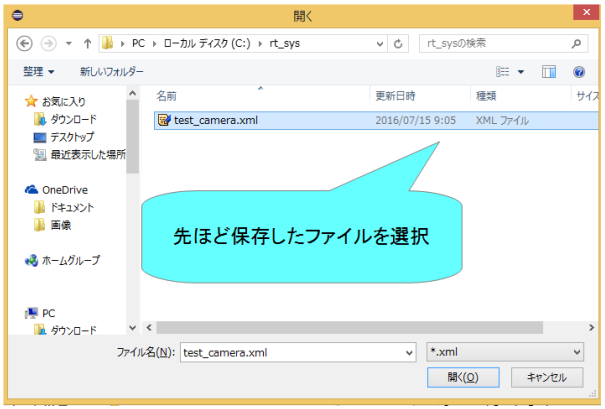
システムの保存

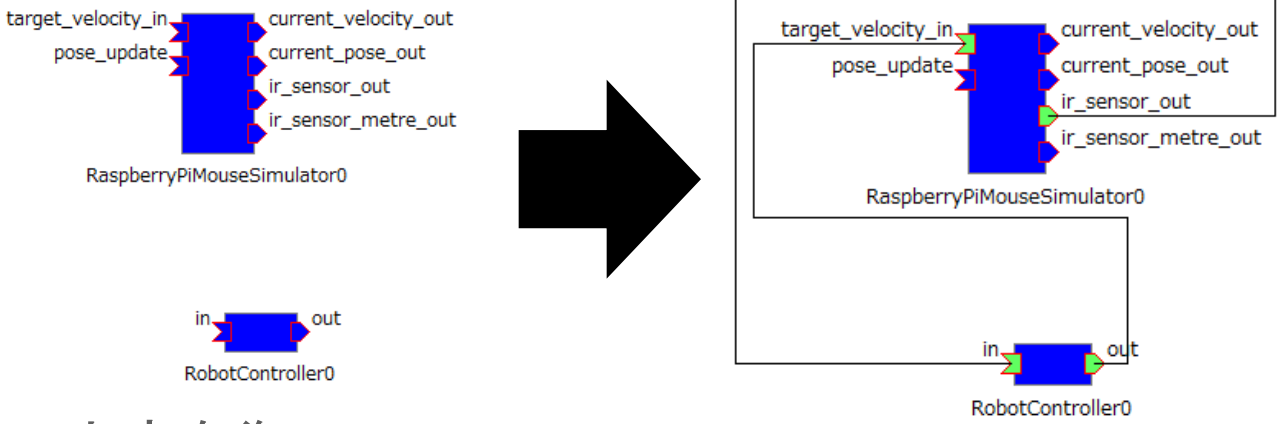

target_velocity_in
pose_update
current_velocity_out
current_pose_out
ir_sensor_out
System Diagram上で右クリックして
「Save As...」を選択する
in
out
RobotController0
All Deactivate (H)
Open...
Save
Save As...
Open and Restore...
Open and Quick Restore...
Open and Create Restore...
Configuration...
Manager Con...


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:
Required:
RobotController0
RaspberryPiMouseSimulator0
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



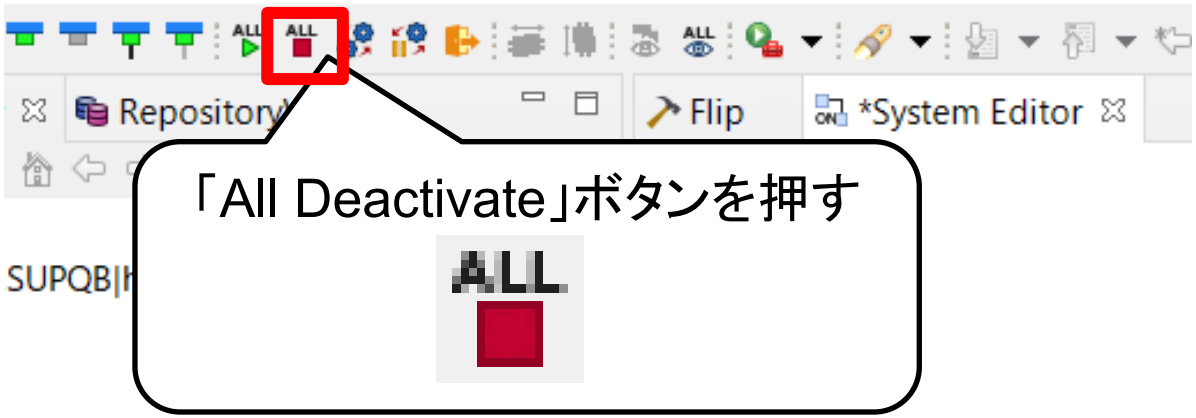




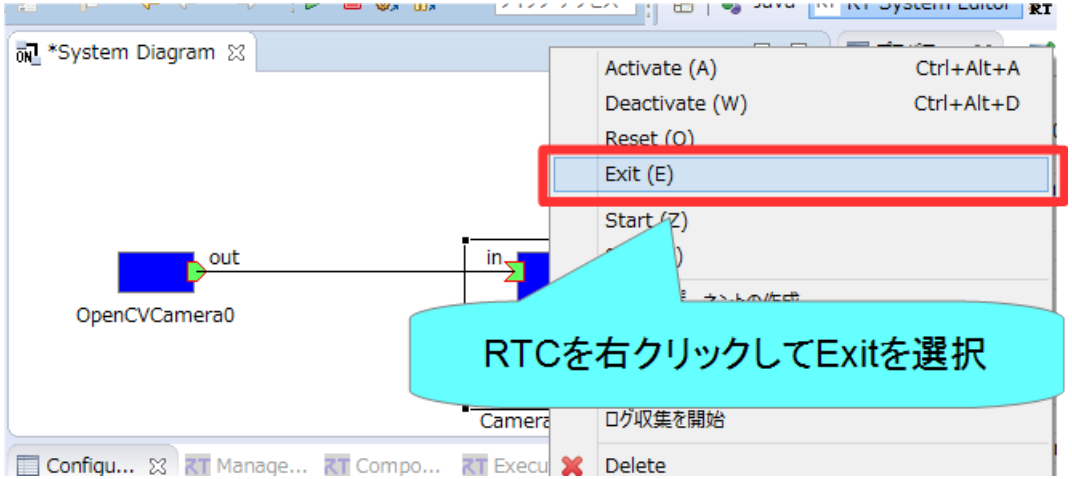
- 以下の内容を復元
  - ポート間の接続
  - コンフィギュレーション
  - 「Open and Create Restore」を選択した場合はマネージャからコンポーネント起動

非アクティブ化、終了

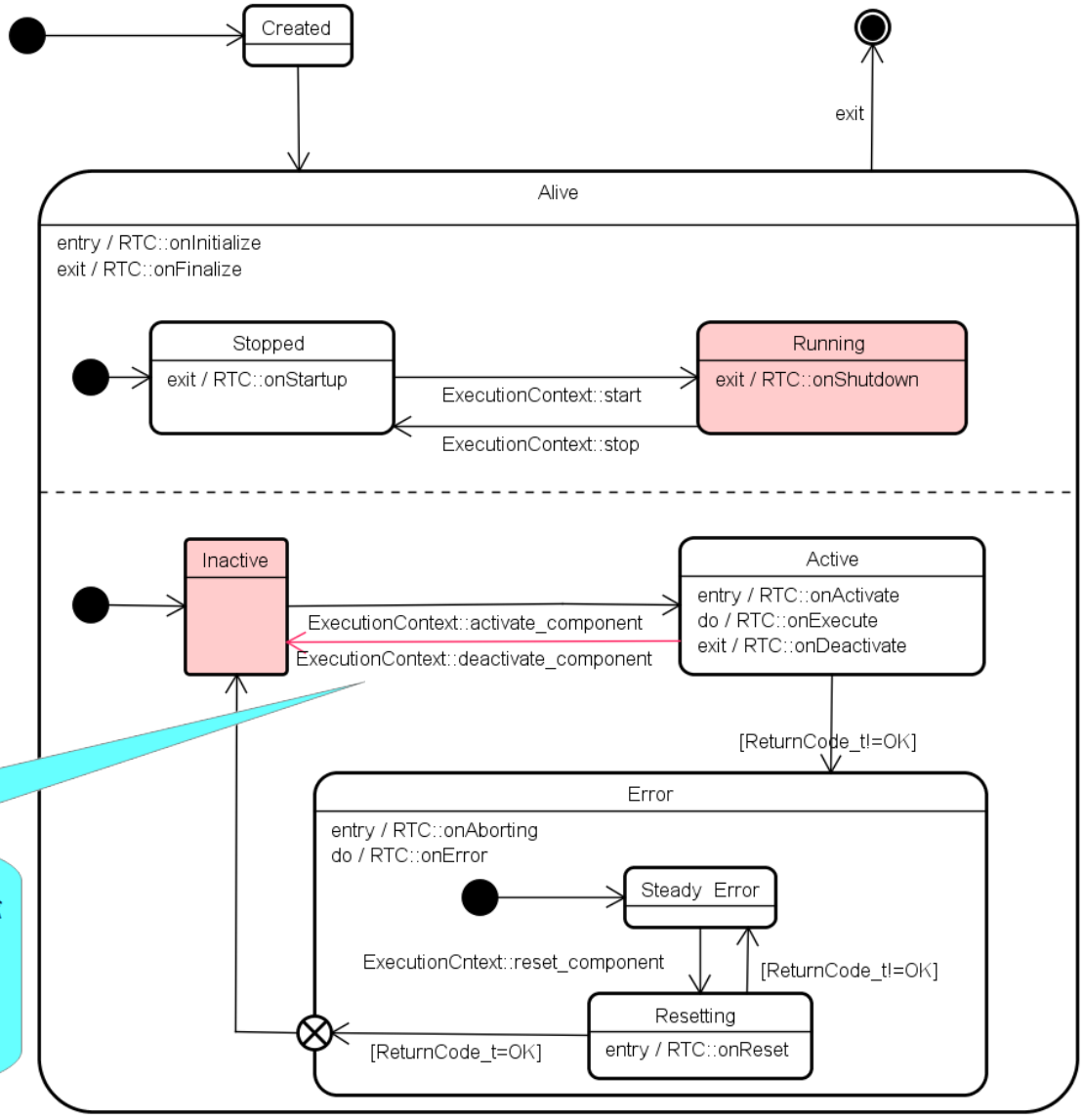
- 非アクティブ化



- 終了



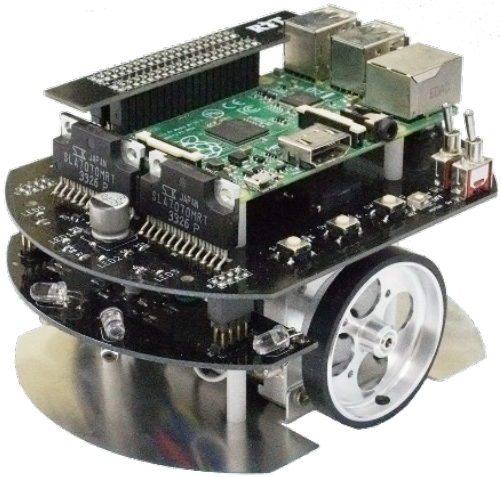
RTコンポーネントの状態遷移(非アクティブ化)

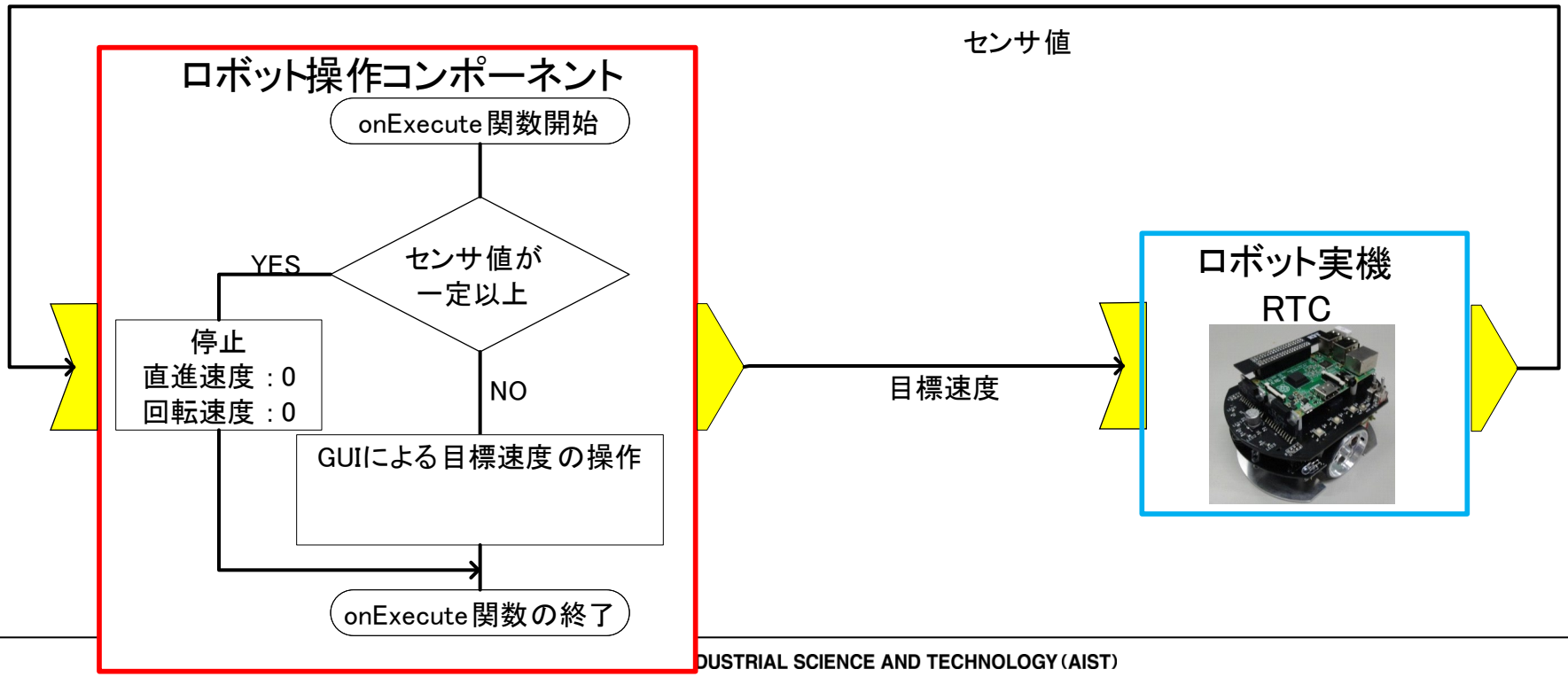


RTシステムエディタの操作によりRTコンポーネントの非アクティブ化を行うとdeactivate_componentメソッドが呼び出される。
deactivate_componentメソッドによりコンポーネントがInactive状態に遷移する。
この時onDeactivatedコールバックが実行される

実習内容（2）

- 車輪型移動ロボットを操作するRTシステムの作成
  - Raspberry Pi Mouse を使用
- ~~まずはジョイスティックコンポーネントで動作確認を行う~~
- ~~動作確認後、~~各自で作成したコンポーネントでロボットの操作を行う





Raspberry Piマウス実機との接続

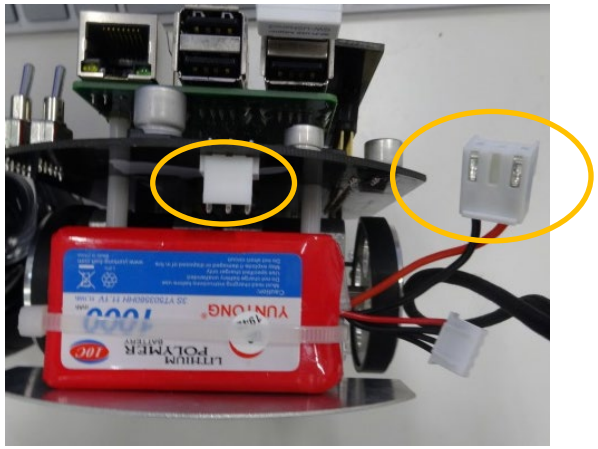
- Raspberry PiとノートPCを無線LANで接続
  - Raspberry Piが無線LANアクセスポイントになる

- 注意事項
  - ノートPCに複数のネットワークインターフェースが存在する場合にRTCの通信ができなくなる可能性があります。
    - 問題が発生した場合は個別に対応します。
  - ~~Raspberry Piアクセスポイント接続後はインターネットに接続できなくなります。~~
  - Raspberry Piアクセスポイント接続後に、**起動済みの**OpenRTP、**ネームサーバー、RTCは再起動してください**。
  - Raspberry Piはシャットダウンしてから電源スイッチをオフにするようにしてください
  - モーター電源スイッチはこまめに切るようにしてください

Rapberry Piマウスの電源について

- Raspberry Piマウスを以下のどちらかと接続する
  - リチウムポリマーバッテリー
    - 充電が必要(配布したバッテリーは充電済み)

    - VH3ピンでRaspberry Piマウスと接続する

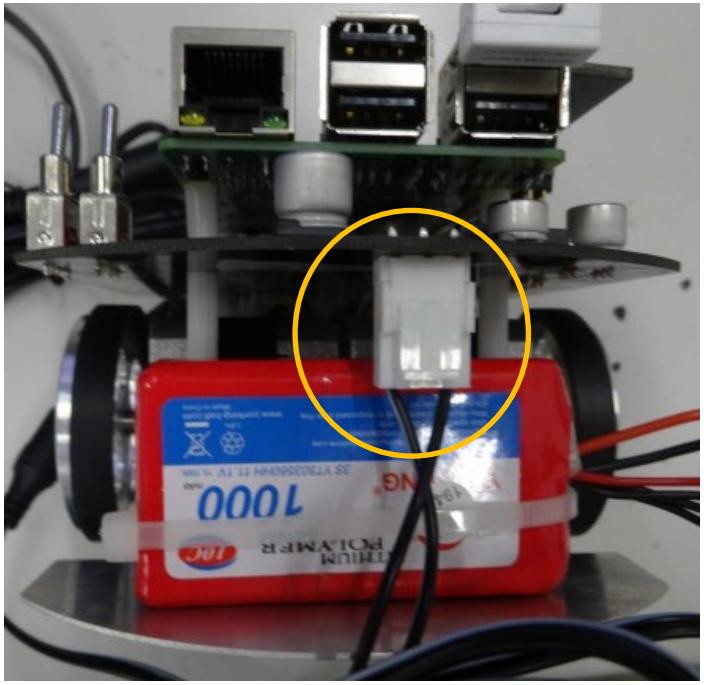


  - 電源ケーブル

電源ケーブルの接続

- 充電が切れることがあるのでこちら推奨
- 3本のケーブルを接続する

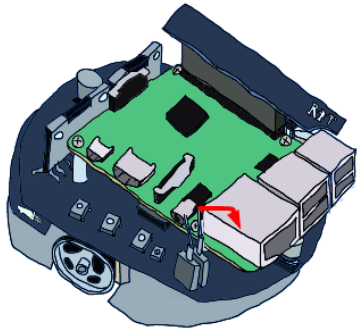
Raspberry Piマウスに接続する。



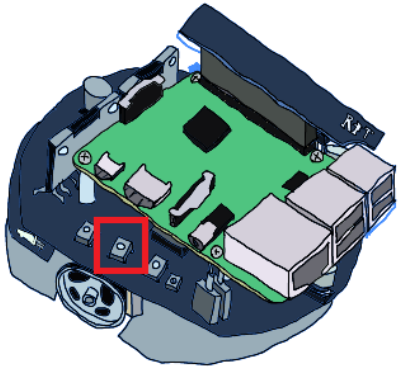
コンセントに接続
必要に応じて3P→2P変換アダプターを使用する

Raspberry Piの起動

- Raspberry Piの電源投入
  - 2つ並んだトグルスイッチの内側のスイッチをオンにする

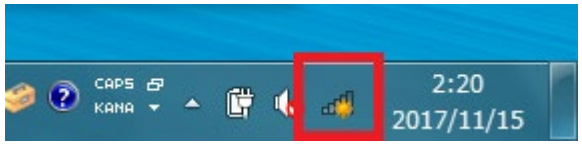


- 電源を切る場合
  - 3つ並んだボタンの中央のボタンを1秒以上押す
  - 10秒ほどでシャットダウンするため、その後に電源スイッチをオフにする



Raspberry Piとの接続

- 無線LANアクセスポイントとの接続
  - SSID、パスワードはRaspberry Piマウス上のシールに記載
  - 接続手順(Windows)
    - 画面右下のネットワークアイコンをクリック

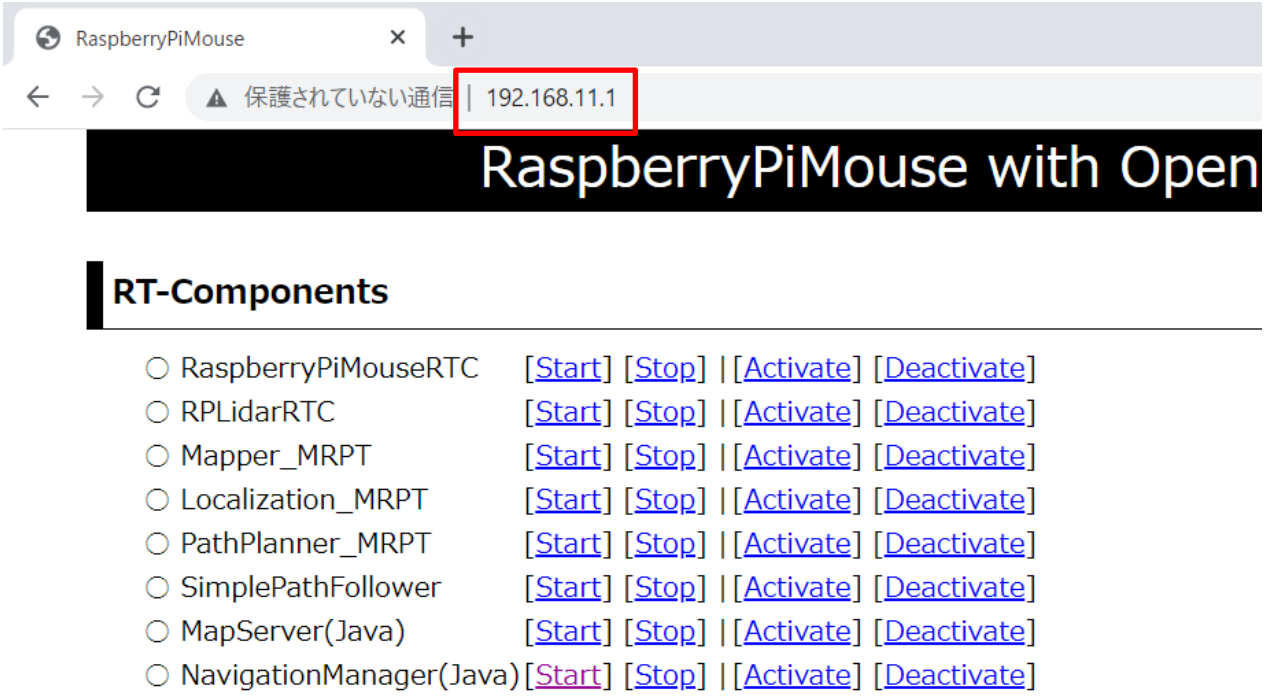


    - raspberrypi_xx(もしくはRPiMouse_xx)に接続後、パスワードを入力

RaspberryPi Mouse V3

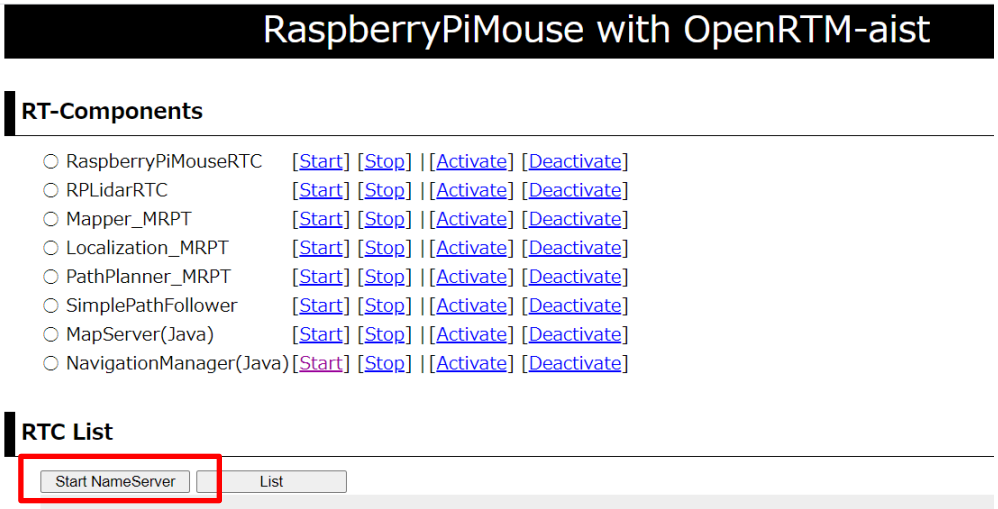
- **RaspberryPi Mouse V3** (LiDARが付属している) の場合、ネームサーバー、RTCが自動起動しないため、WEBブラウザからの操作で起動する必要がある。

- Chrome、Firefox、Edge等で**192.168.11.1**にアクセスする。

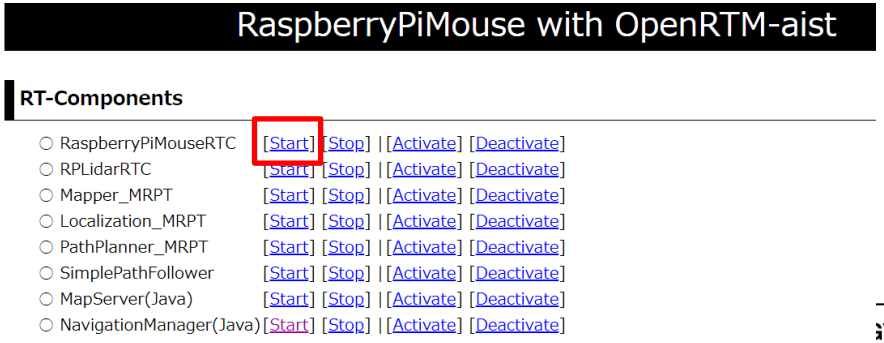


RaspberryPi Mouse V3

- ネームサーバーを起動する
  - Start NameServerボタンを押す

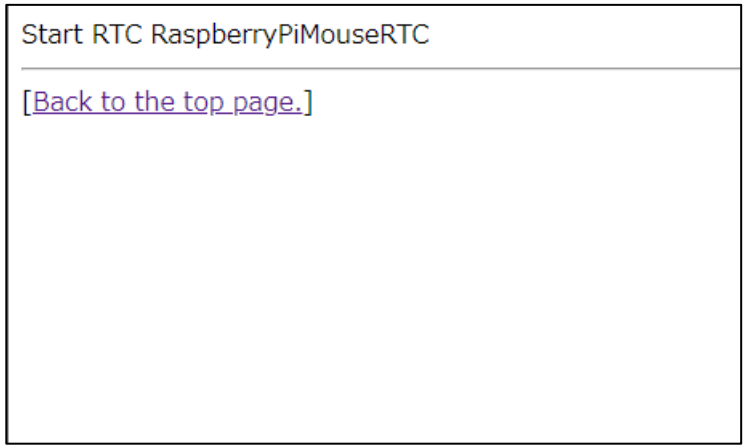


- RaspberryPiMouseRTCを起動する
  - RaspberryPiMouseRTCのStartを押す



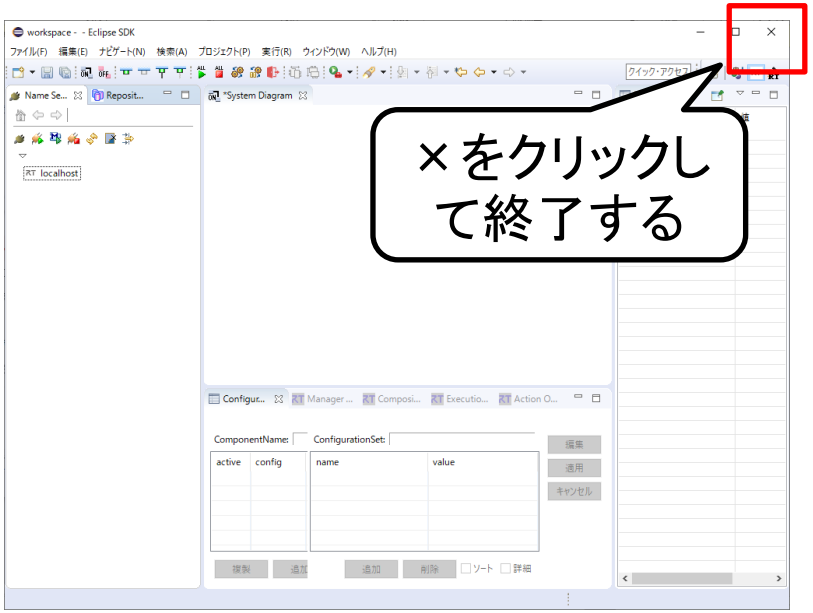
RaspberryPi Mouse V3

- RaspberryPiMouseRTCを起動する
  - 元の画面に戻るには「Back to the top page.」をクリックする



OpenRTP再起動

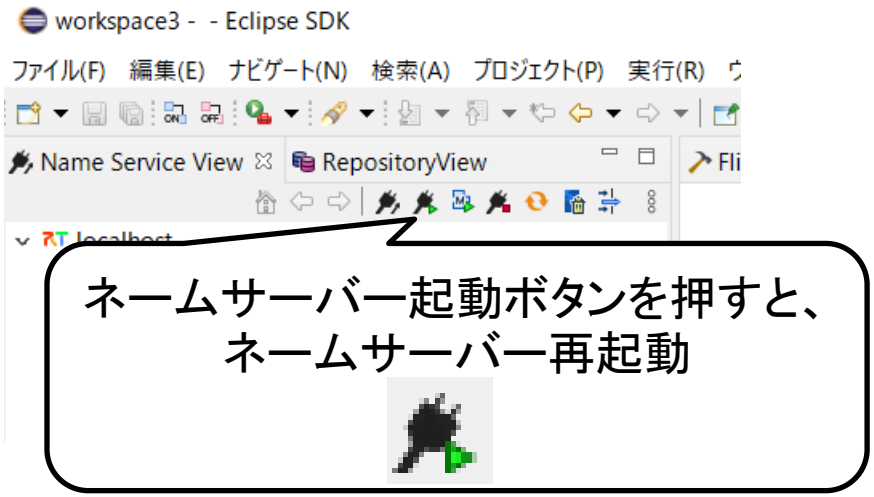
- OpenRTPを終了する



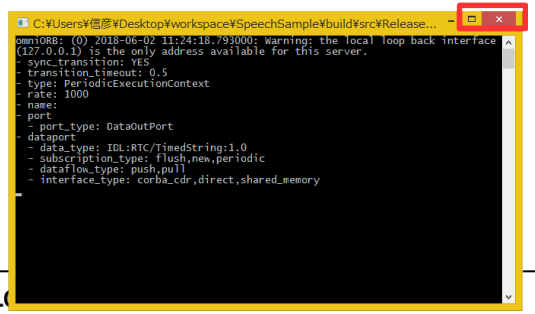
- OpenRTPを再起動する
  - デスクトップのショートカットをダブルクリックする(Windows)
  - 「openrtp」というコマンドを入力(Ubuntu)

起動済みのRTC、ネームサーバー再起動

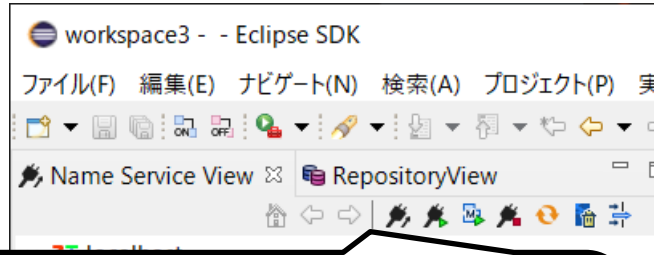
- ネームサーバーを再起動する
  - ネームサーバー起動ボタンで再起動



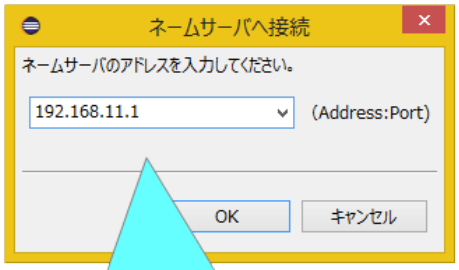
- RTC再起動
  - RTCをexitするか、RTC起動時に表示したウインドウの×ボタンを押して終了する
  - 実行ファイル(RobotControllerComp.exe)を再度実行



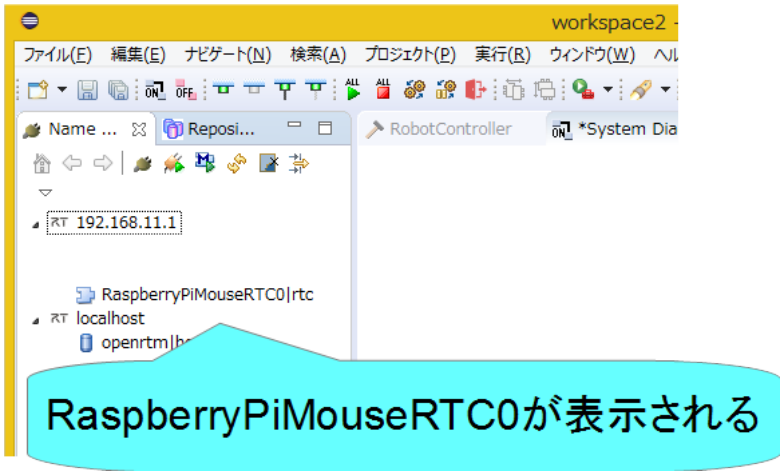
ネームサーバーとの接続



コンセントのアイコンをクリック

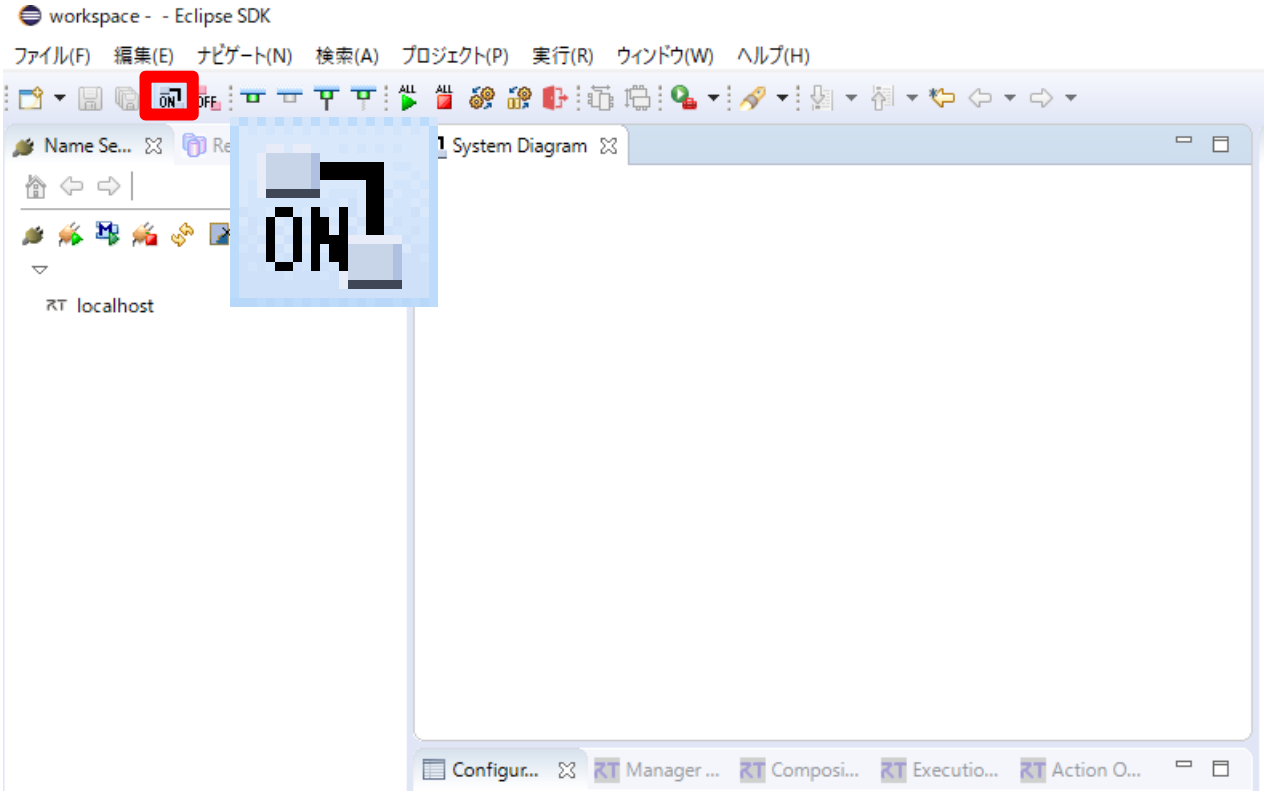


192.168.11.1を入力



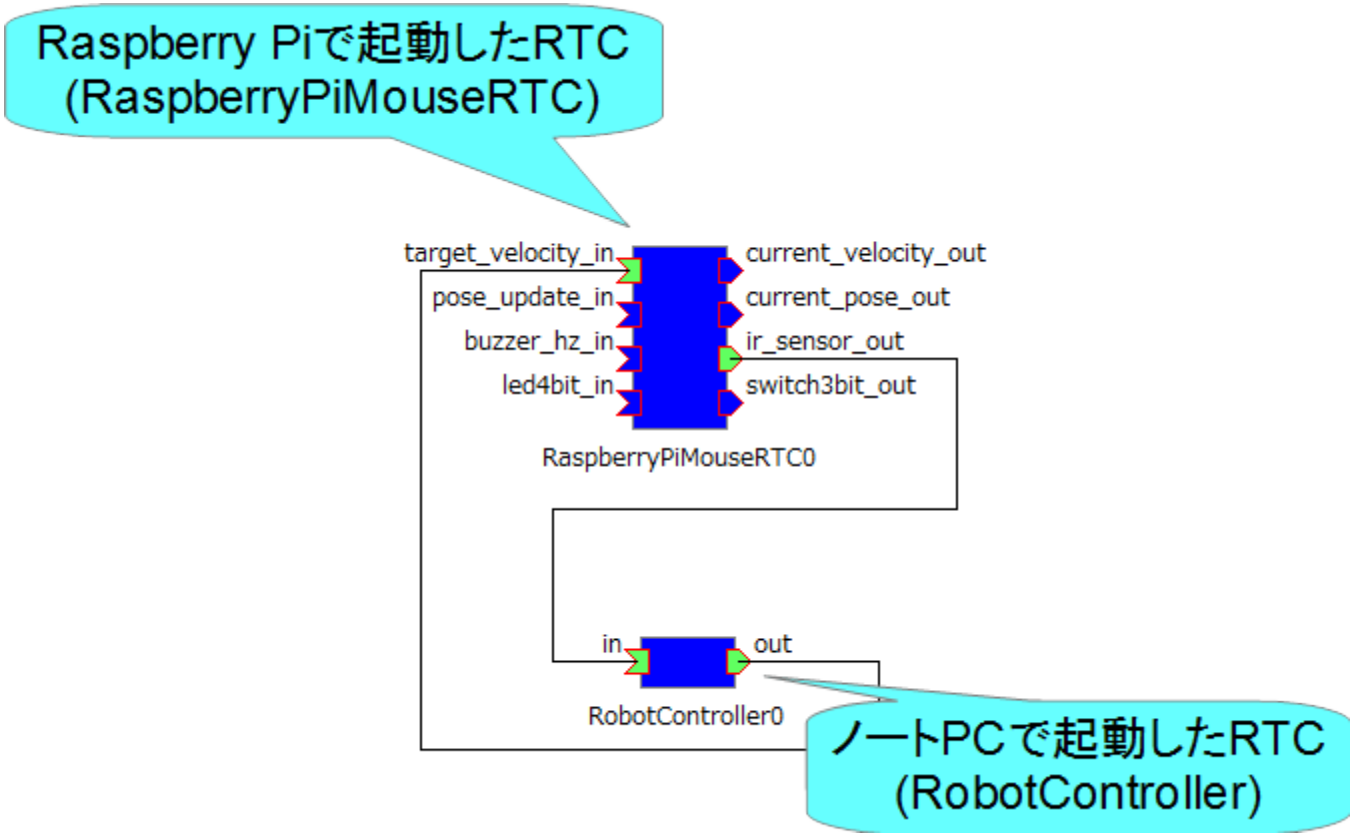
RaspberryPiMouseRTC0が表示される

System Editorの表示



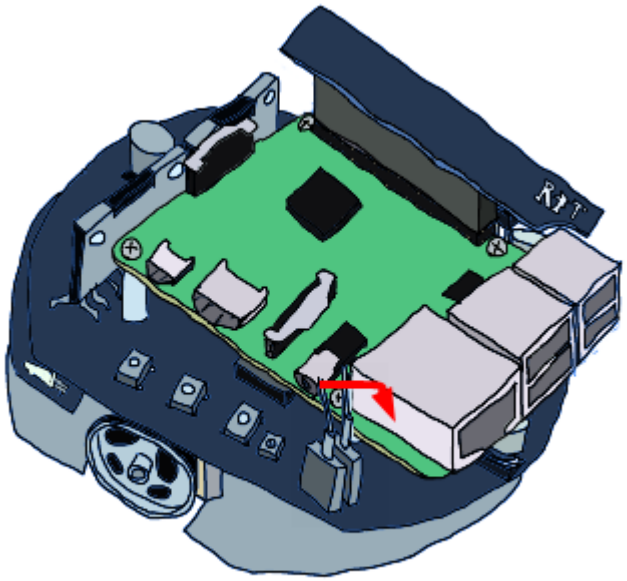
ポートの接続

- **RobotController0**と**RapberryPiMouseRTC0**を接続する



動作確認

- モーターの電源投入
  - 2つ並んだトグルスイッチの外側のスイッチをONにする



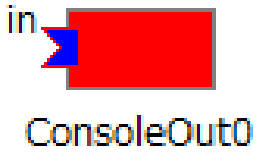
- RTCをアクティブ化して動作確認

おわりに

- 時間があまったら、自作したコンポーネントを改造してみましょう
  - RTCBuilderでポートを追加して、自律移動とJoystickのハイブリッド制御をしてみる
  - センサ値を利用して、壁伝いに移動してみる
  - 迷路で壁にぶつからずに走行する
  - 等

リセット

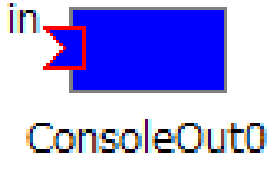
- RTCがエラー状態に遷移した場合にエディタ上には赤く表示される。



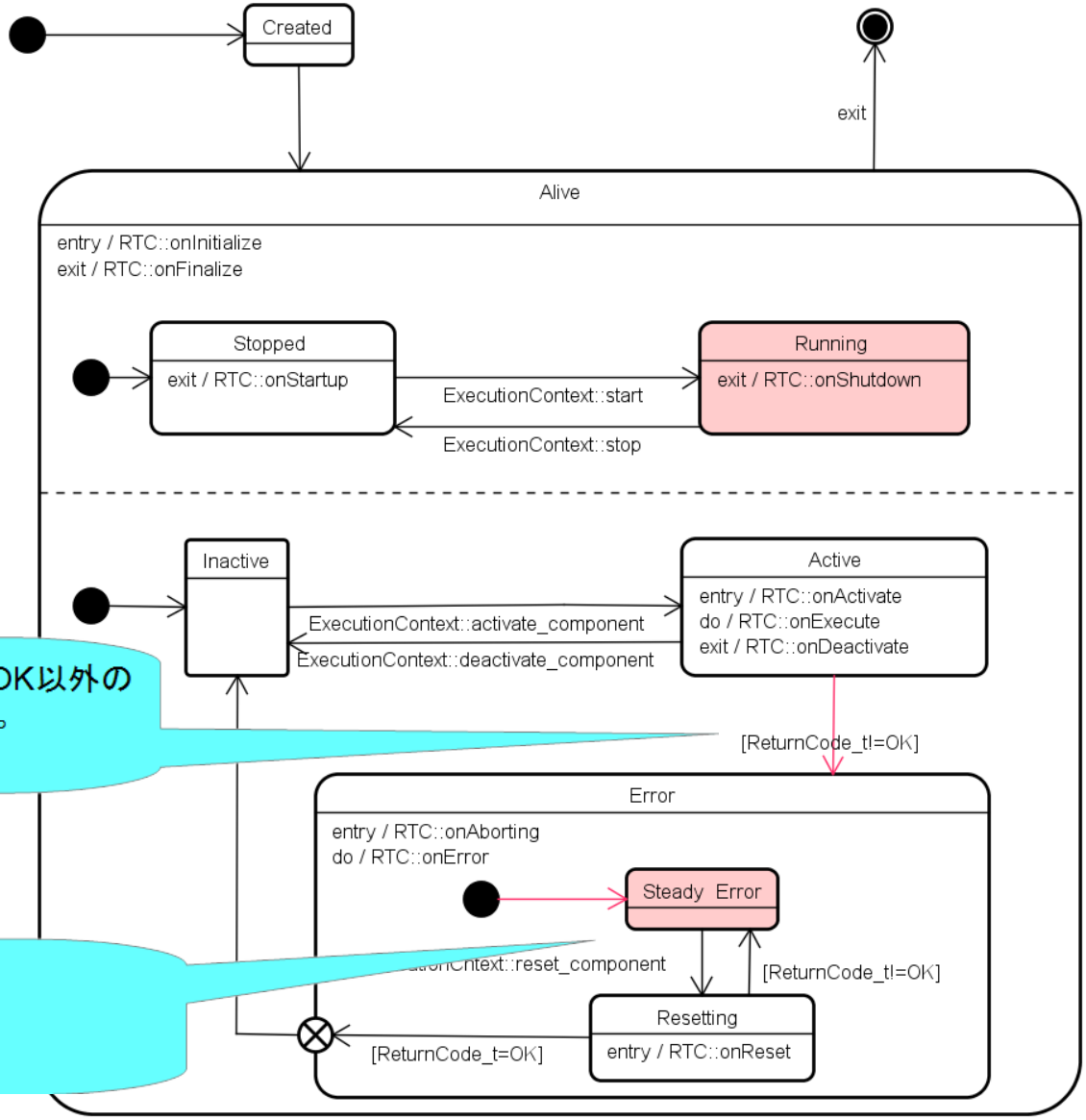

```
RTC::ReturnCode_t Test::onActivated(RTC::UniqueId ec_id)
{
        HANDLE hCom = INVALID_HANDLE_VALUE;
        hCom = CreateFile("COM5", GENERIC_READ | GENERIC_WRITE, 0, NULL, OPEN_EXISTING, 0, NULL);
        if (hCom == INVALID_HANDLE_VALUE)
        {
                return RTC::RTC_ERROR;
    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

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



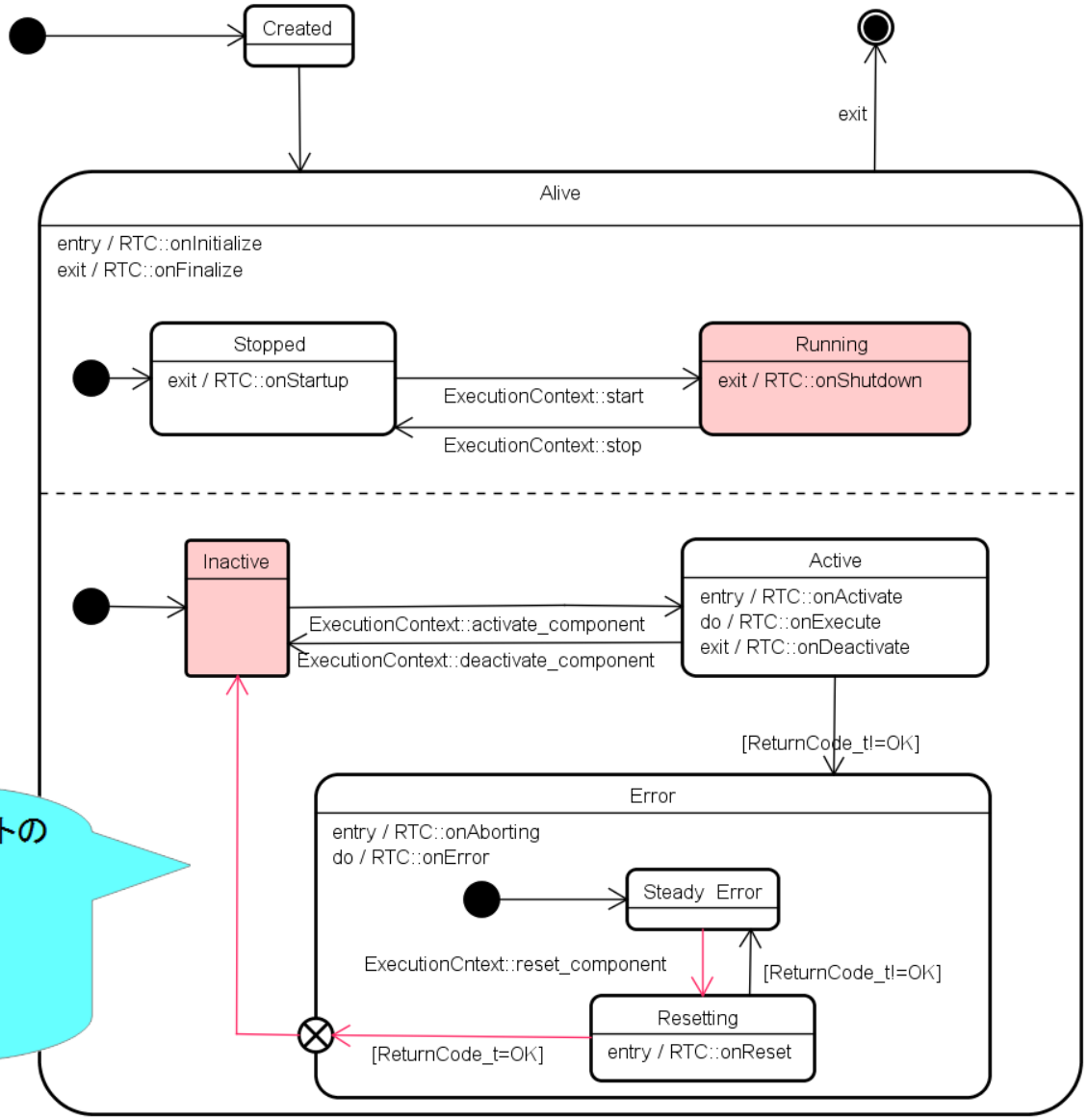
RTコンポーネントの状態遷移(エラー)



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)

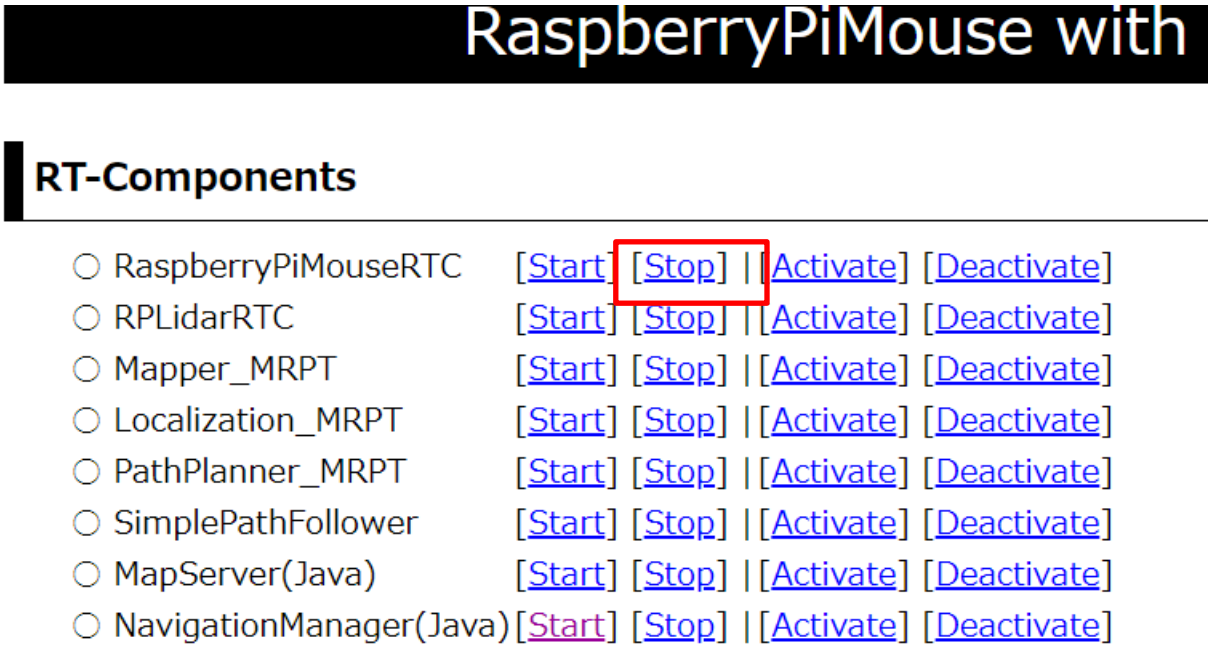


RTシステムエディタの操作によりRTコンポーネントのリセットを行うとreset_componentメソッドが呼び出される。
reset_componentメソッドによりコンポーネントがInactive状態に遷移する。
この時onResetコールバックが実行される

RTC Builder
補足

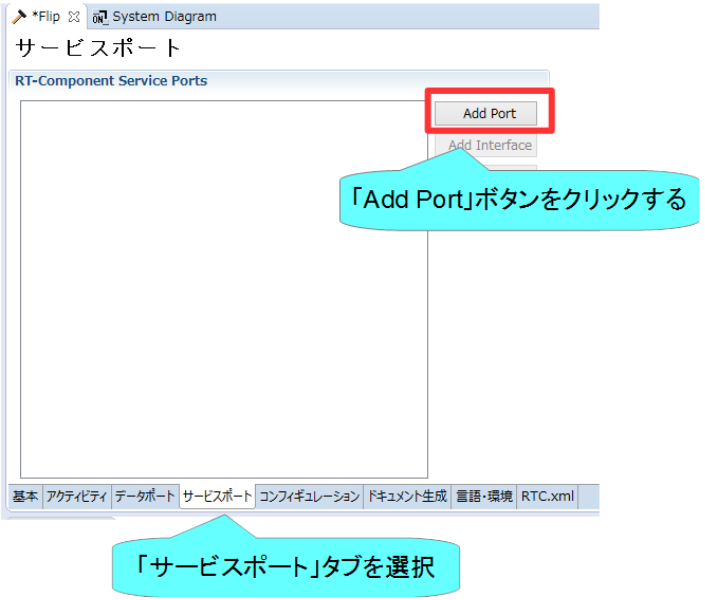
起動済みのRTCの終了

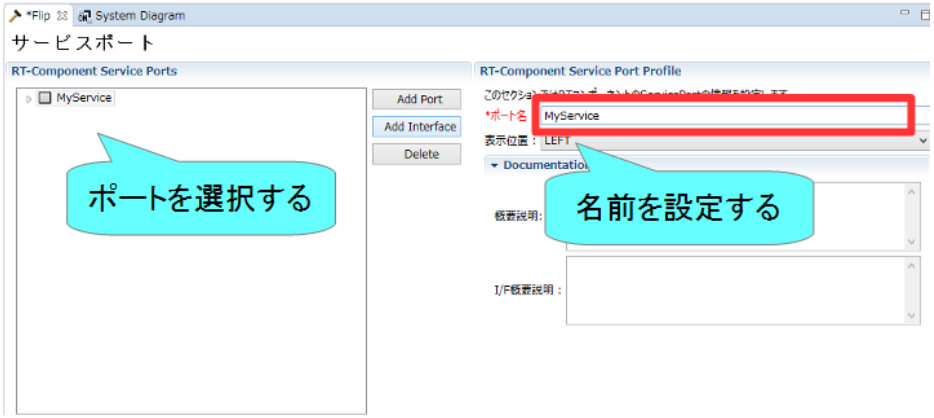
- RaspberryPiMouseRTC、RobotControllerコンポーネントが起動している場合は終了してください。
  - RaspberryPiMouseRTCはWEBブラウザの操作でStopボタンを押す



サービスポートの設定

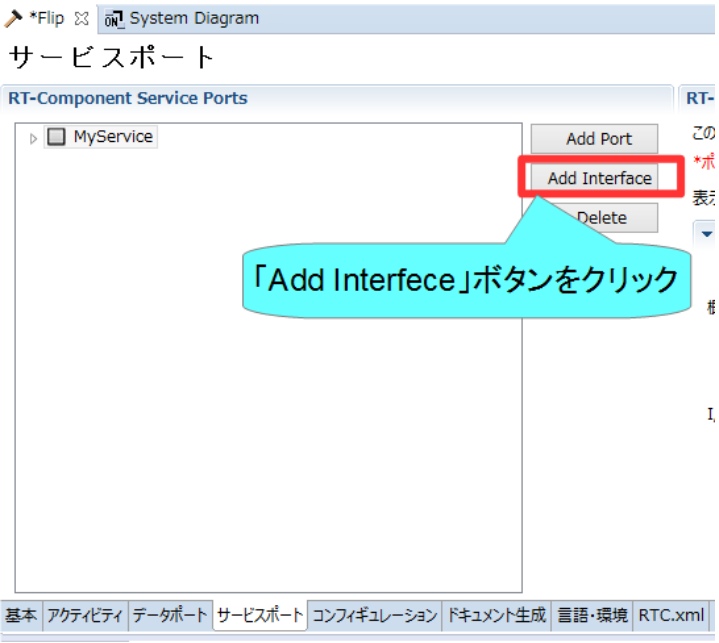
- サービスポートの追加、インターフェースの追加、設定を行う

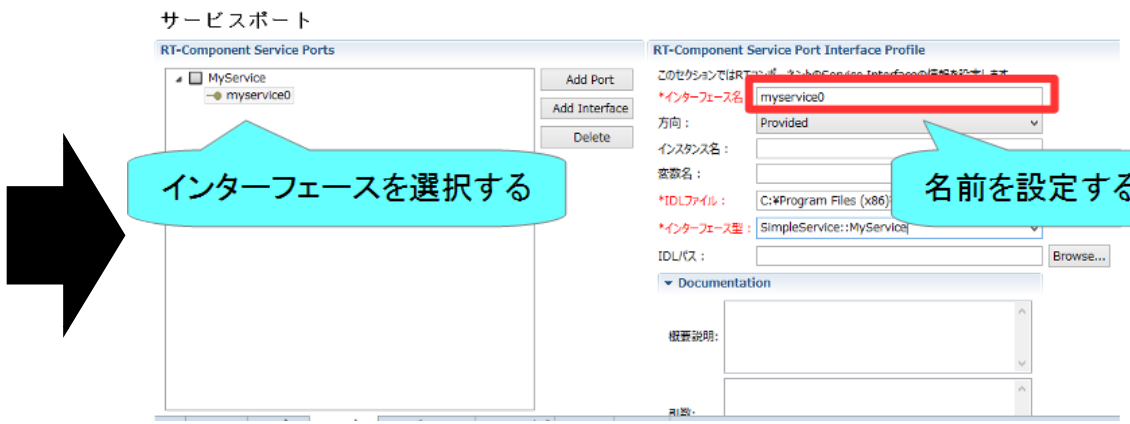




サービスポートの設定

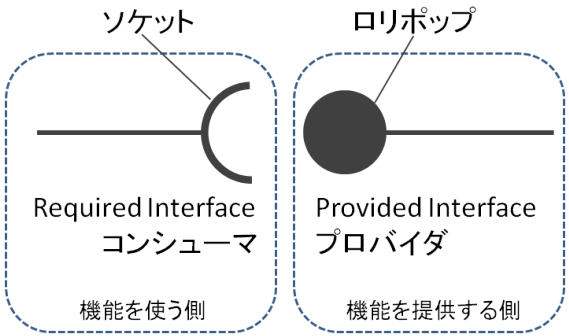
- インターフェースを追加する



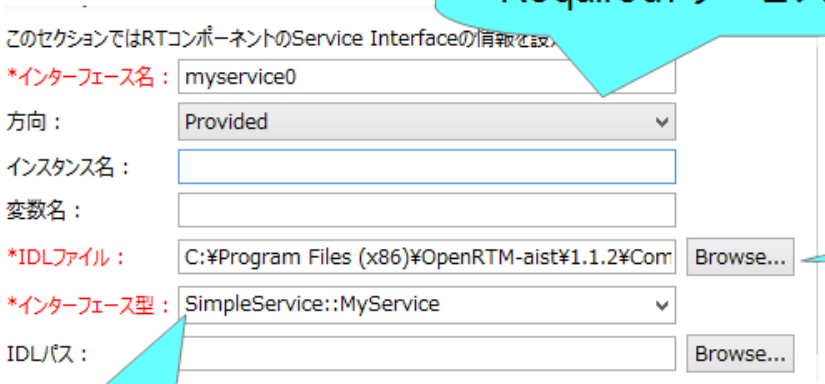


サービスポートの設定

- 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



「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

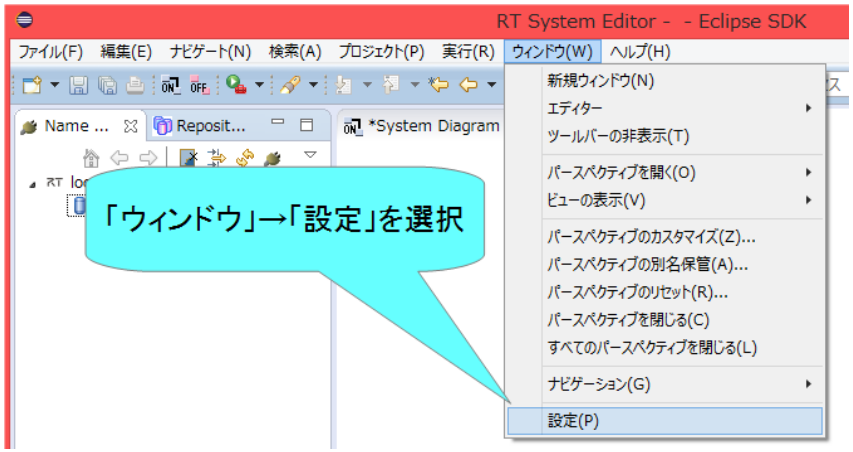
サービスポートの設定

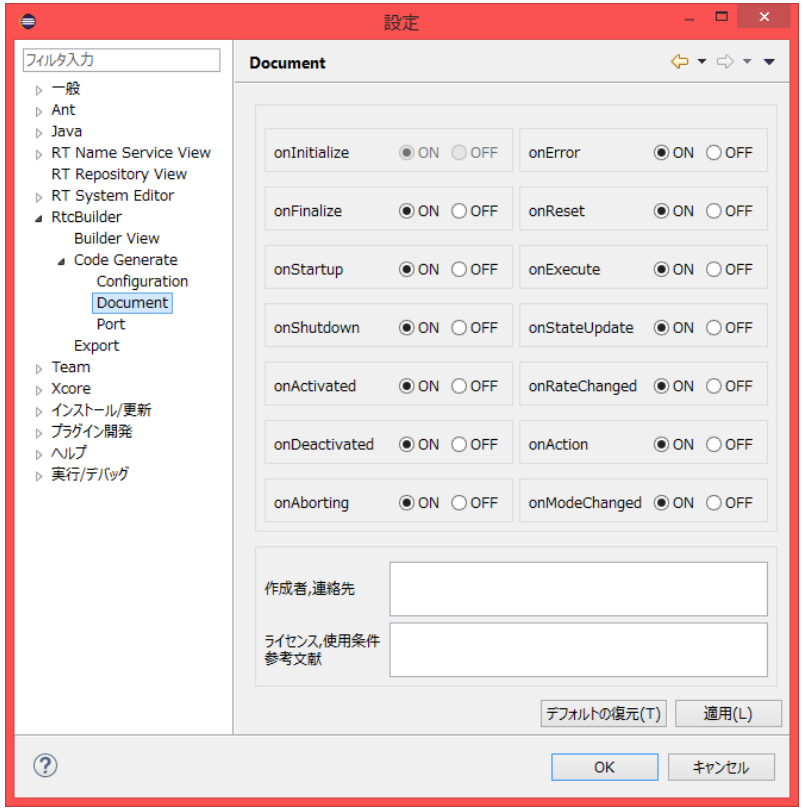
- IDLファイルについて
  - プログラミング言語に非依存のインターフェース定義言語

```
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 };↓
```

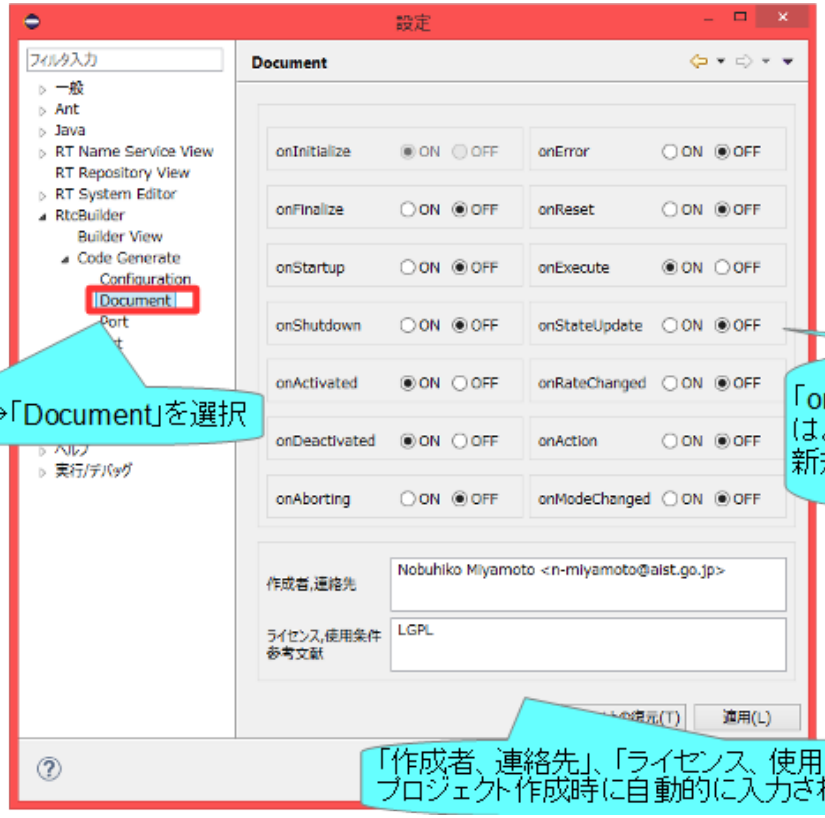
- コンシュマー側でプロバイダ側のecho、get_valueなどのオペレーションを呼び出す

RTC Builderに関する設定





RTC Builder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。

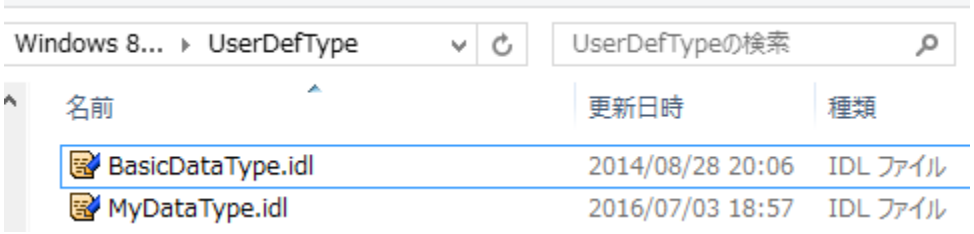
「作成者、連絡先」、「ライセンス、使用条件、参考文献」を入力しておくとプロジェクト作成時に自動的に入力されるので便利。

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

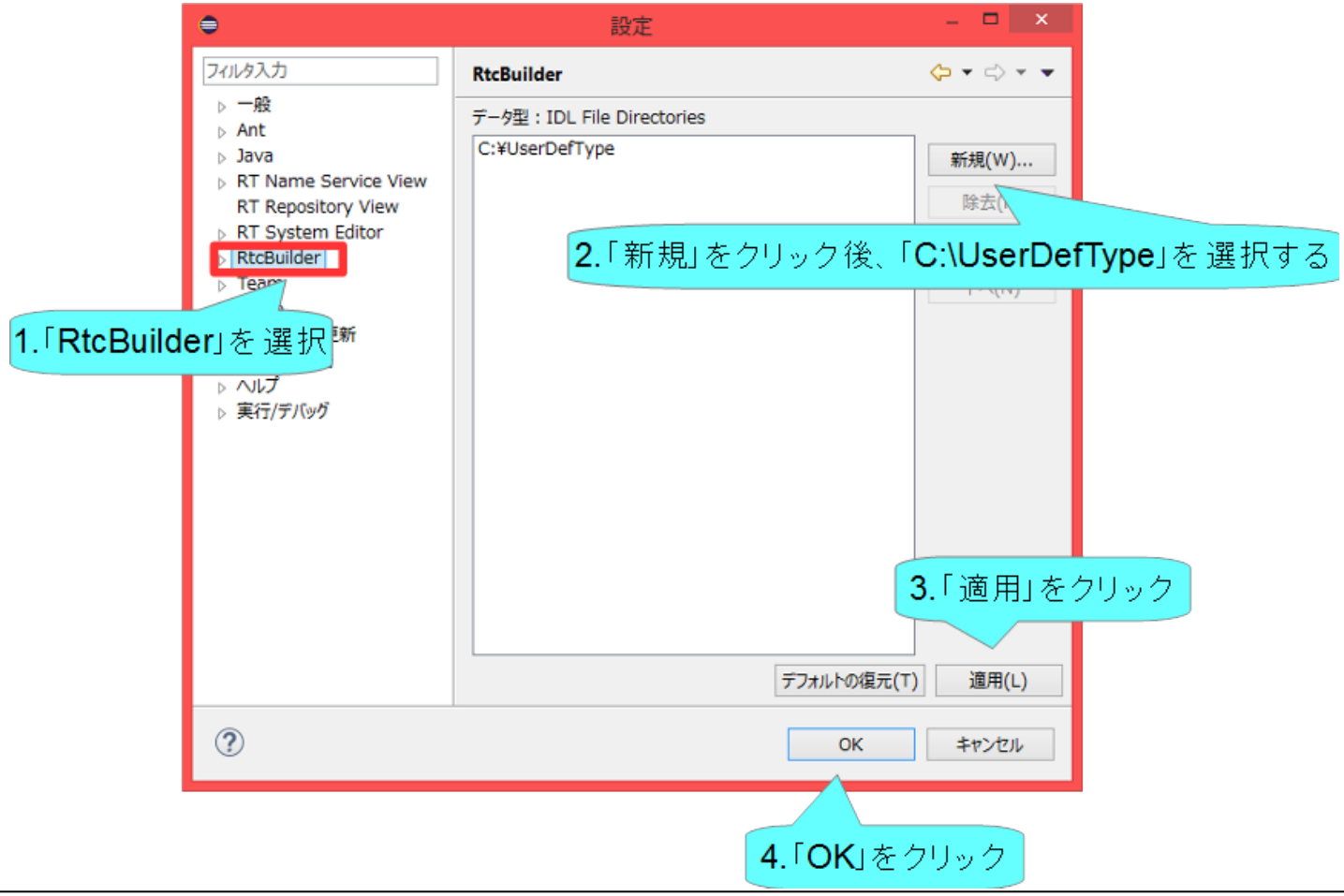
```
1  // @file MyDataType.idl↓
2  #include "BasicDataType.idl"↓
3   ↓
4  struct MyData↓
5  {↓
6    RTC::Time tm;↓
7    short shortVariable;↓
8    long longVariable;↓
9    sequence<double> data;↓
10 };[EOF]
```

    - 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



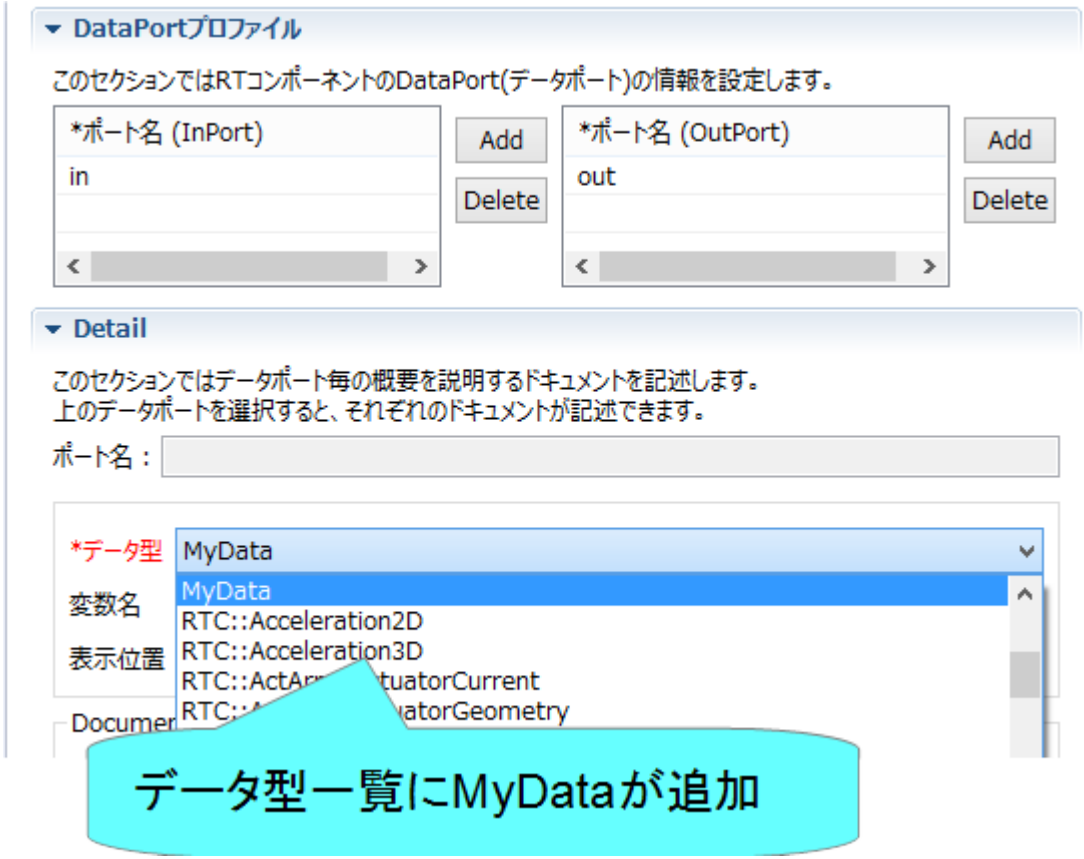
独自のデータ型の利用

- 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



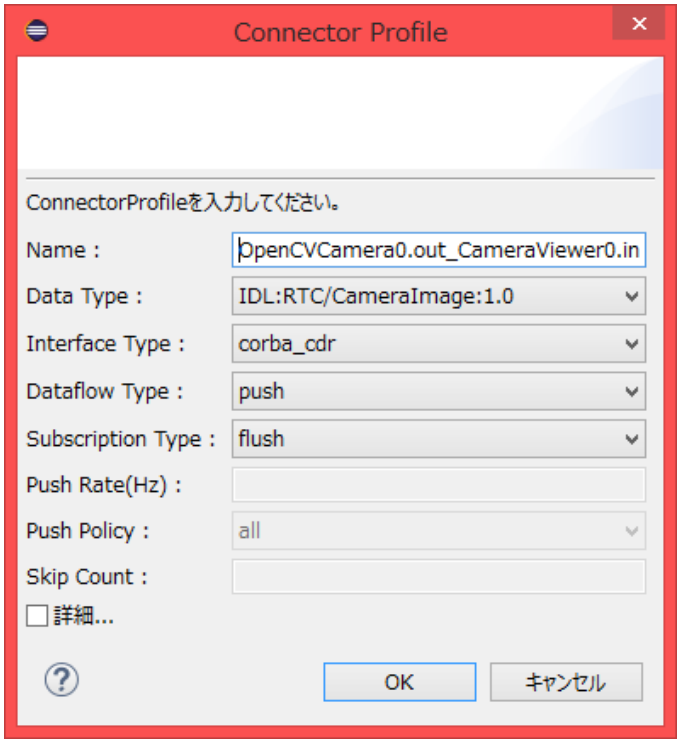
RT System Editor
補足

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

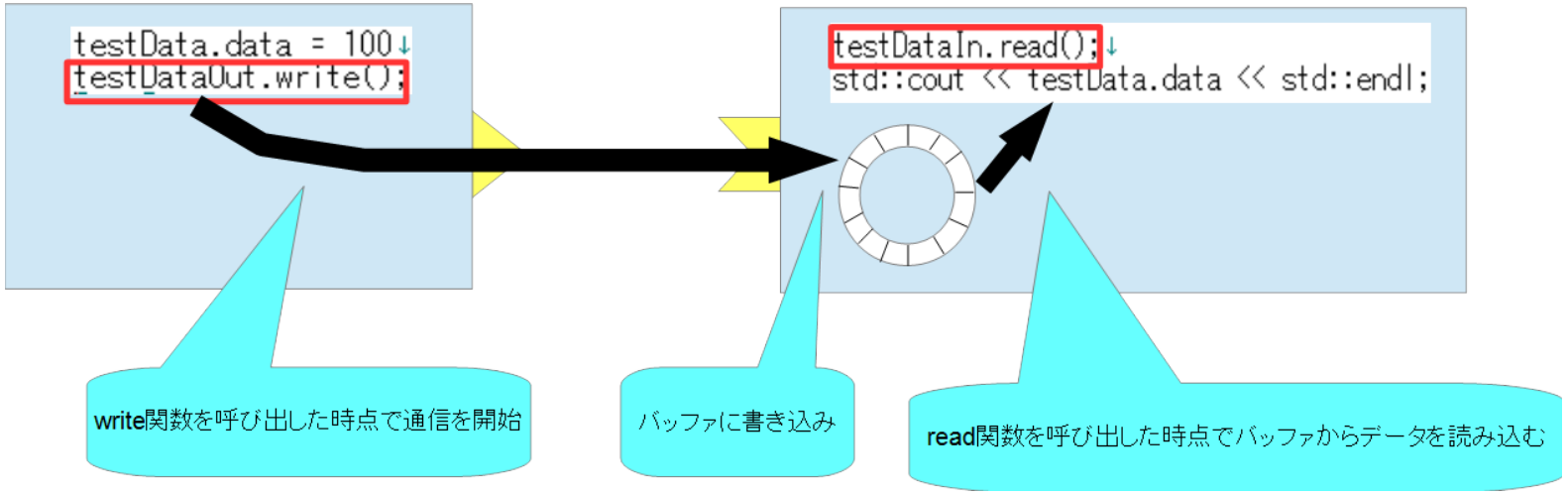
コネクタプロファイルの設定

- InterfaceTye
  - データの送信方法
  - 1.1.2ではcorba_cdr(CORBAによる通信)のみ選択可能
  - 1.2.0では以下の通信方法も選択可能になる予定
    - 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に問い合わせてデータを受け取る
- 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)

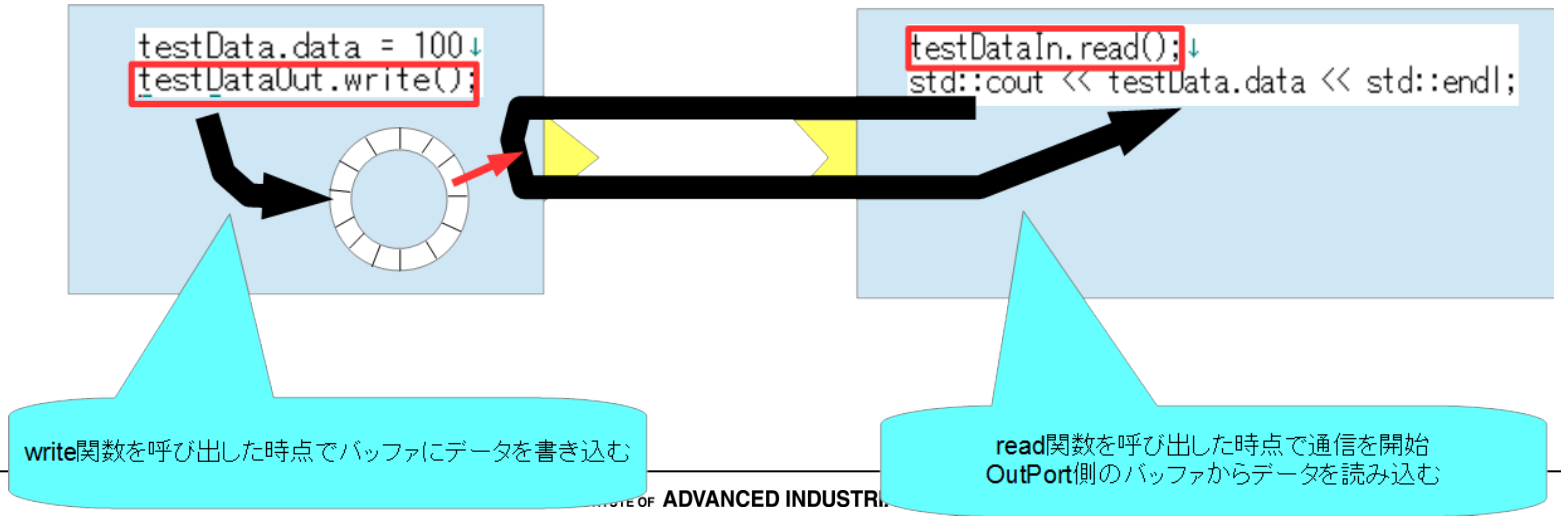


コネクタプロファイルの設定

- DataFlowType
  - Push

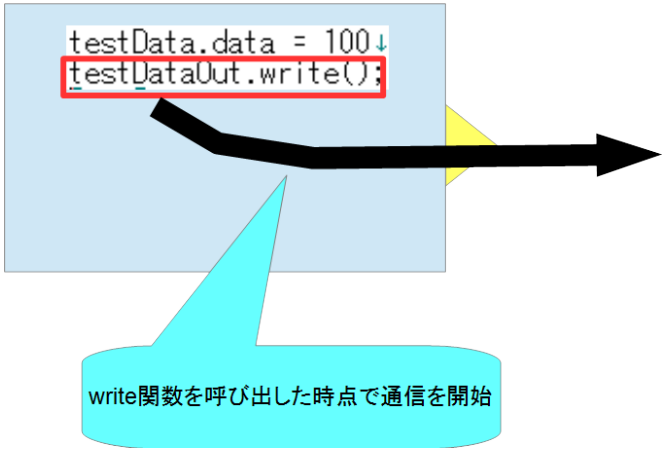


  - Pull

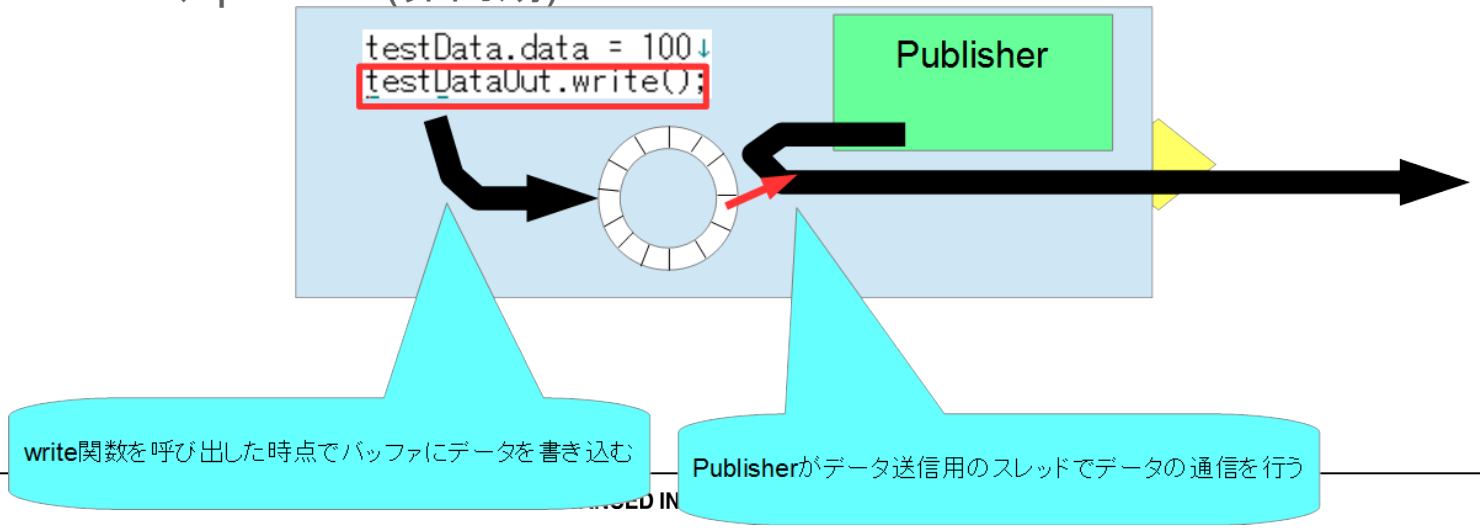


コネクタプロファイルの設定

- SubscriptionType
  - flush(同期)

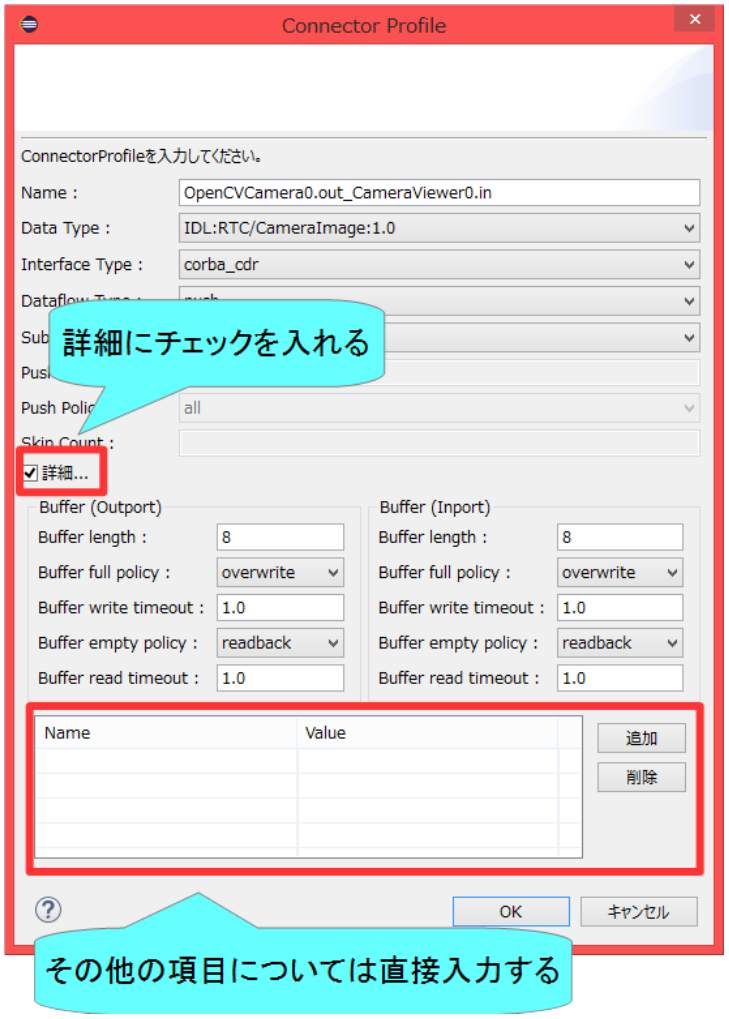


  - new、periodic(非同期)



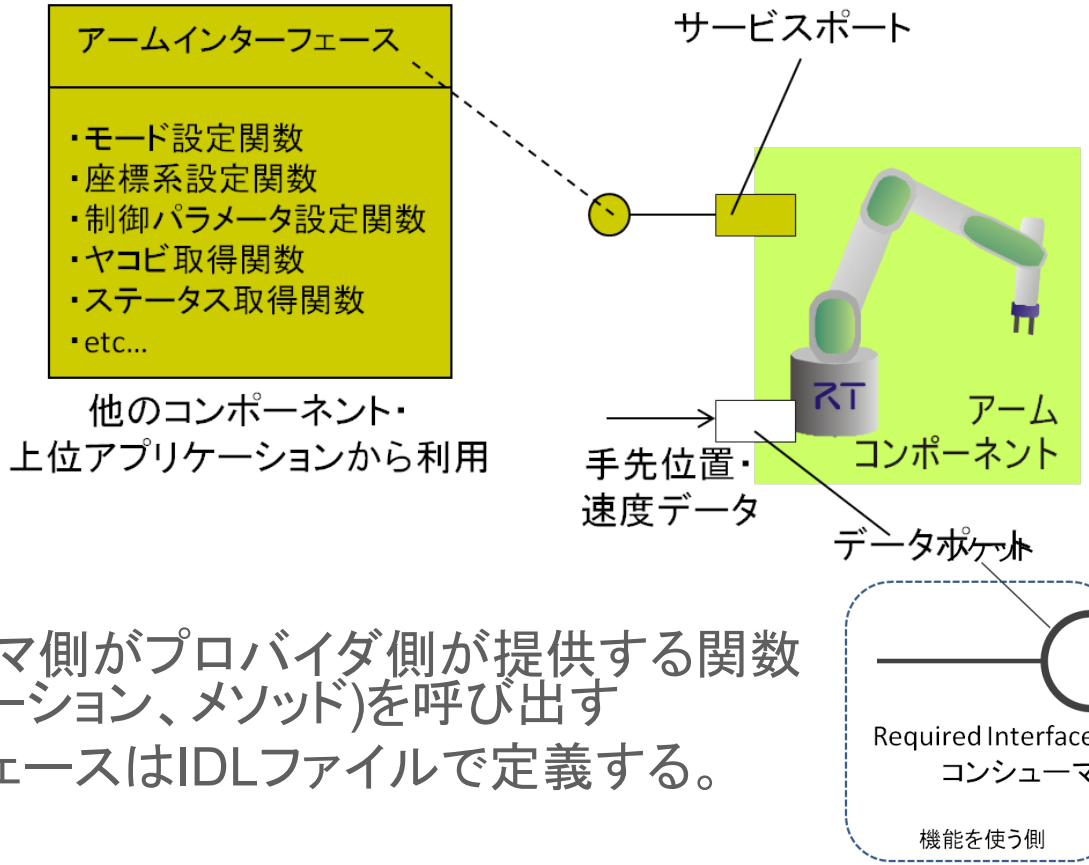
コネクタプロファイルの設定

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)



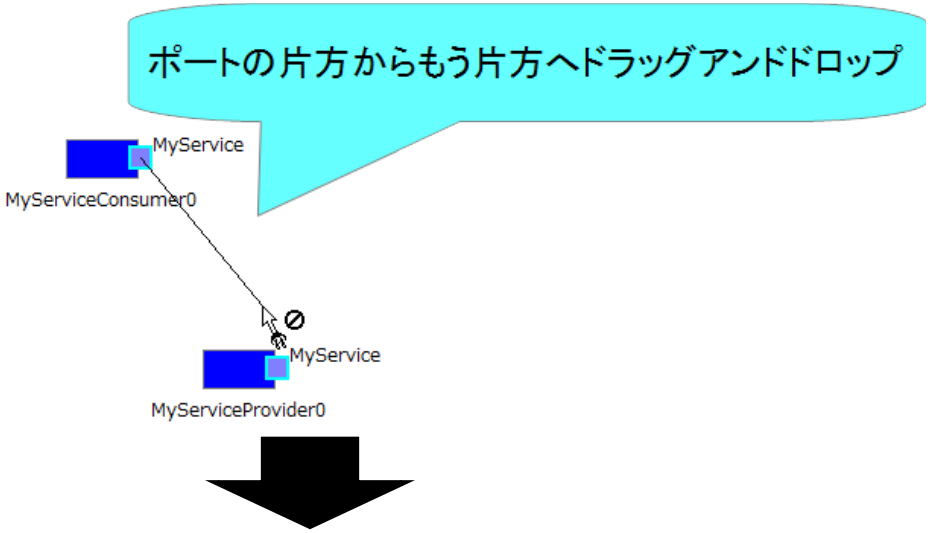
サービスポートについて

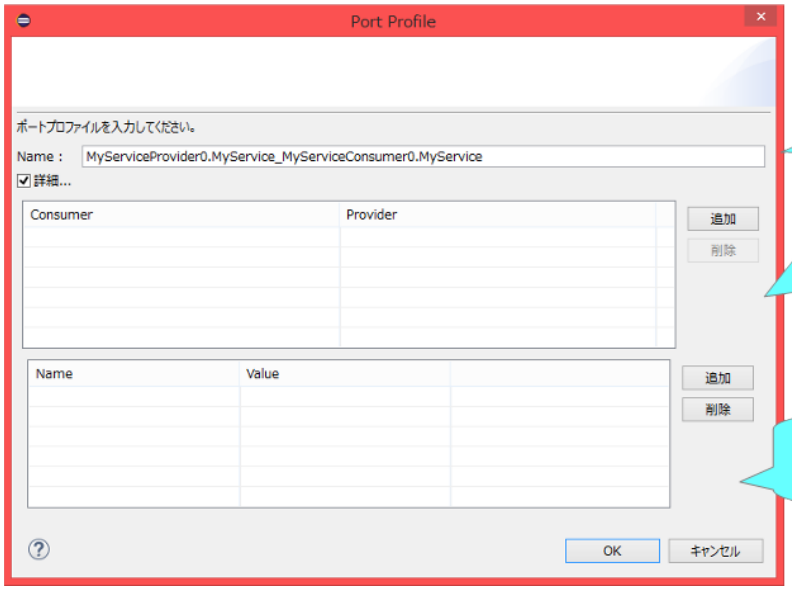
- コマンドレベルのやり取りを行うための仕組み
  - 任意のタイミングで操作を行いたい時などに使用
    - 例えばロボットアームのサーボを停止させる、ハンドを閉じる等



- 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





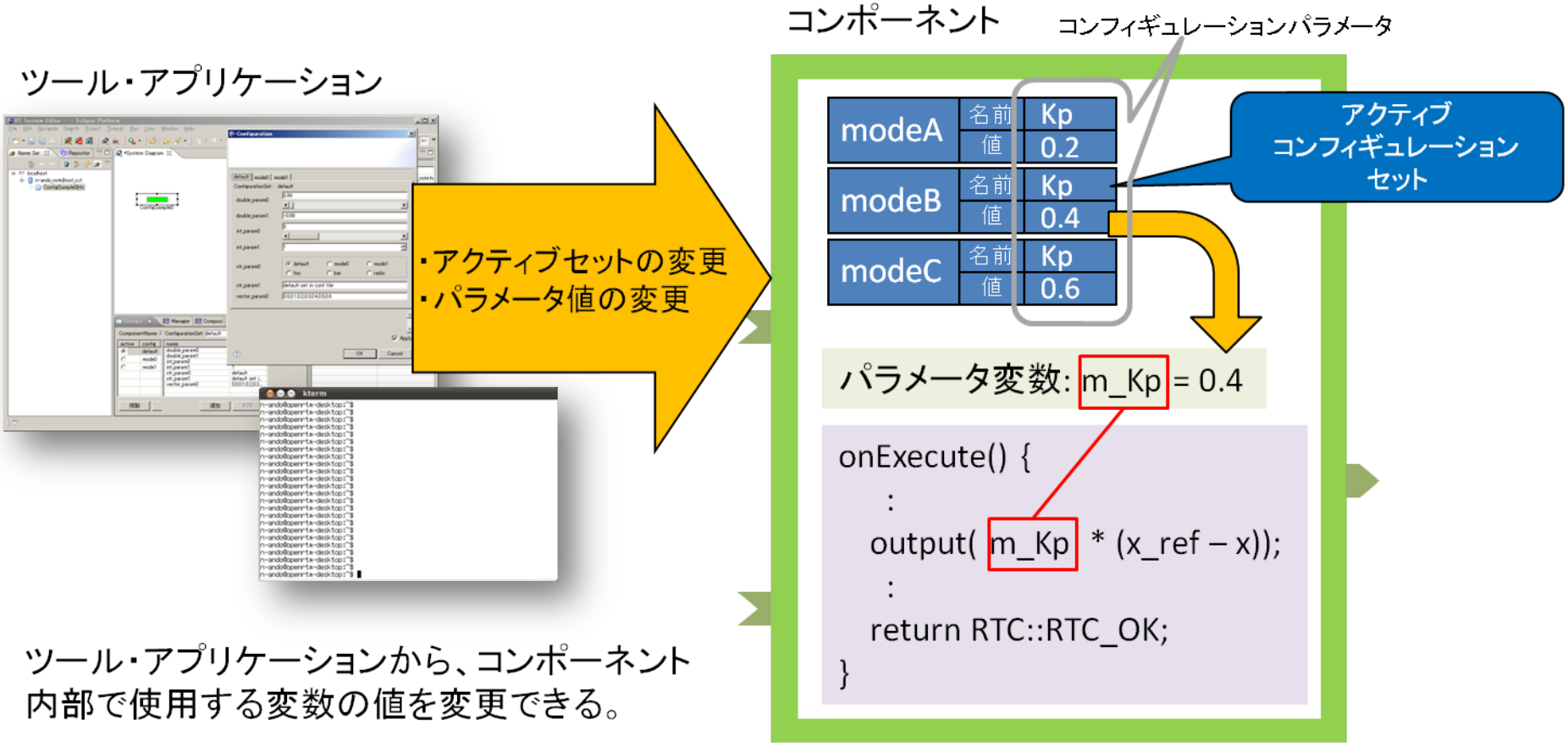
名前の設定

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

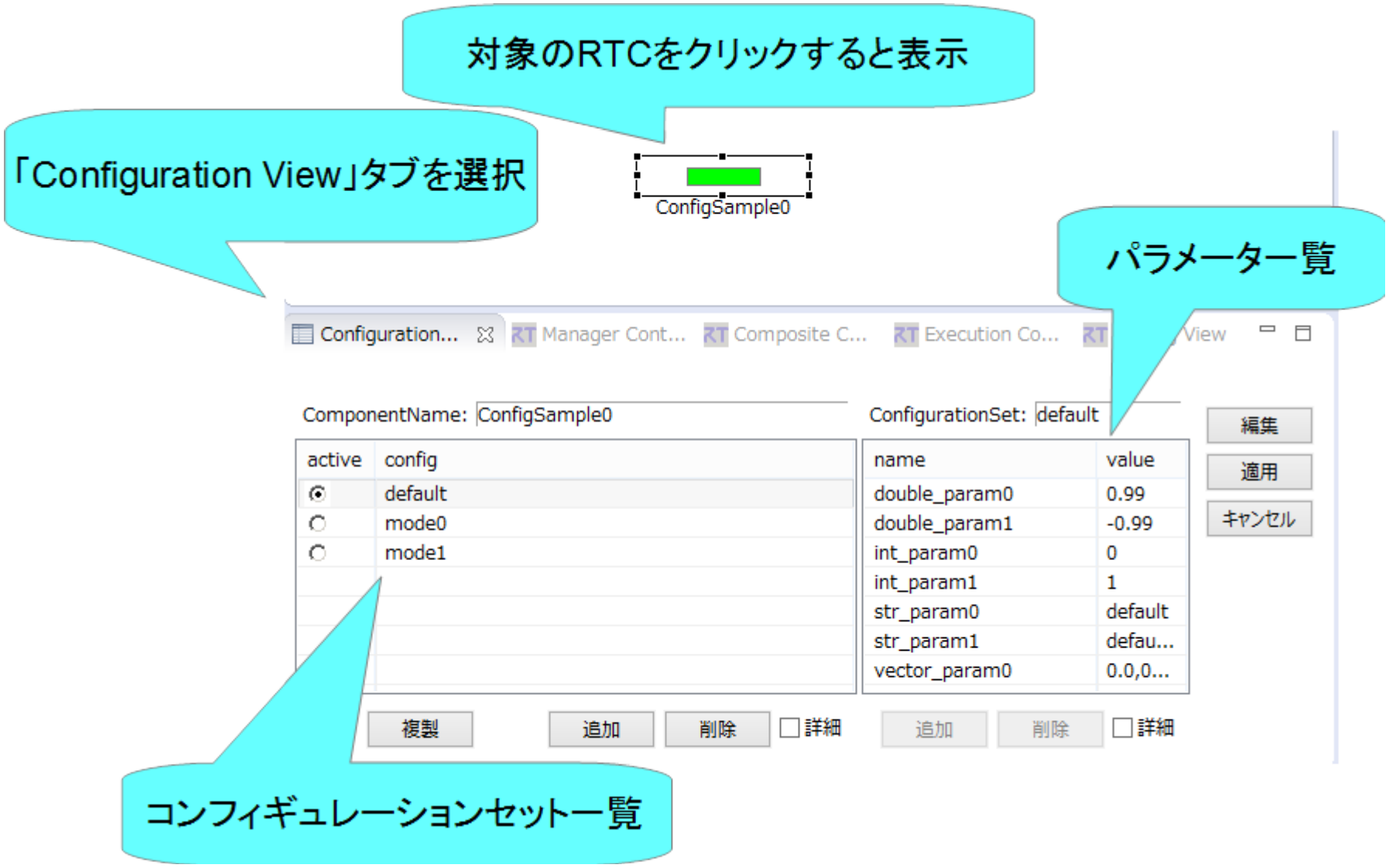
コンフィギュレーションパラメータについて

- 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



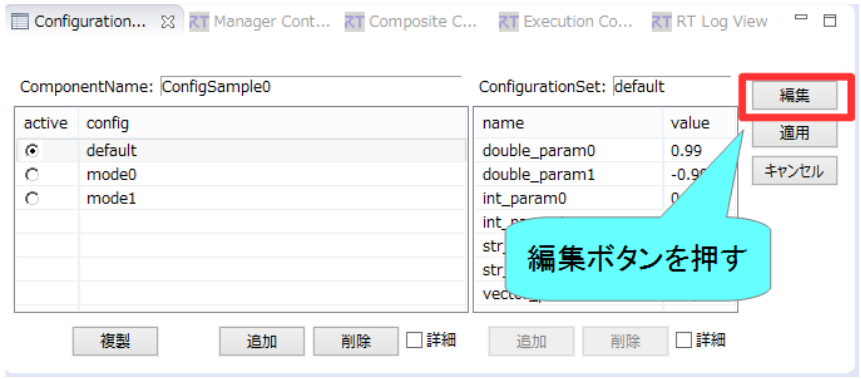
ツール・アプリケーションから、コンポーネント内部で使用する変数の値を変更できる。

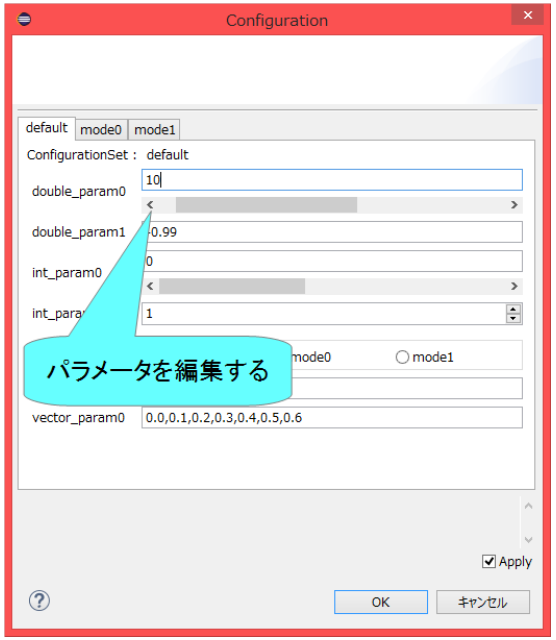
コンフィギュレーションパラメータの設定



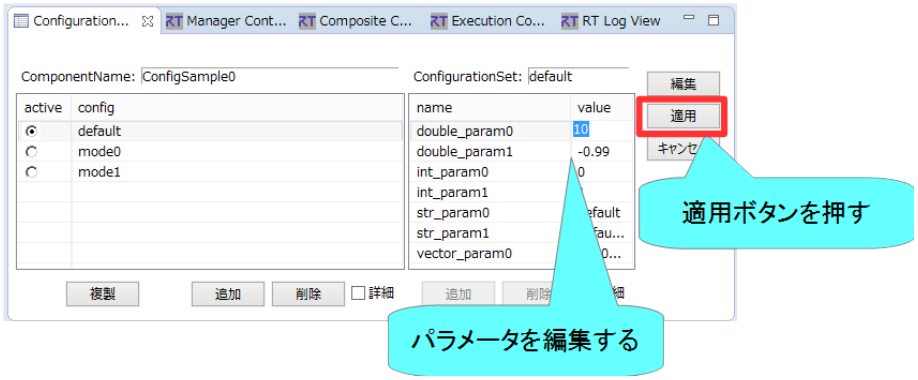
コンフィギュレーションパラメータの設定

- 方法1



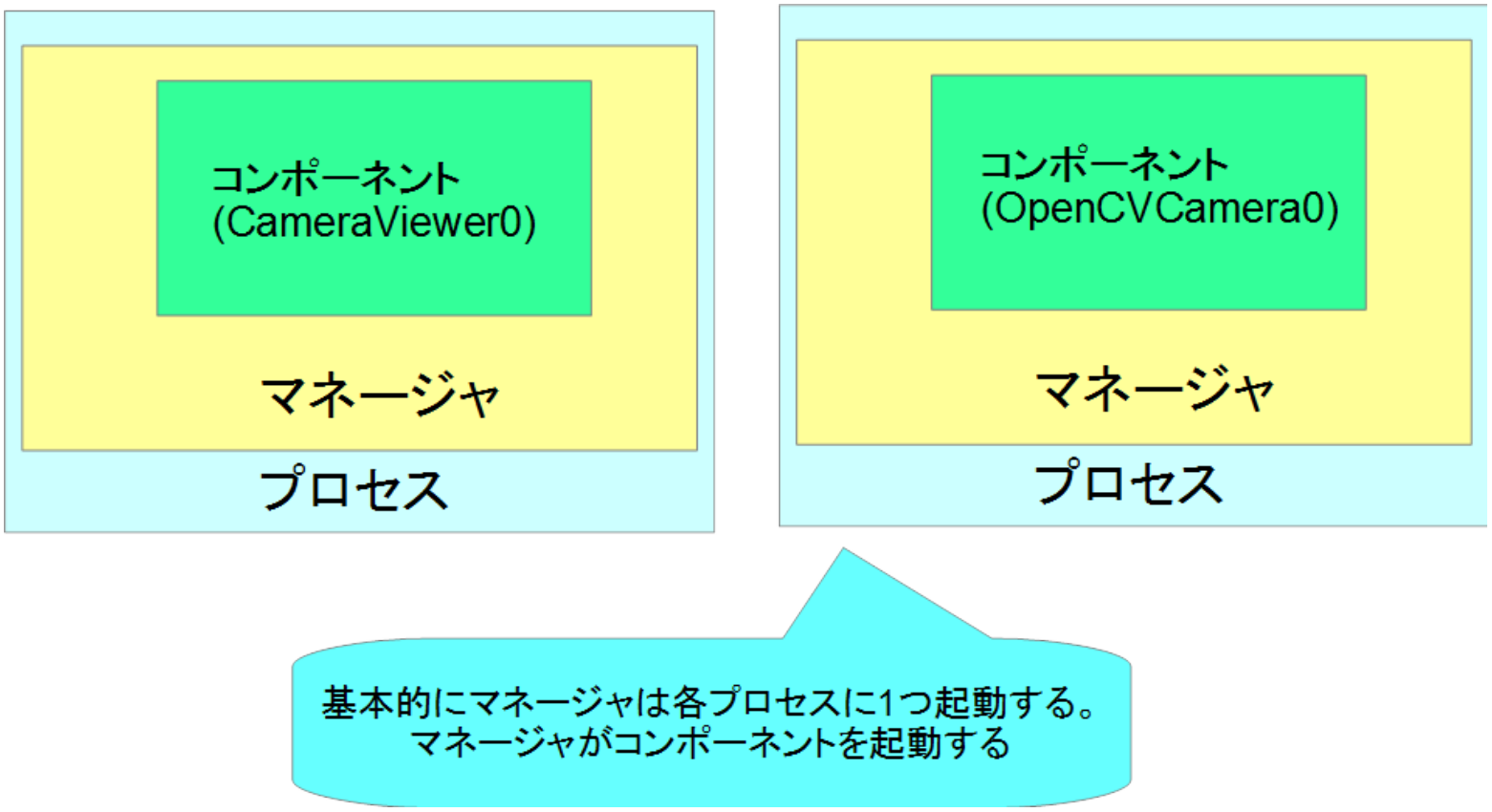


- 方法2

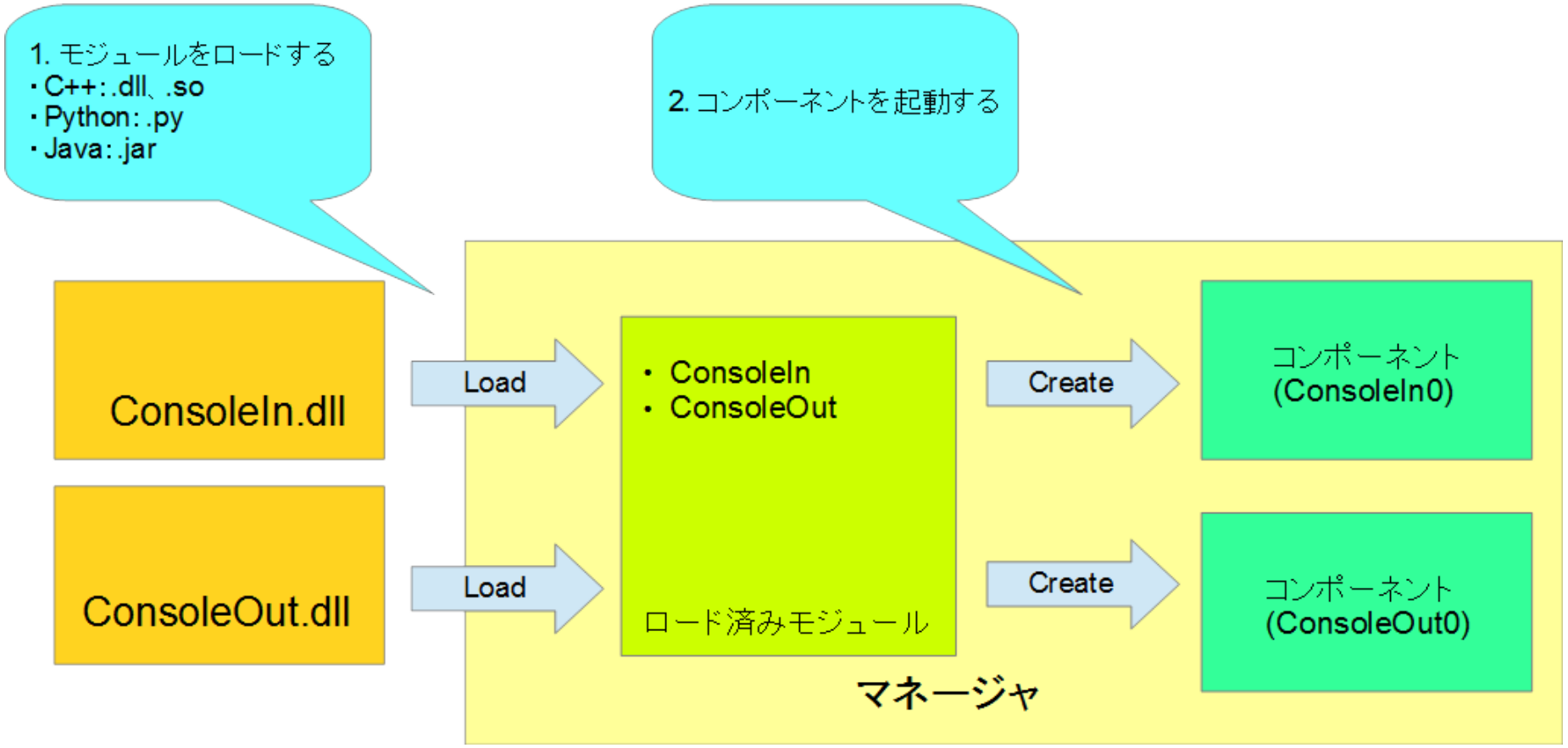


マネージャの操作

- 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

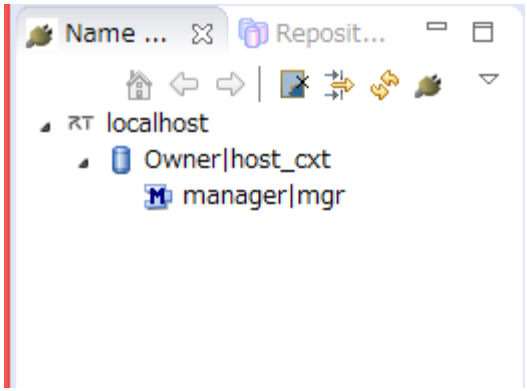


マネージャの操作


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
ConsoleIn.dll
ConsoleOut.dll
Load
Load
・ConsoleIn
・ConsoleOut
ロード済みモジュール
Create
Create
コンポーネント
(ConsoleIn0)
コンポーネント
(ConsoleOut0)
マネージャ

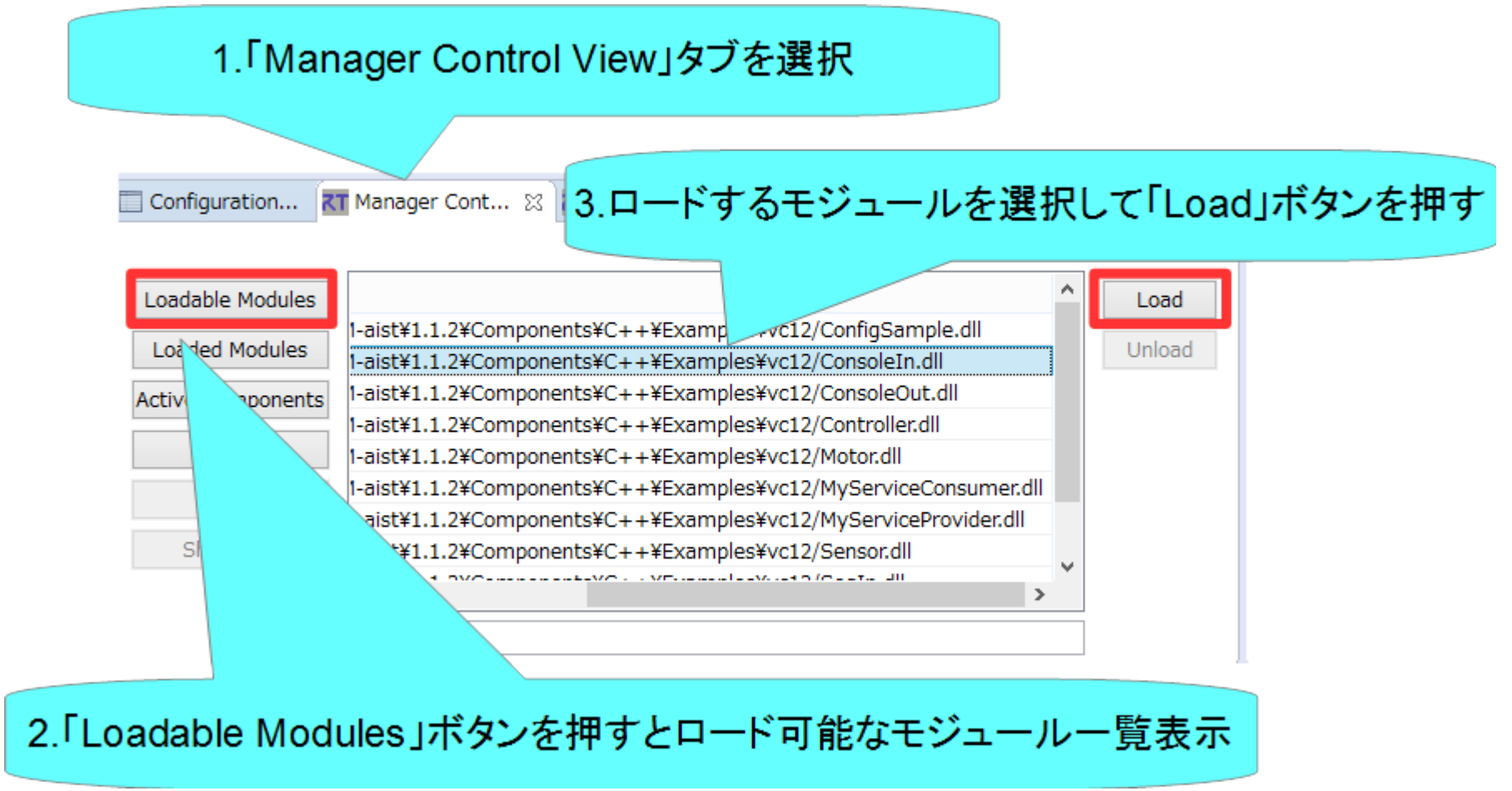
マネージャの操作

- 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¥¥1.1.2¥¥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

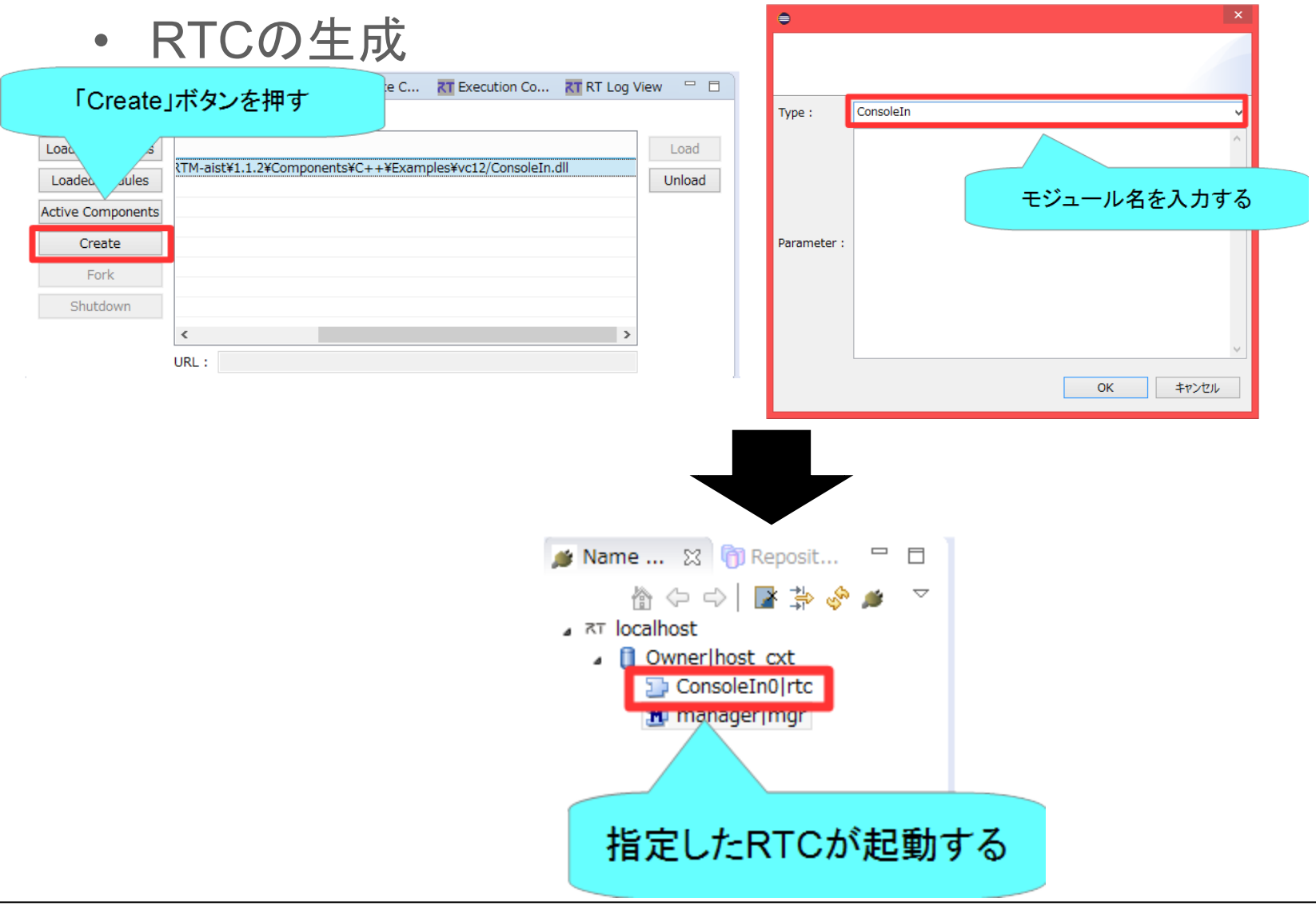


マネージャの操作

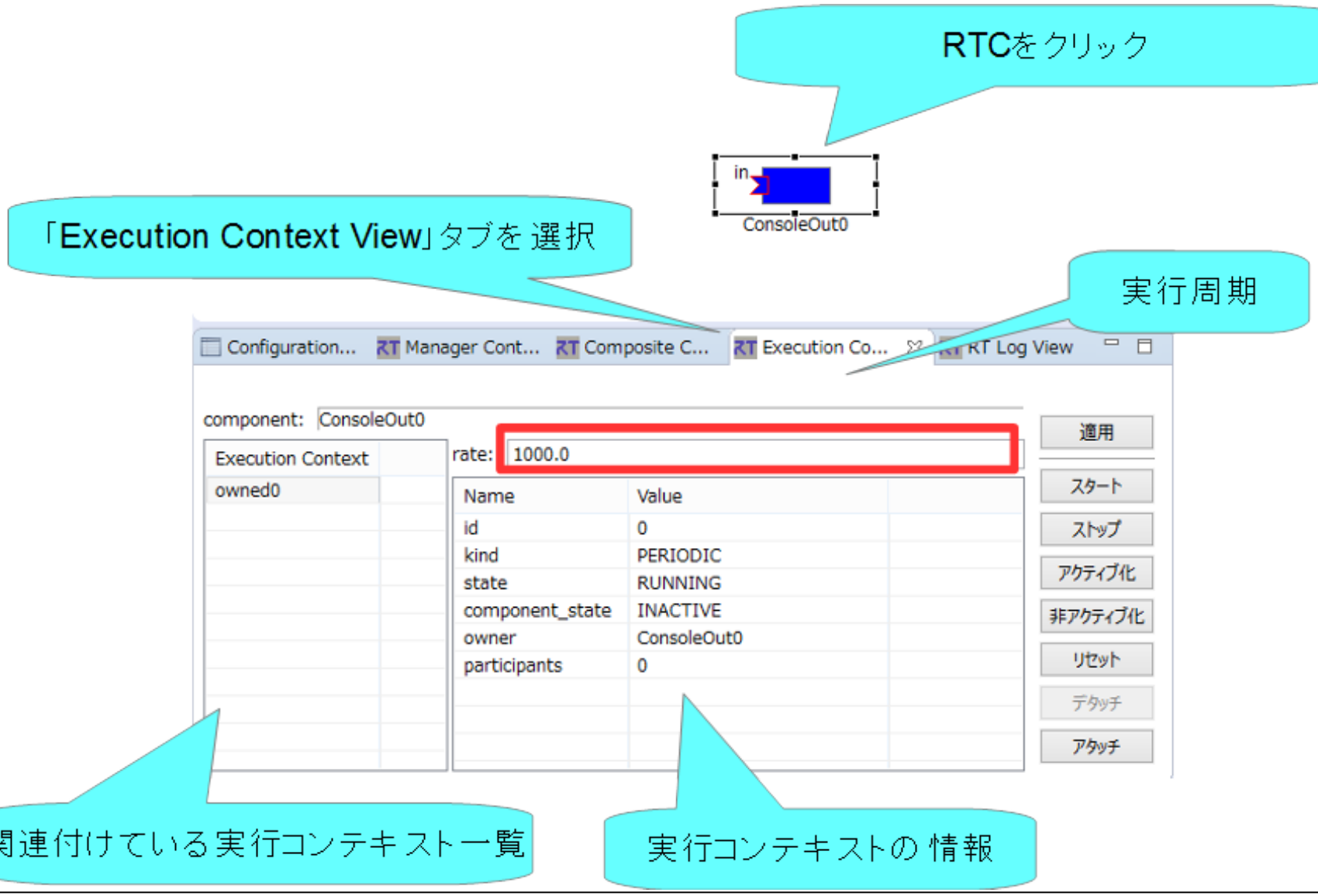
- モジュールのロード

マネージャの操作

- RTCの生成

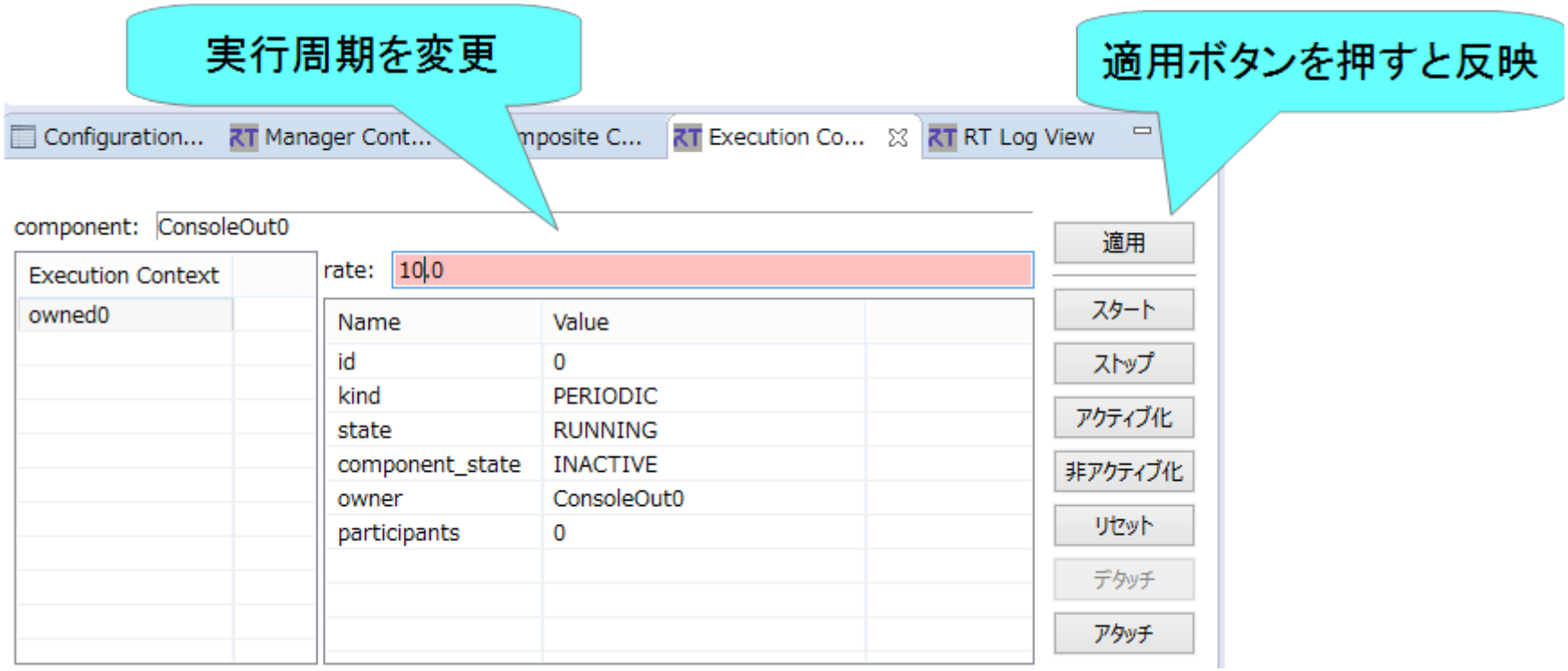


実行コンテキストの操作



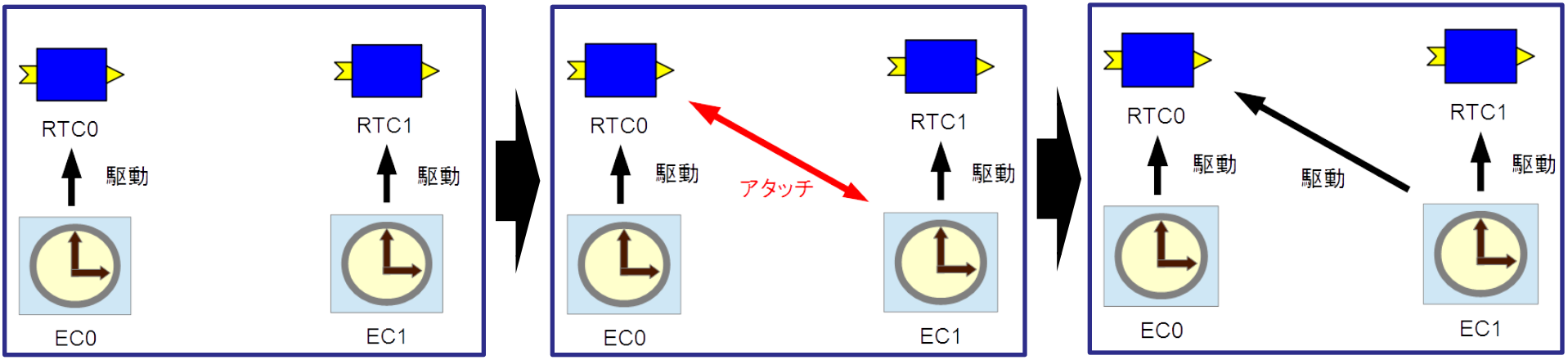
実行コンテキストの操作

- 実行周期の設定



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



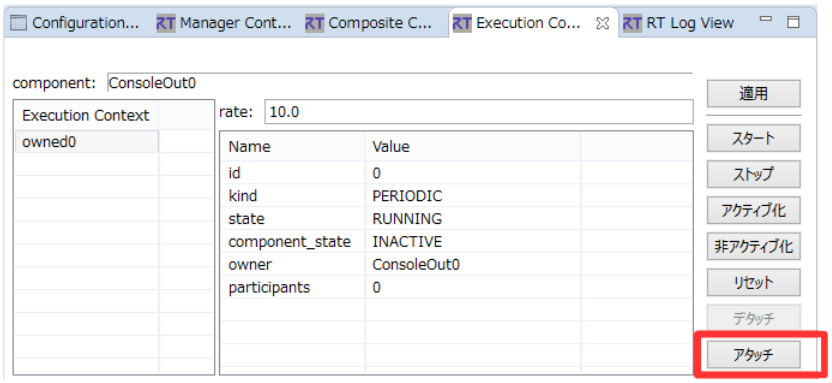
RTC0とRTC1は別々の実行コンテキストで駆動

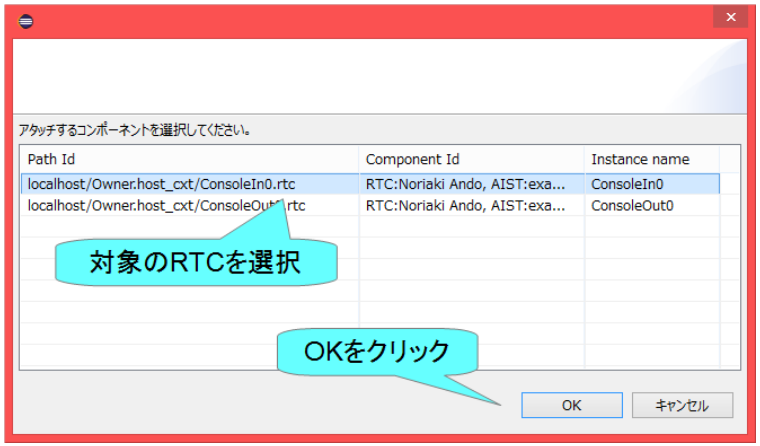
RTC0とEC1を関連付ける

EC1がRTC0も駆動する

実行コンテキストの操作

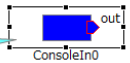
- 実行コンテキストの関連付け



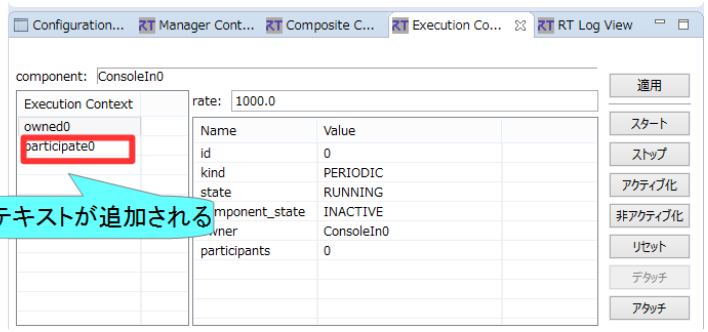


アタッチボタンを押す

対象のRTCをクリック



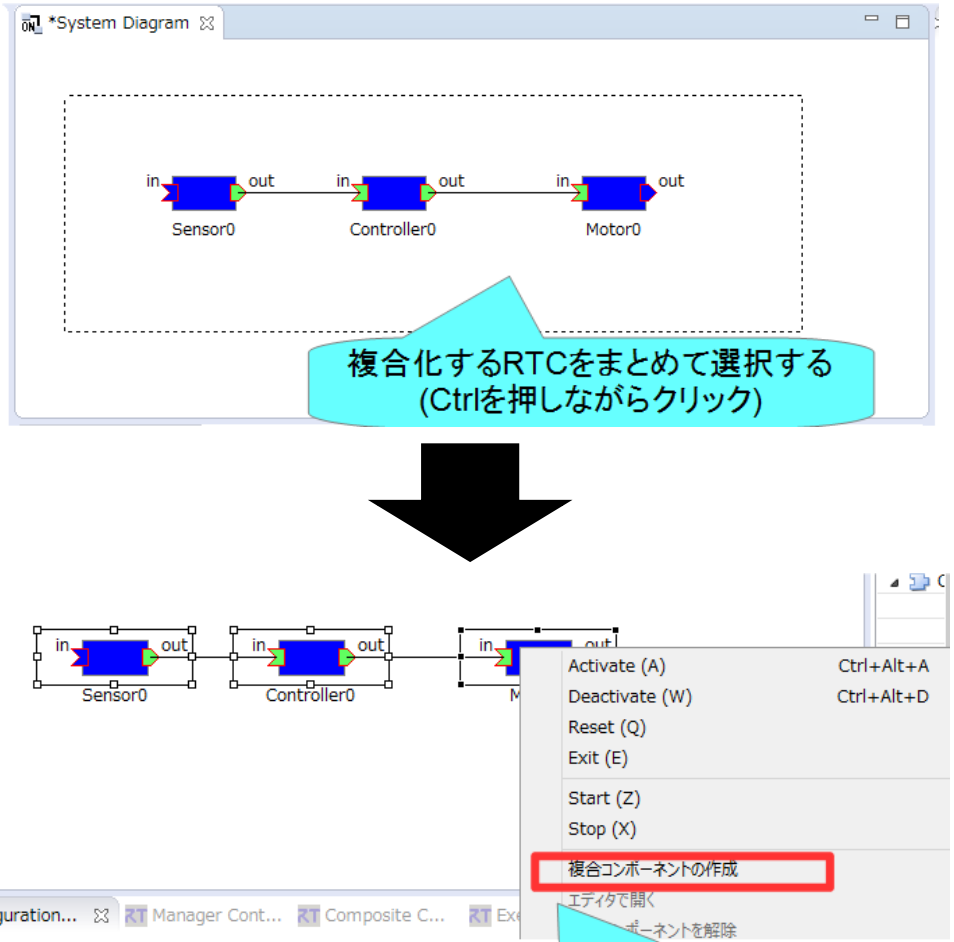
ConsoleOut0



実行コンテキストが追加される

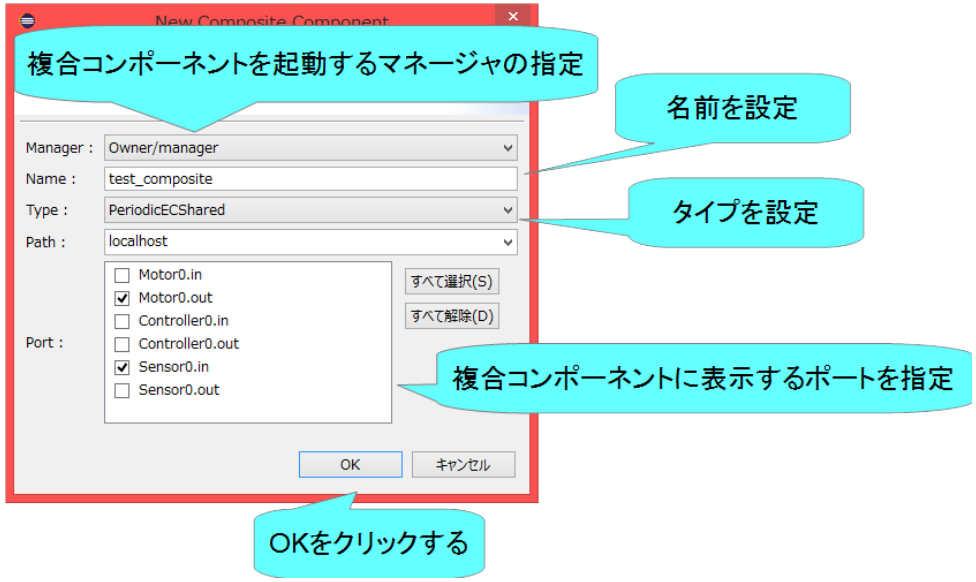
複合コンポーネントの操作

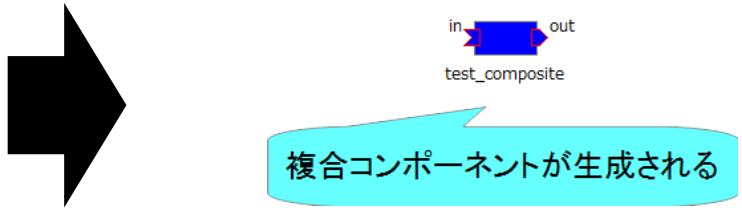
- 複合コンポーネントの生成



複合コンポーネントの操作

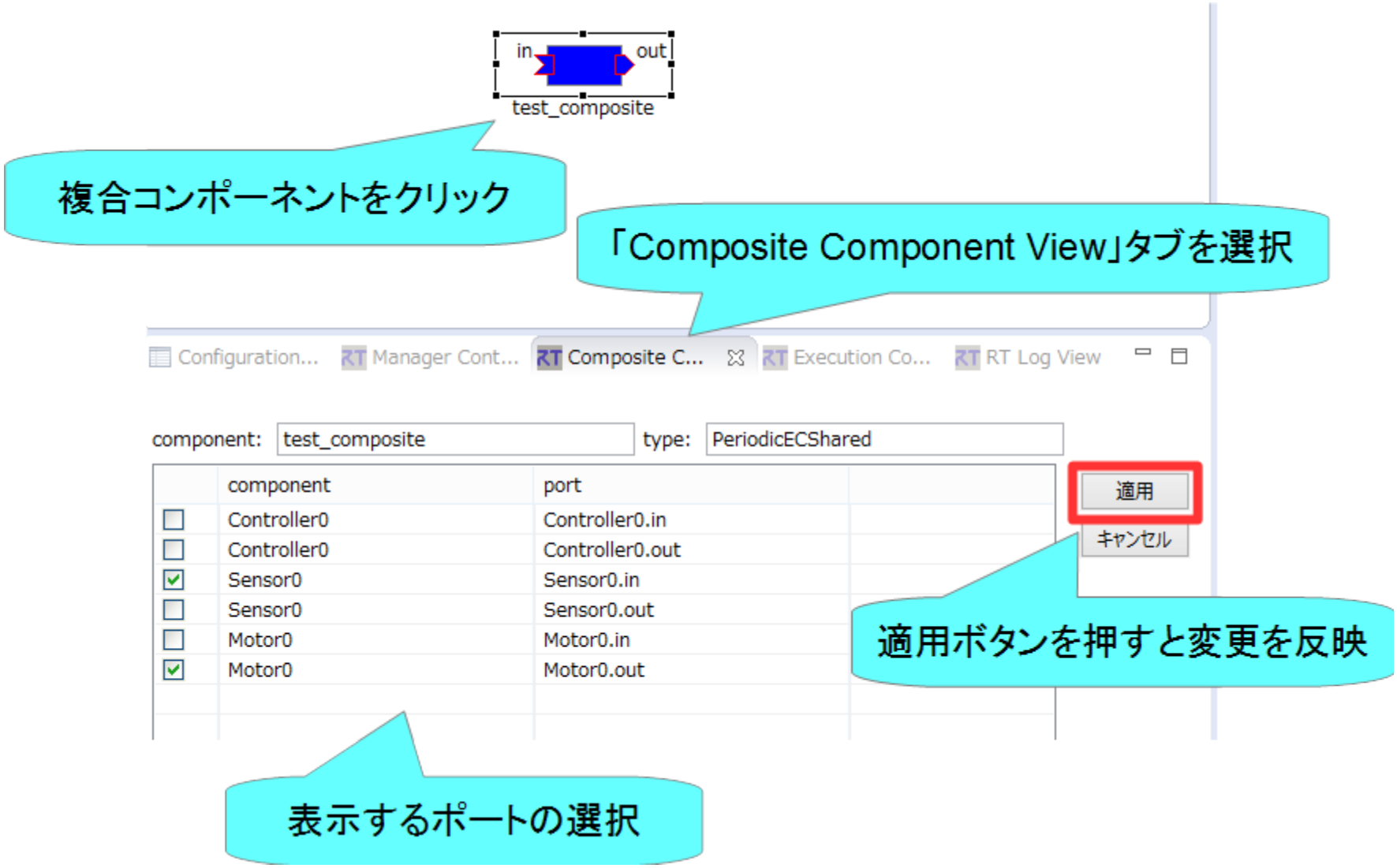
- 複合コンポーネントの生成



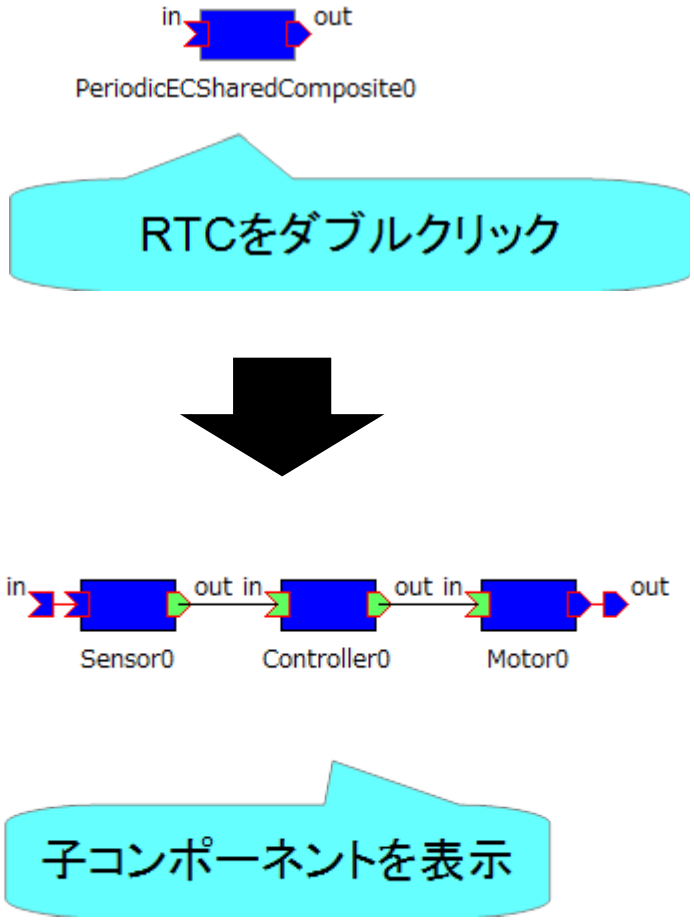


- Type
  - 以下の3種類から選択可能
    - PeriodicECShared
      - 実行コンテキストの共有
    - PeriodicStateShared
      - 実行コンテキスト、状態の共有
    - Grouping
      - グループ化のみ

複合コンポーネントの操作

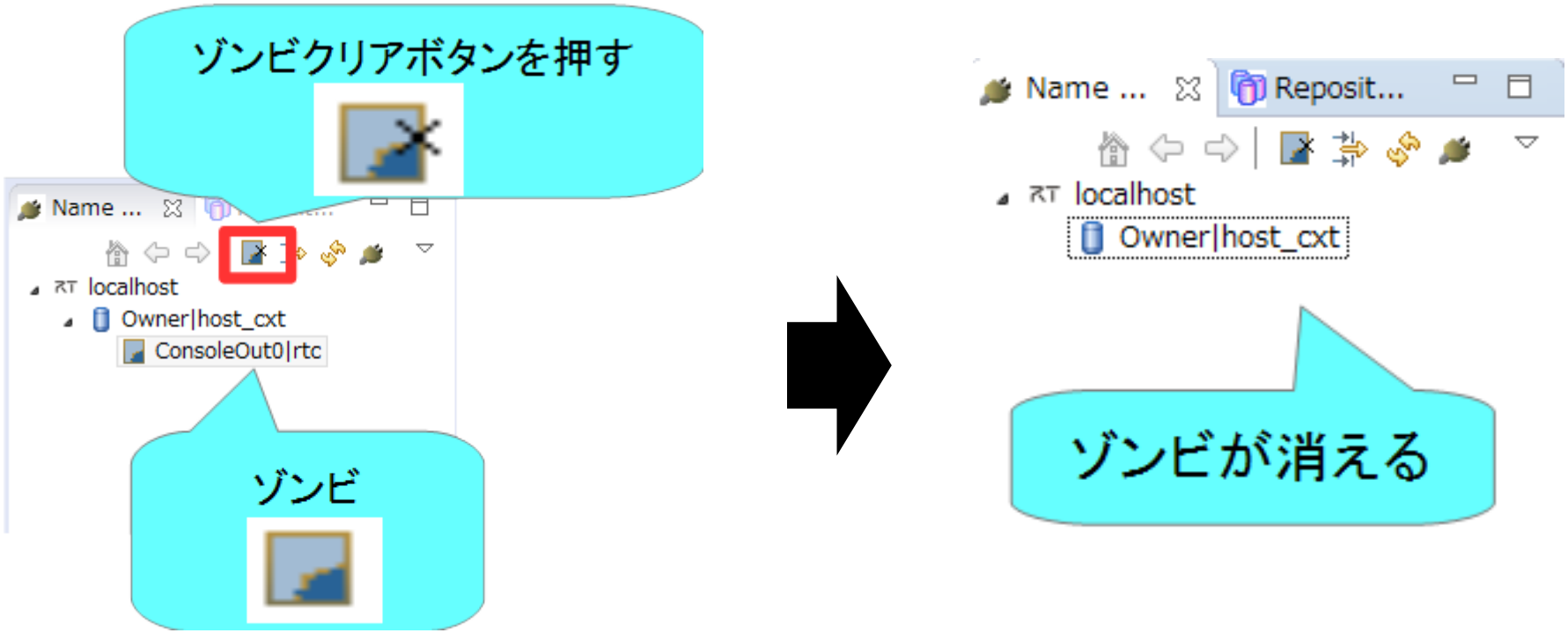


複合コンポーネントの操作



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



RT System Editorに関する設定