RTミドルウェアによるロボットプログラミング技術

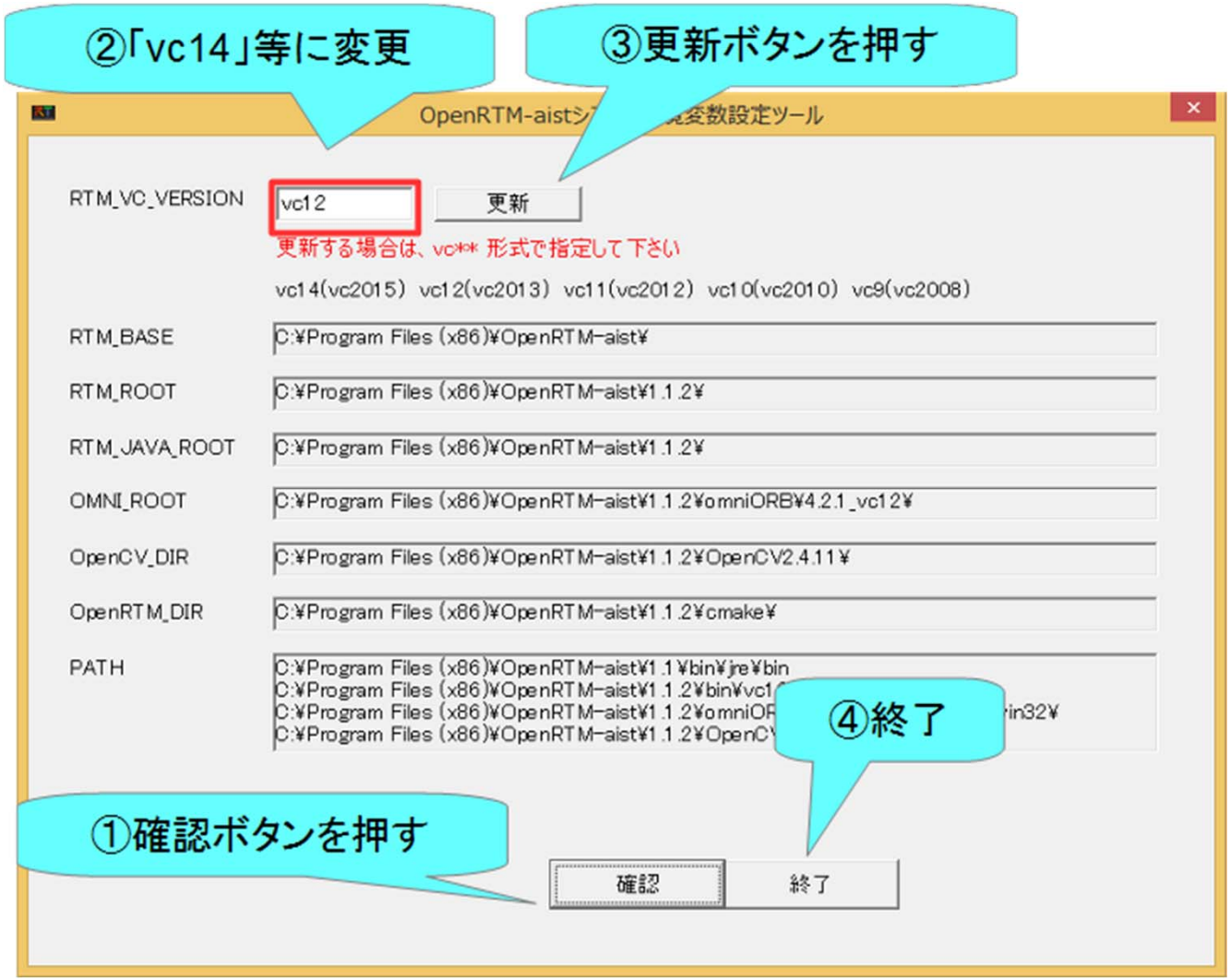
5．総合演習

インストールの確認(Windows)

- OpenRTM-aist
  - OpenRTM-aist-1.1.2-RELEASE_x86.msi
  - インストール後に再起動する(2回再起動を必要とする環境もある)
  - Visual Studio 2013以外(2010、2012、2015)を使用する場合は環境変数を変更(個別に対応します)
    - 「RTM_VC_VERSION」をvc10、vc11、vc13
    - ツールで変更可能(OpenRTM-aistトップページ→ダウンロード→OpenRTM-aist (C++版))→1,1,2-RELEASE→ツールを使った「 RTM_VC_VERSION 」設定手順
- Python
  - python-2.7.10.msi
    - 2.7.11は不具合が発生するため非推奨
  - ※OpenRTM-aistの32bit版をインストールする場合Pythonも32bit版をインストールする。
    OpenRTM-aistの64bitをインストールする場合はPythonも64bit版をインストールする。
- PyYAML
  - PyYAML-3.11.win32-py2.7.exe
- CMake
  - cmake-3.5.2-win32-x86.msi
- Doxygen
  - doxygen-1.8.11-setup.exe
- Visual Studio
  - Visual Studio 2013 Community Edition

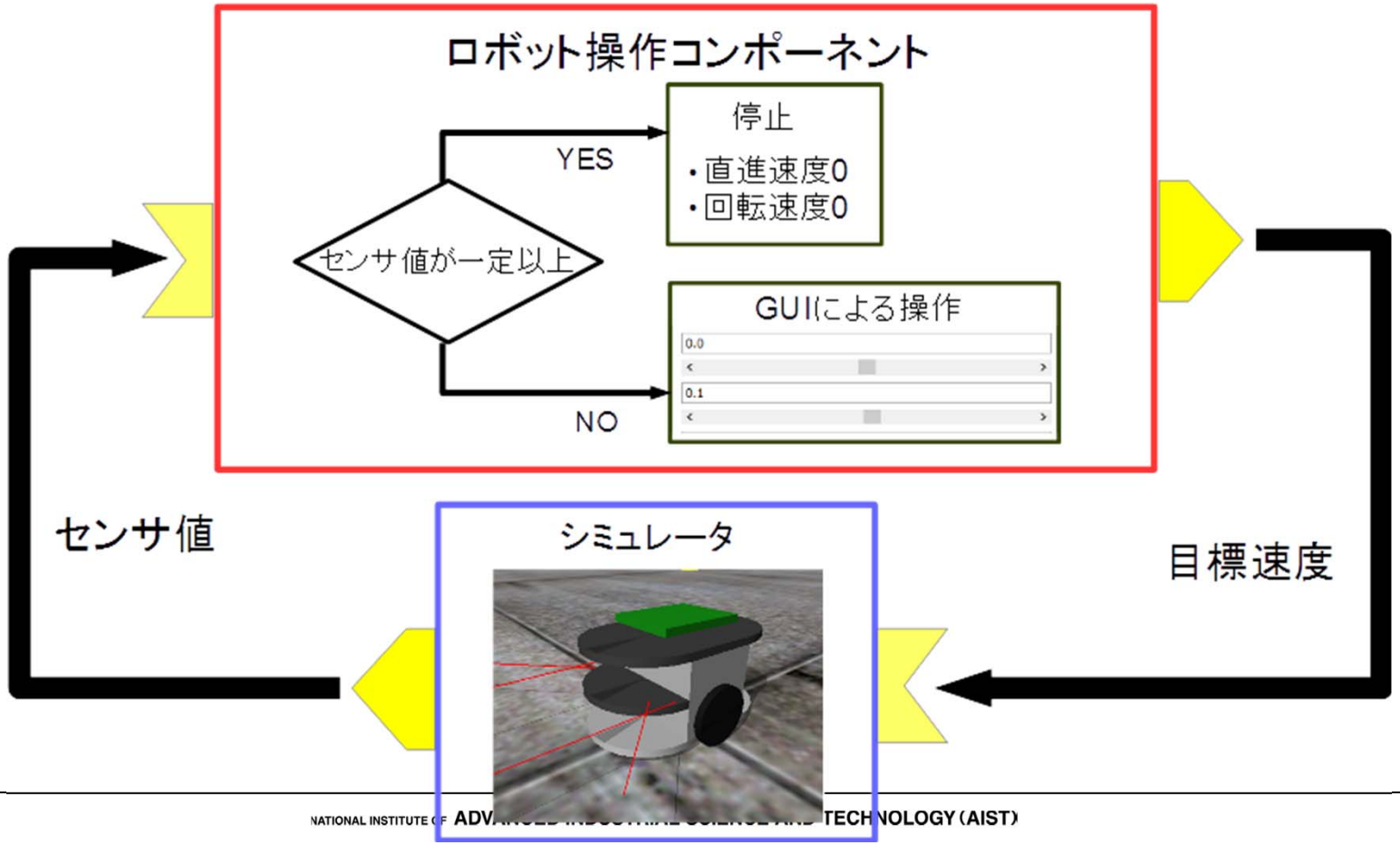
Visual Studio 2013以外を使う場合

- ツール： OpenRTMEnvTool_vc12.exe



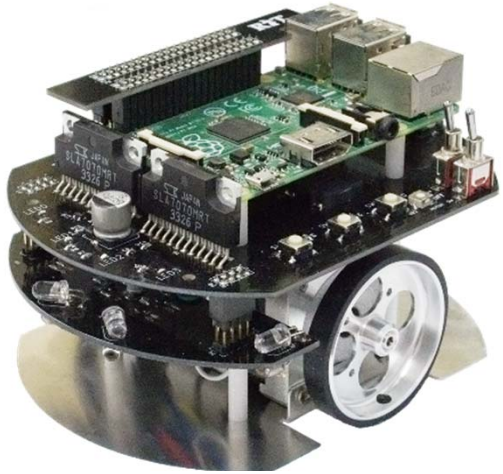
実習内容（１）

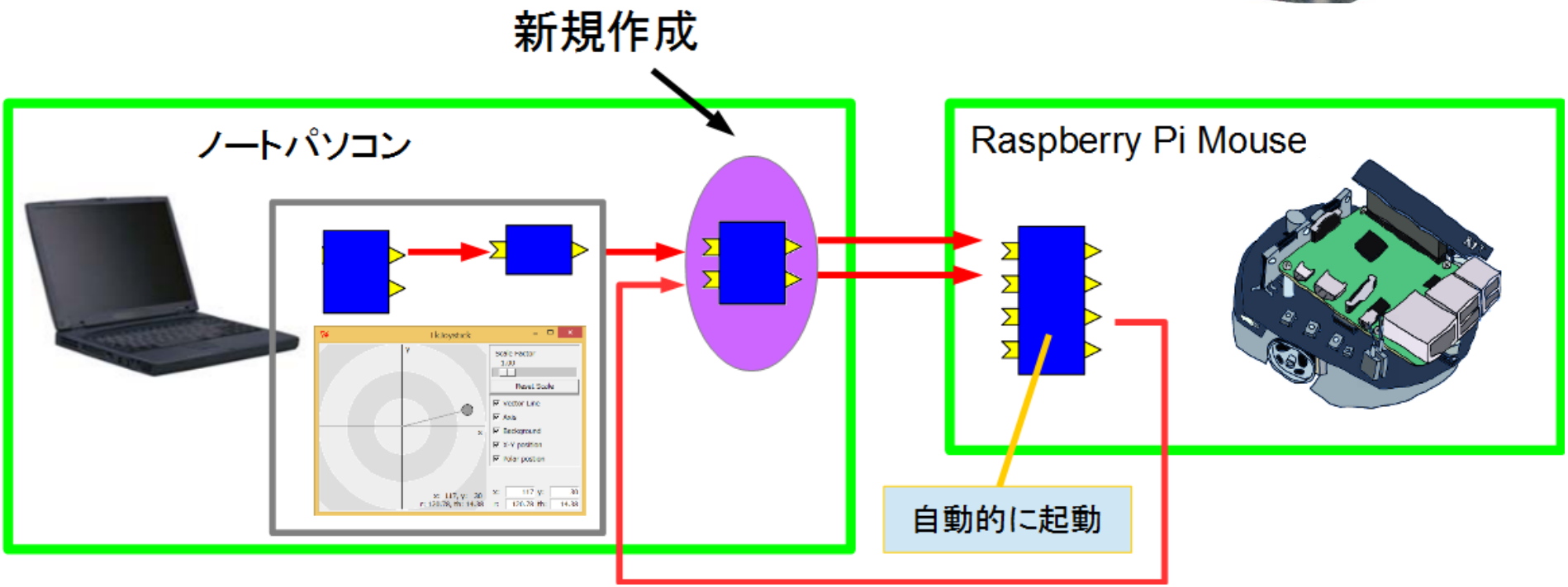
- シミュレータ上の車輪型移動ロボット(Raspberry Piマウス)の操作を行うコンポーネントの作成
  - GUIにより目標速度入力
  - センサ値が一定以上の場合に停止



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





チュートリアル

- 本日のコースのページから以下のページに移動
  - チュートリアル（RaspberryPiマウスシミュレータ）

11月7日（水）	
10:00 -12:00	**4．ロボットの運動学と制御の基礎** （1）ロボットと運動学 （2）ロボットと制御 **資料:** 171108-04.pdf **プログラム1:** arm2dof.zip **プログラム2:** joystick.zip **解答:' 181107-06.pdf** プログラム1(解答): **arm2dof.ans_.zip** プログラム2(解答): **joystick.ans_.zip**
12:00 -13:00	昼食
13:00 -16:30	5．総合演習 **（1）ロボットシステムの設計** **（2）ロボット制御プログラムの作成** **チュートリアル（Raspberry Pi Mouseシミュレータ、Windows編）** **チュートリアル（RaspberryPiマウス）** 資料:'' 181107-05.pdf

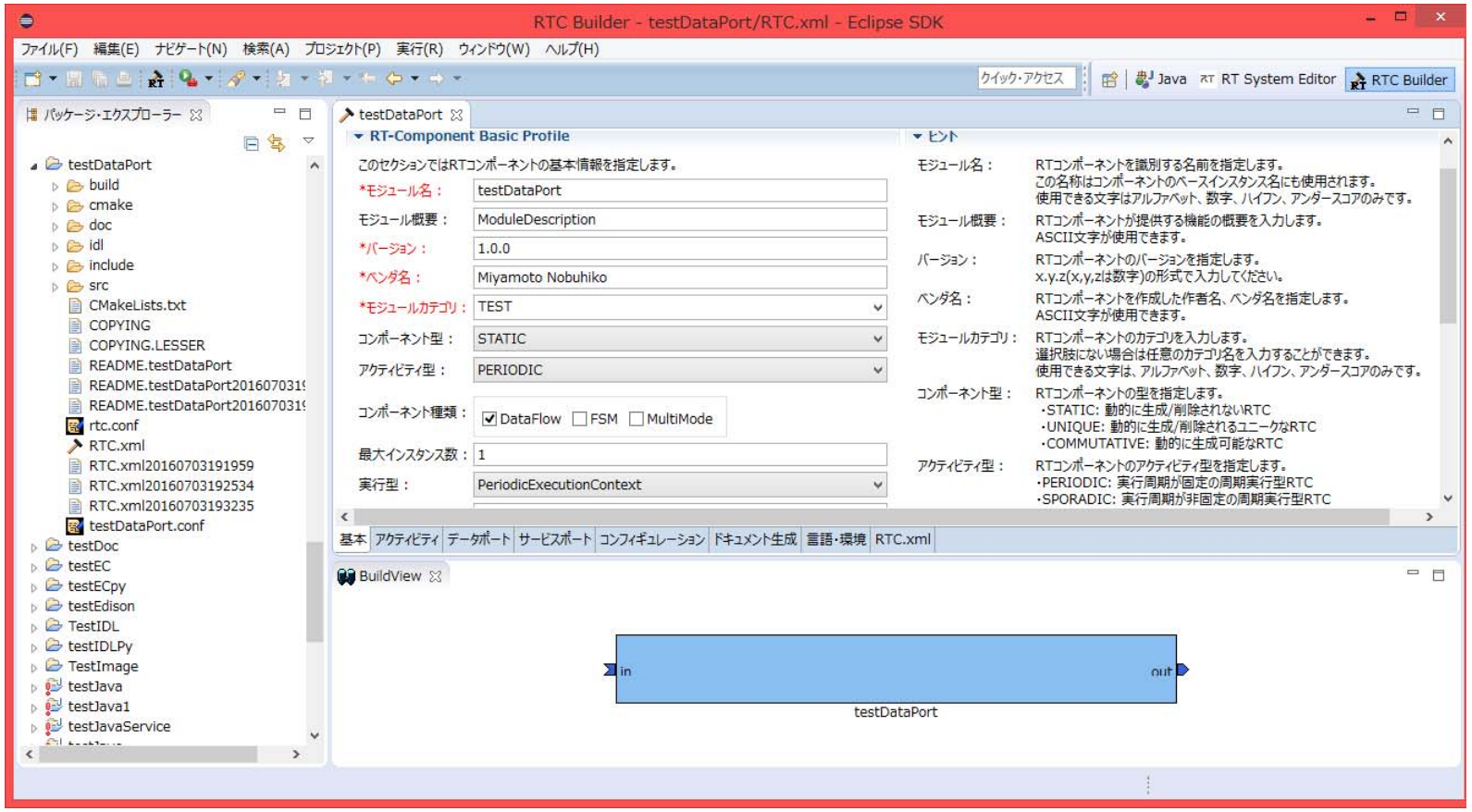
全体の手順

- 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
    - RobotController.h、 RobotController.cppの編集
  - ビルド
    - Visual Studio、Code::Blocks
- RTシステムエディタによるRTシステム作成、動作確認
  - RTシステム作成
    - データポート接続、コンフィギュレーションパラメータ設定

コンポーネント開発ツール
RTC Builderについて

RTC Builder

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



RTC Builderの起動

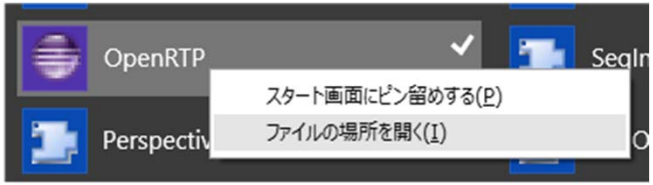
- 起動する手順
  - Windows 7
    - 「スタート」→「すべてのプログラム」→「OpenRTM-aist 1.1.2」→「Tools」→「OpenRTP」
  - Windows 8.1
    - 「スタート」→「アプリビュー(右下矢印)」→「OpenRTM-aist 1.1.2」→「OpenRTP」
    - ※同じフォルダに「RTSystemEditorRCP」がありますが、これはRTC Builderが使えないので今回は「OpenRTP」を起動してください。
  - Windows 10
    - 検索窓→「OpenRTP」
  - Ubuntu
    - Eclipseを展開したディレクトリに移動して以下のコマンド
    - $ ./openrtp

RTC Builderの起動

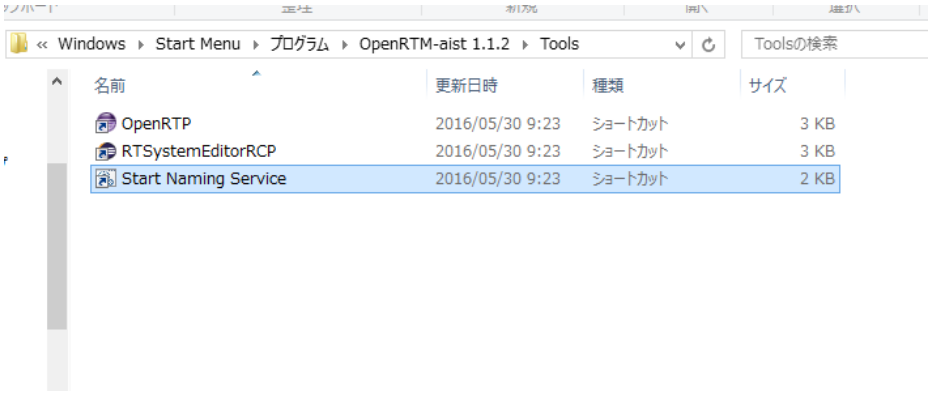
- Windows 8.1

RTC Builderの起動

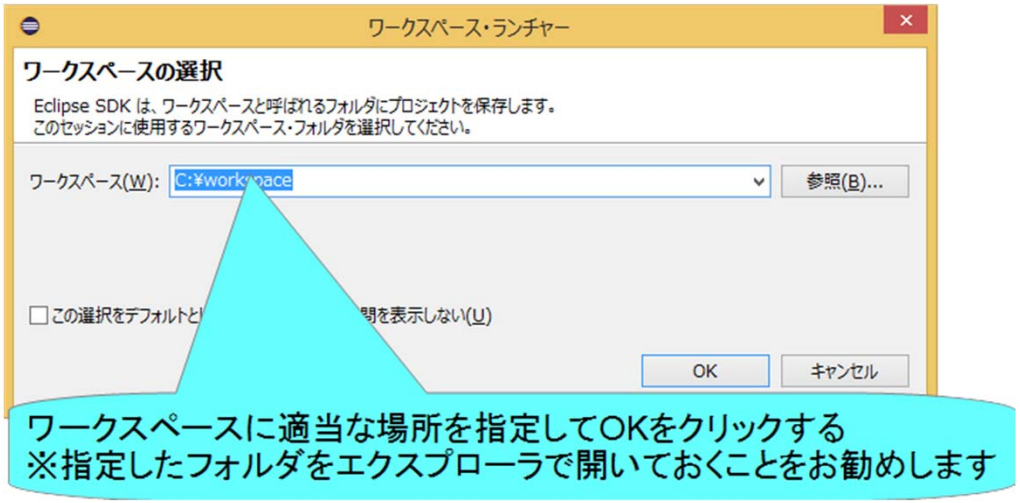
- いちいちアプリビューから起動するのは非常に手間がかかるため、以下の作業をしてスタートメニューのフォルダを開いておくことをお勧めします。



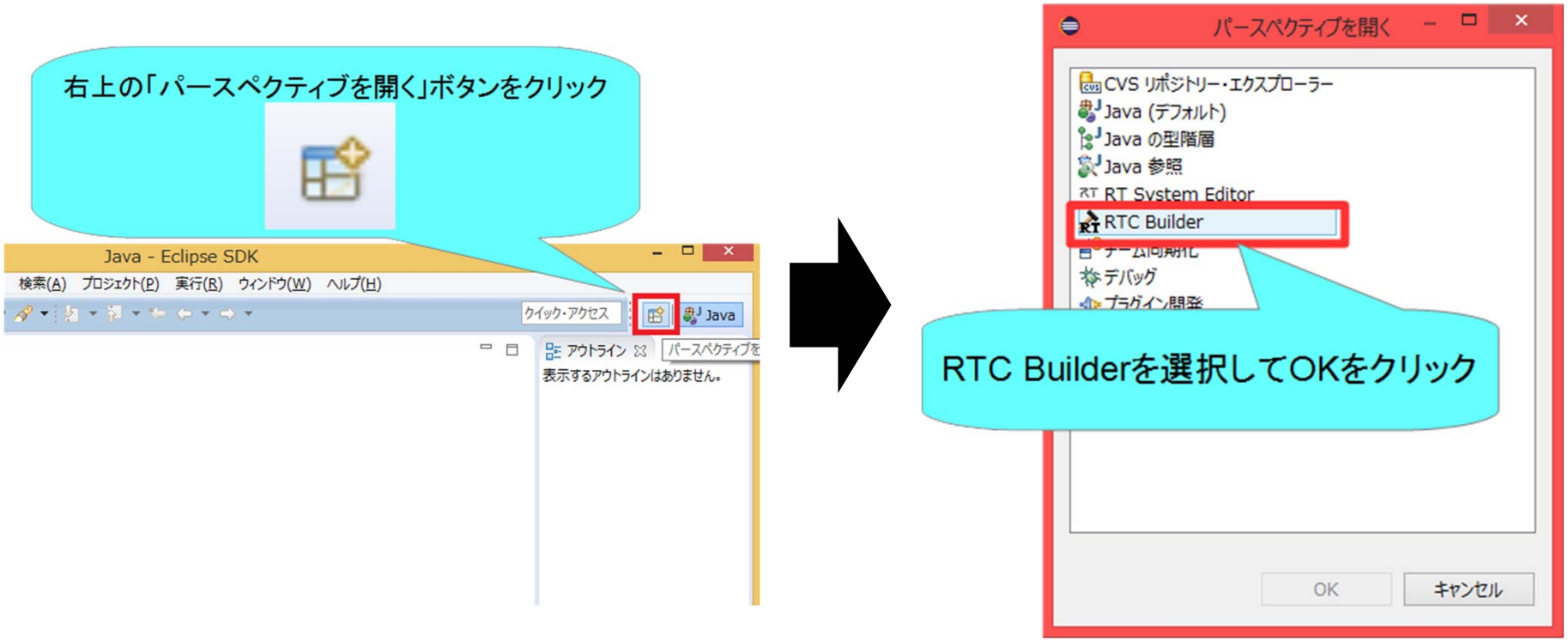
Start naming Serviceを右クリックして
「ファイルの場所を開く」を選択



RTC Builderの起動

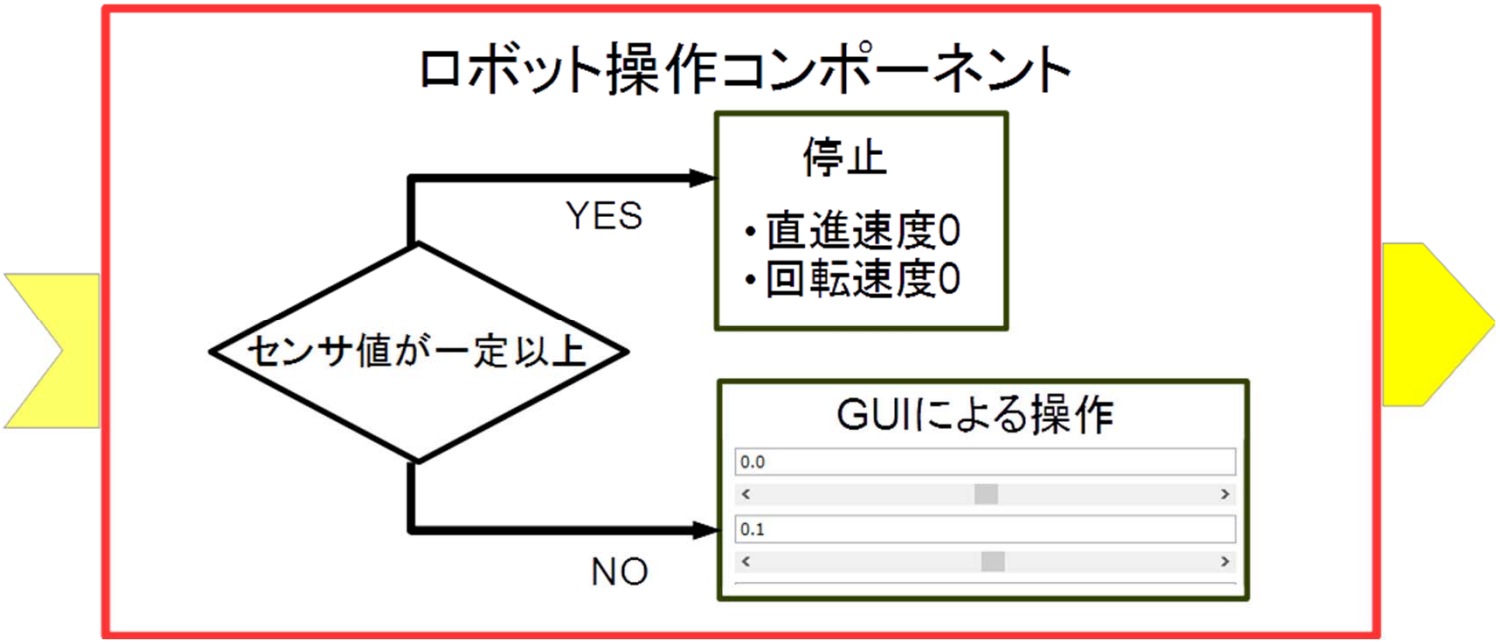


RTC Builderの起動



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



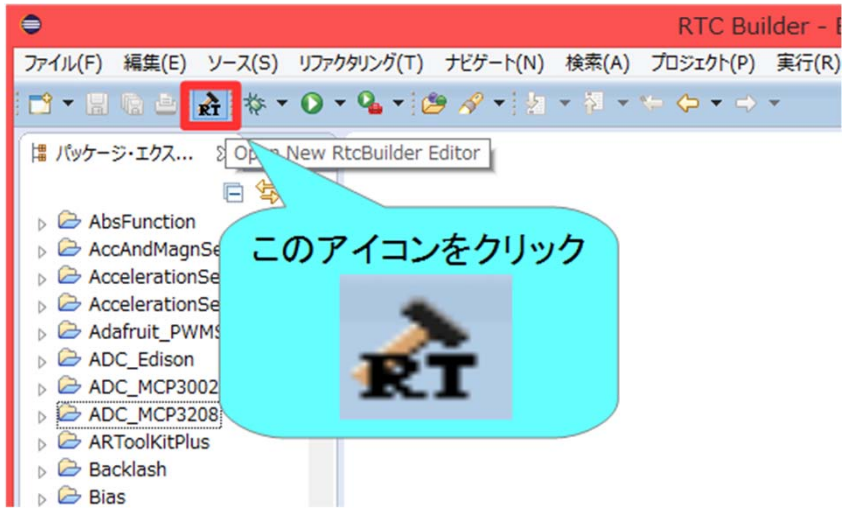
資料

- USBメモリで配布
  - 「WEBページ」フォルダのHTMLファイルを開く
    - Windows/チュートリアル(Raspberry Pi Mouse、Windows、強化月間用) _ OpenRTM-aist.html
    - Ubuntu/チュートリアル(Raspberry Pi Mouse、Ubuntu、強化月間用) _ OpenRTM-aist.html
- もしくはRTミドルウェア講習会のページからリンクをクリック
  - チュートリアル（ Raspberry Piマウスシミュレータ Windows編）
  - チュートリアル（ Raspberry Piマウスシミュレータ Linux編）

プログラム

13:00 -14:00	**第1部：RTミドルウエア: OpenRTM-aist概要** **担当**：安藤慶昭(産総研) **概要**：RTミドルウエアはロボットシステムをコンポーネント指向で構築するソフトウエアプラットフォームです。RTミドルウエアを利用することで、既存のコンポーネントを再利用し、モジュール指向の柔軟なロボットシステムを構築することができます。RTミドルウエアの産総研による実装であるOpenRTM-aistについてその概要について説明します。 **講義資料**：160705-w01.pdf
14:15 -17:00	**第2部: RTコンポーネントの作成入門** **担当**：宮本信彦(産総研) **概要**：RTシステムを設計するツールRTSystemEditorおよびRTコンポーネントを作成するツールRTCBuilderの使用方法について解説するとともに、RTCBuilderを使用したRTコンポーネントの作成方法を実習形式で体験していただきます。 チュートリアル（Raspberry Pi Mouseシミュレータ、Windows編） チュートリアル（Raspberry Pi Mouseシミュレータ、 Ubuntu編）

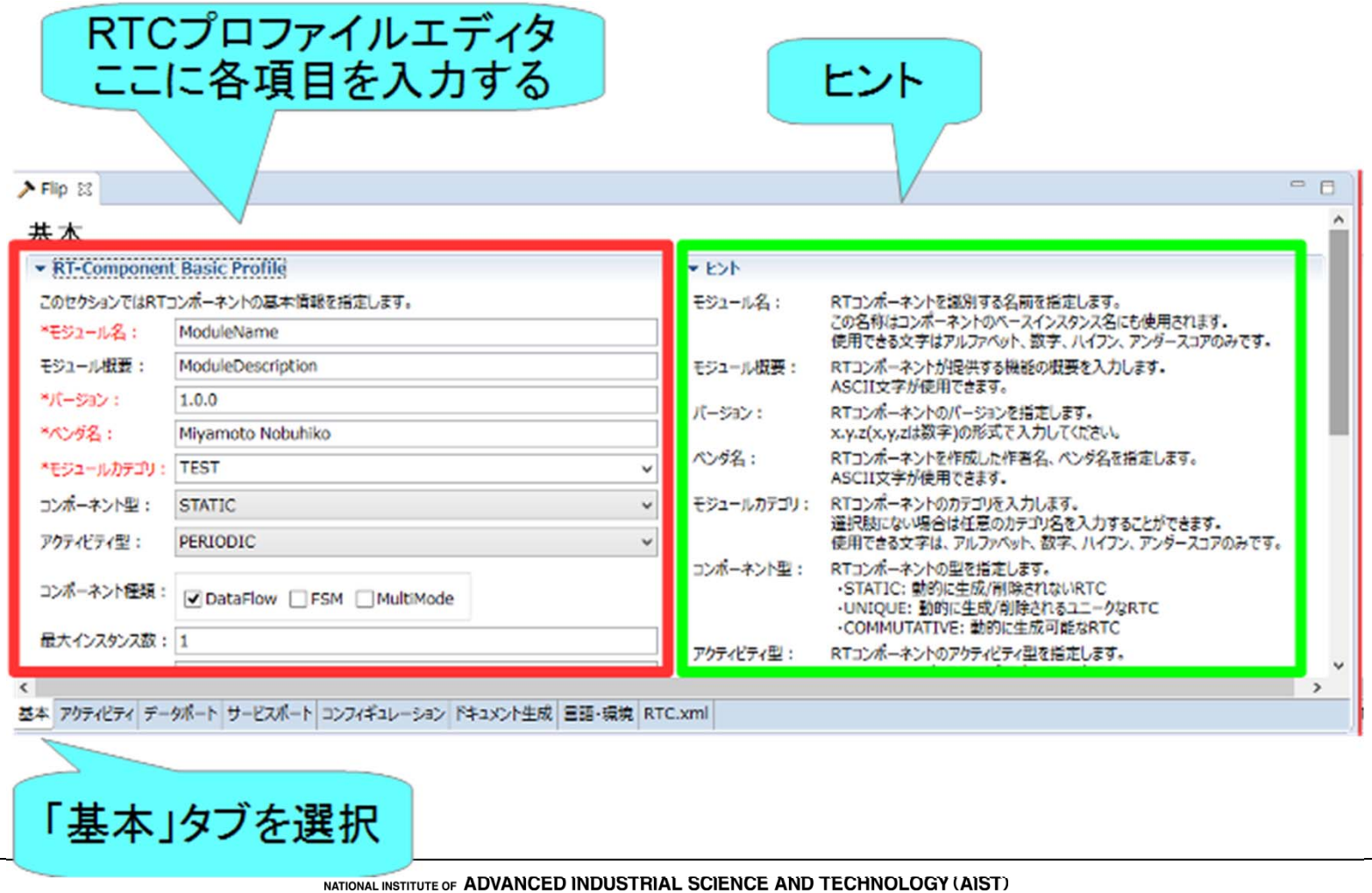
プロジェクト作成



- 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

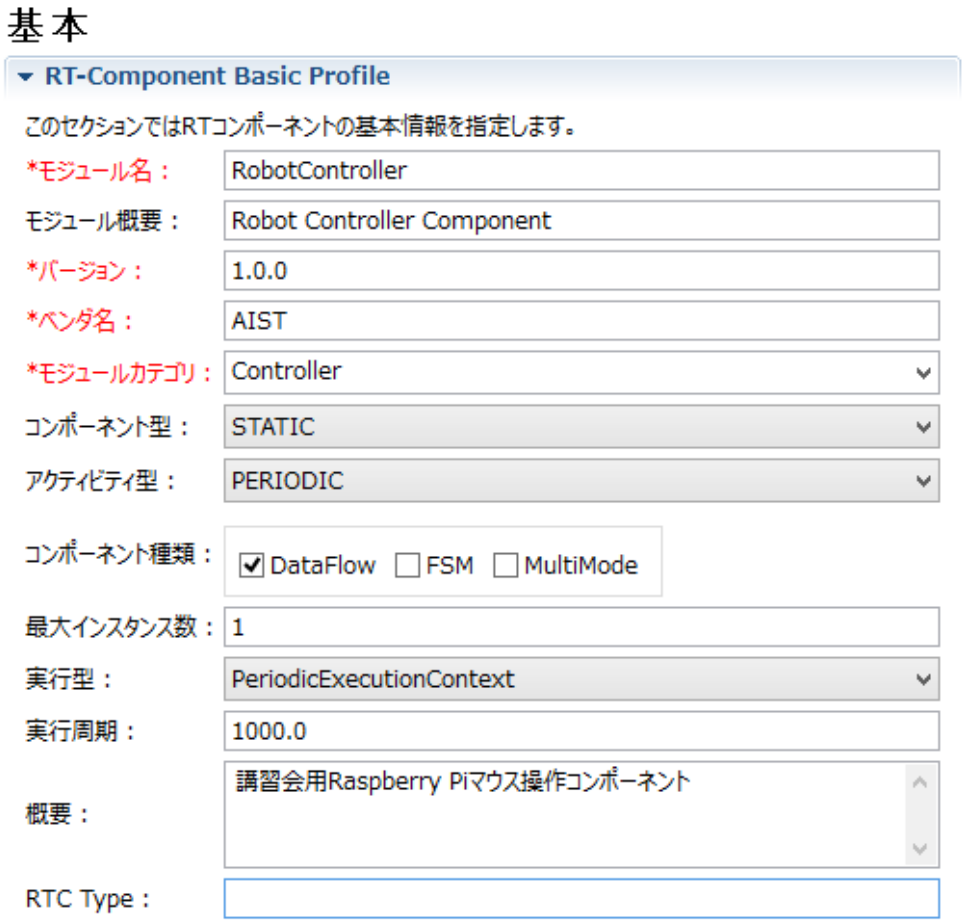
基本プロファイルの入力

- 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



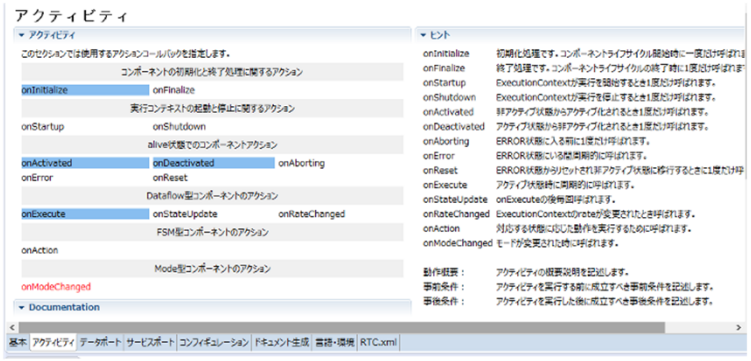
基本プロファイルの入力

- **モジュール名**
  - **RobotController**
- モジュール概要
  - 任意(Robot Controller Componen
- 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



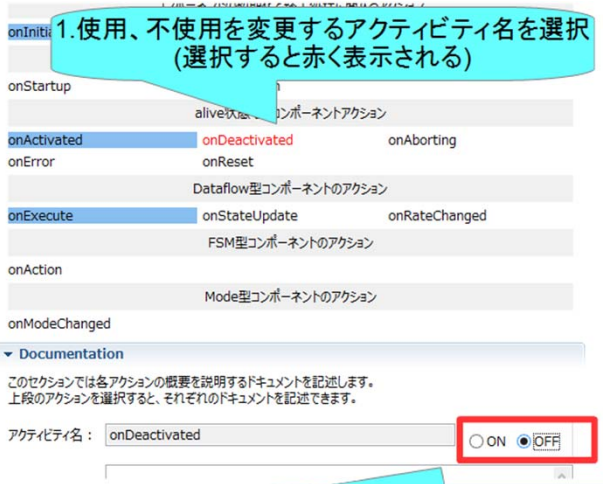
アクティビティの設定

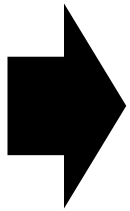
- 使用するアクティビティを設定する

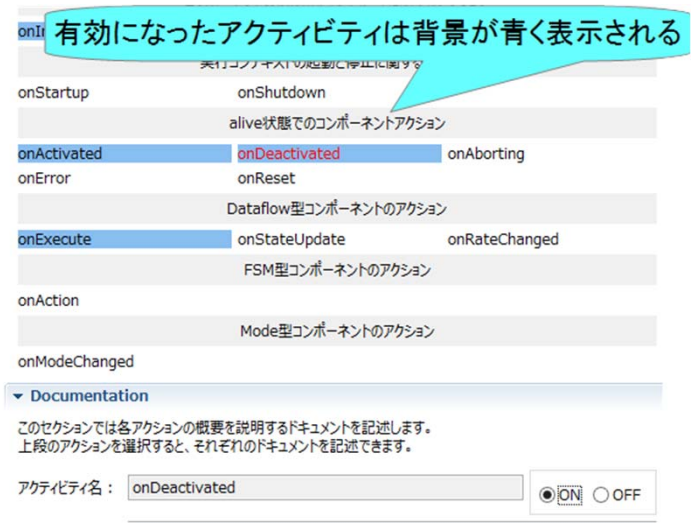


「アクティビティ」タブを選択

- 指定アクティビティを有効にする手順





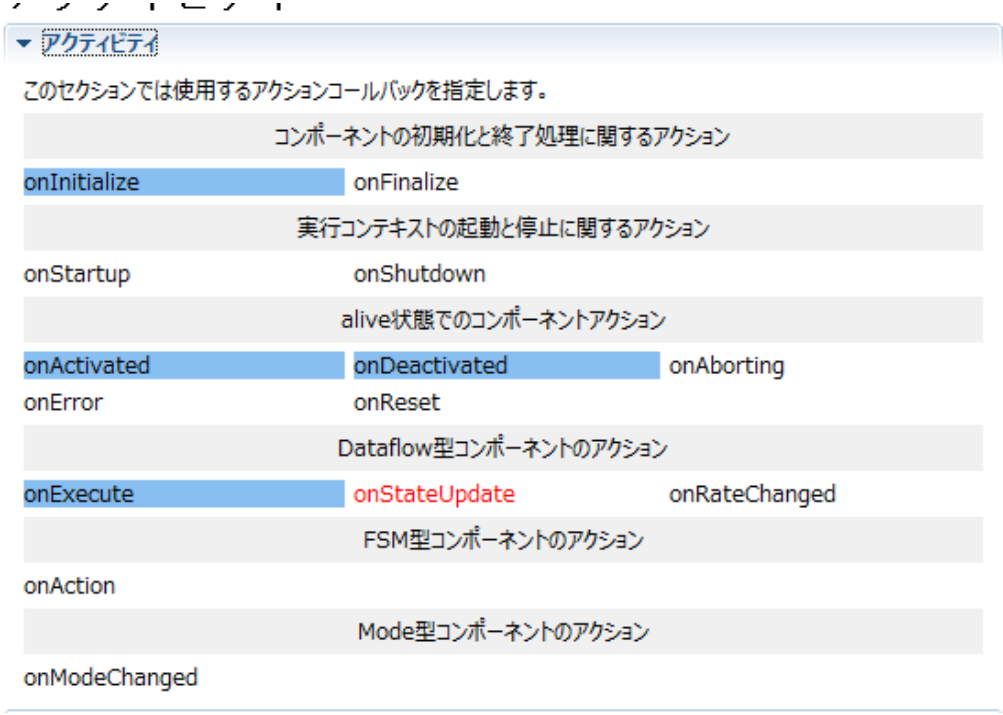


アクティビティの設定

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

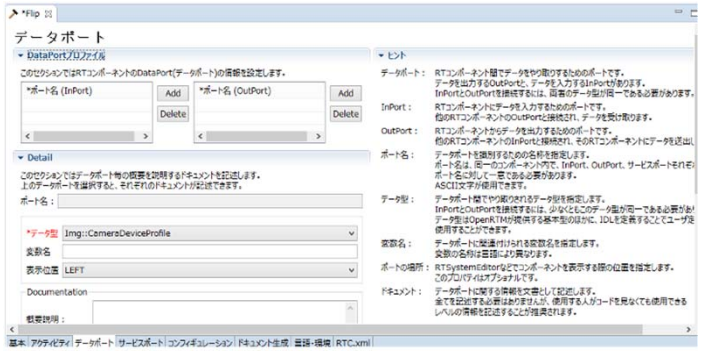
アクティビティの設定

- 以下のアクティビティを有効にする
  - onInitialize
  - **onActivated**
  - **onDeactivated**
  - **onExecute**
- Documentationは適当に書いておいてください
  - 空白でも大丈夫です

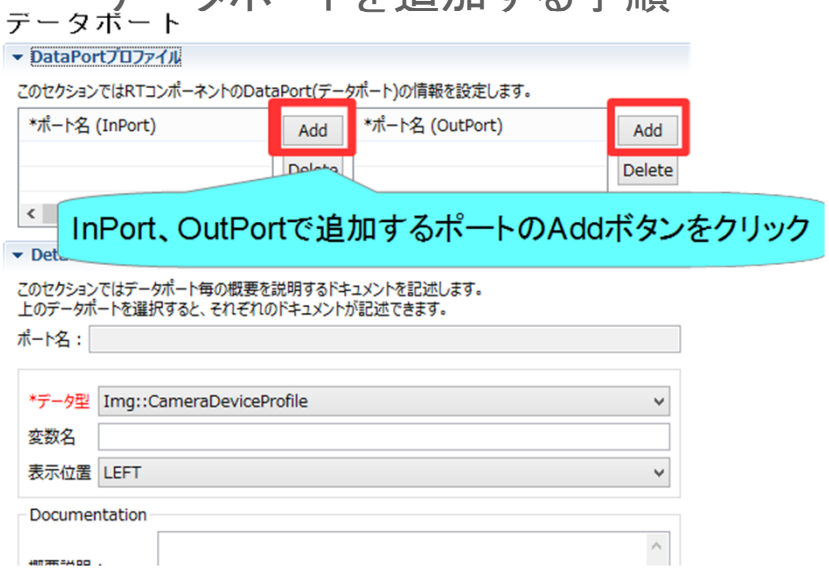


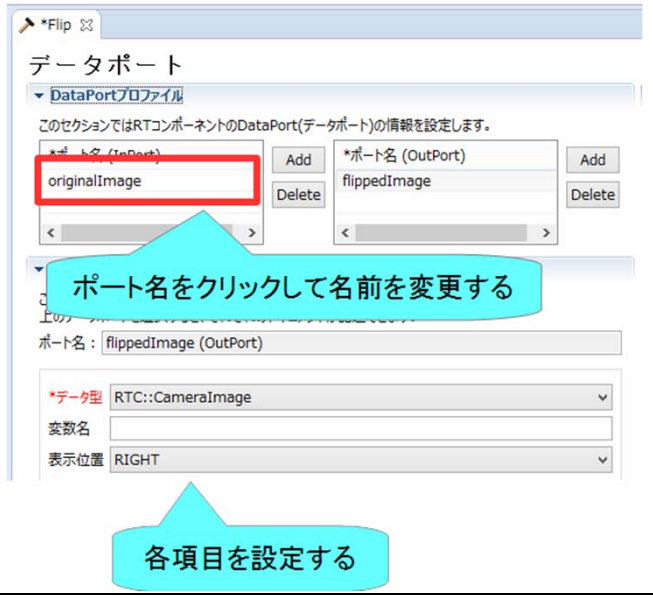
データポートの設定

- InPort、OutPortの追加、設定を行う



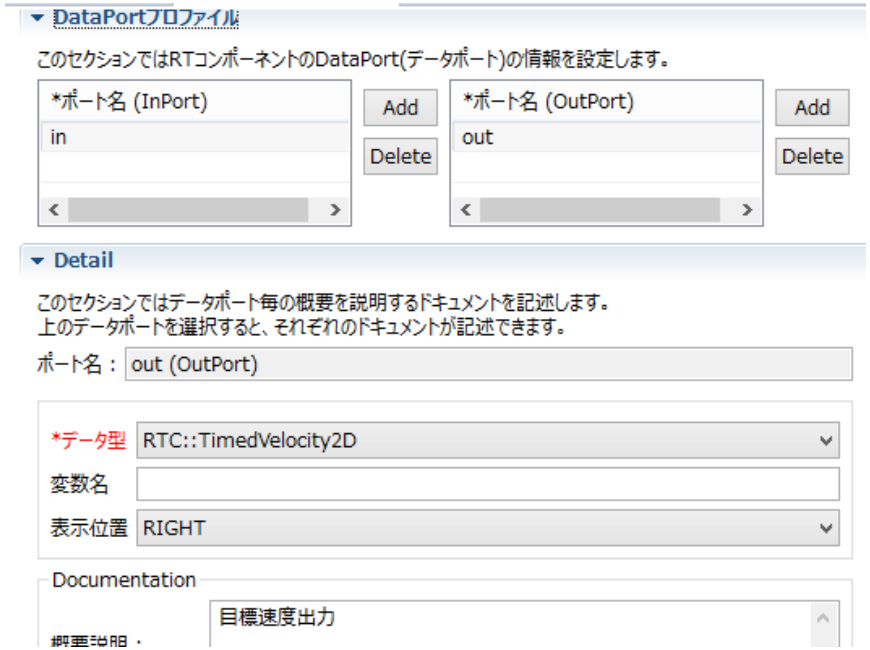
- データポートを追加する手順

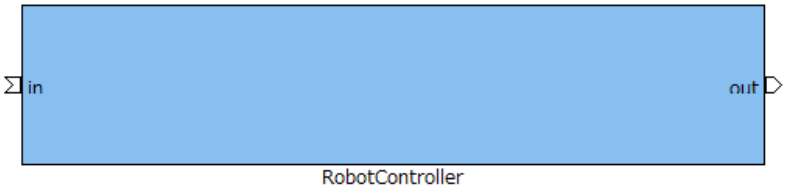




データポートの設定

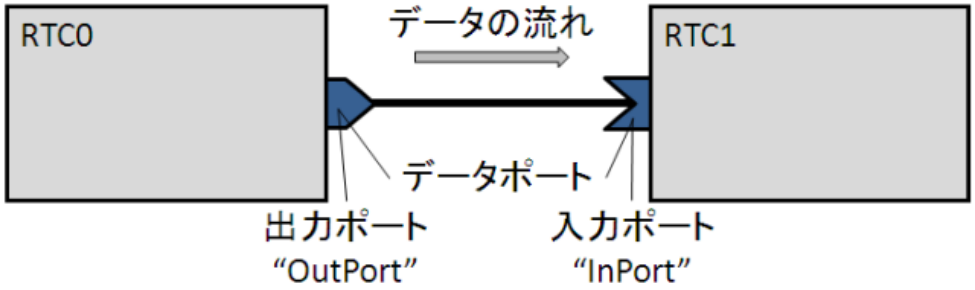
- 以下のOutPortを設定する
  - **out**
    - データ型：**RTC::TimedVelocity2D**
    - 他の項目は任意
- 以下のInPortを設定する
  - **in**
    - データ型：**RTC::TimedShortSeq**
    - 他の項目は任意



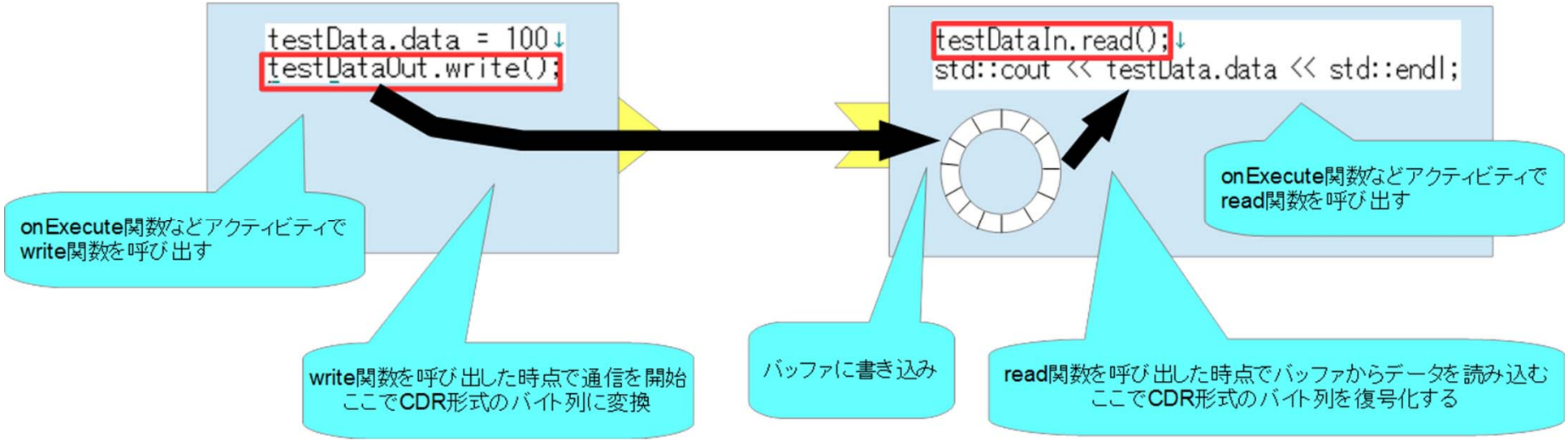


データポートについて

- 連続したデータを通信するためのポート



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

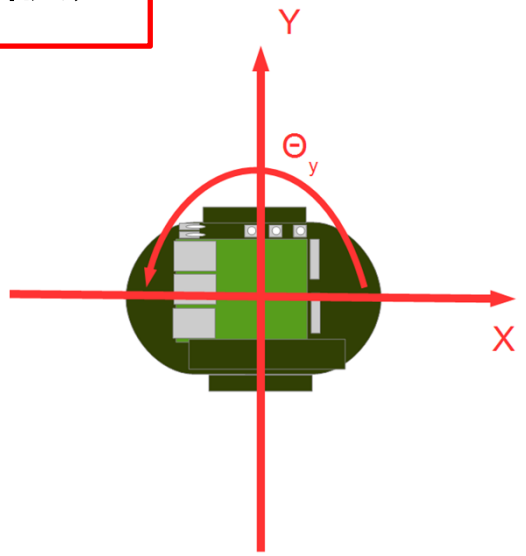


RTC::TimedVelocity2D型について

- 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

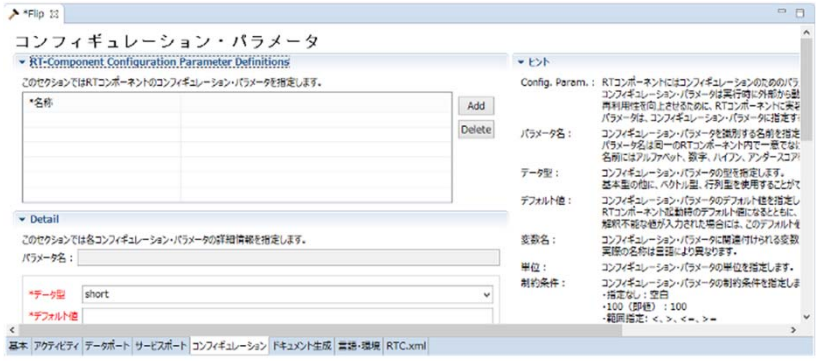
vxで直進速度、vaで回転速度を設定

```
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;
};
```



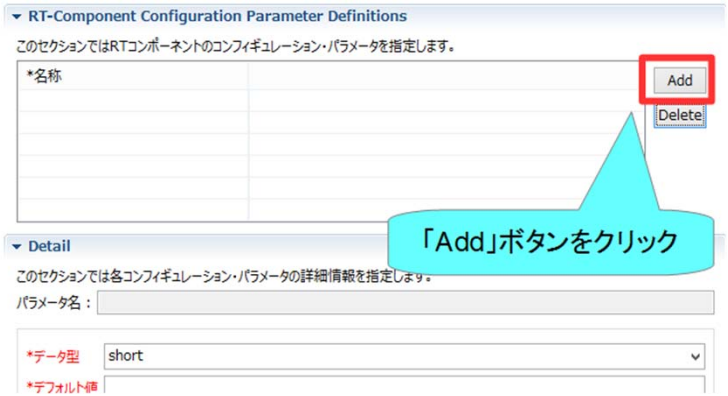
コンフィギュレーションの設定

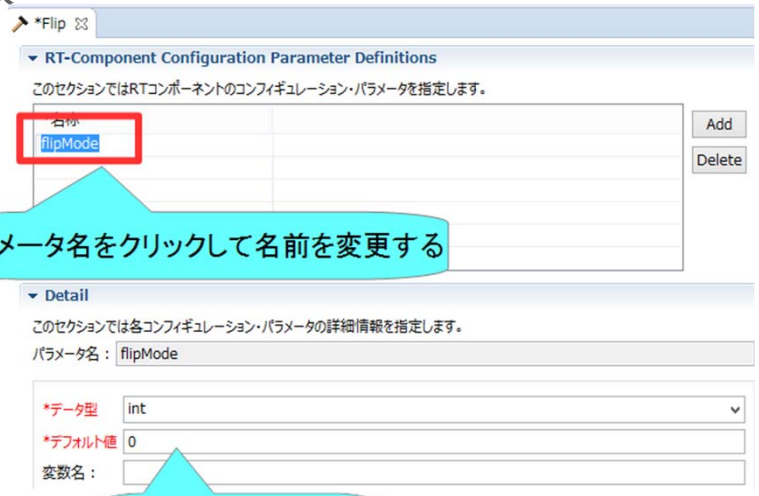
- コンフィギュレーションパラメータの追加、設定を行う



「コンフィギュレーション」タブを選択

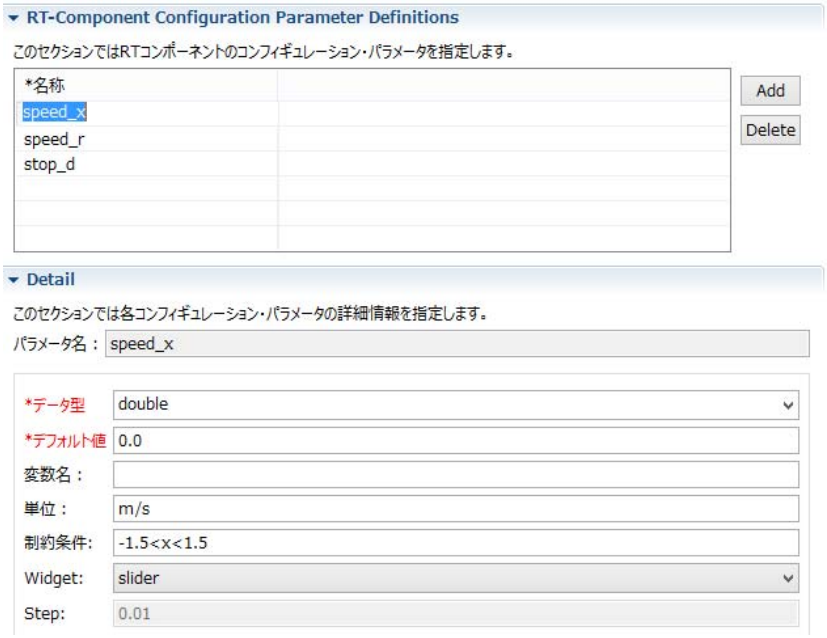
- コンフィギュレーションパラメータを追加する手順





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

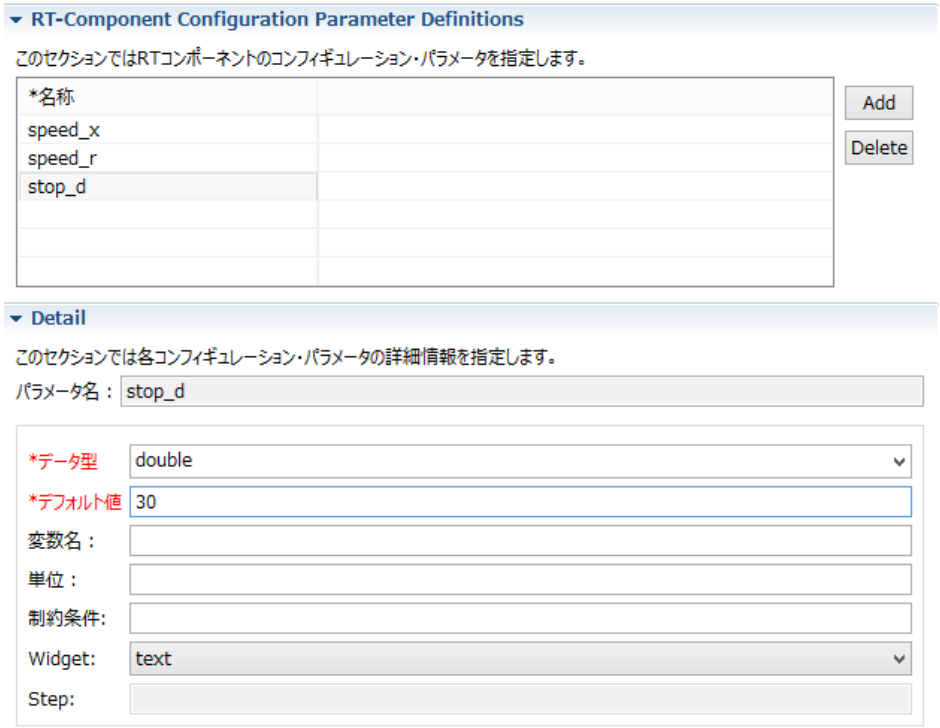
- 制約条件：(0,1,2,3)
- Widget：checkbox

- 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

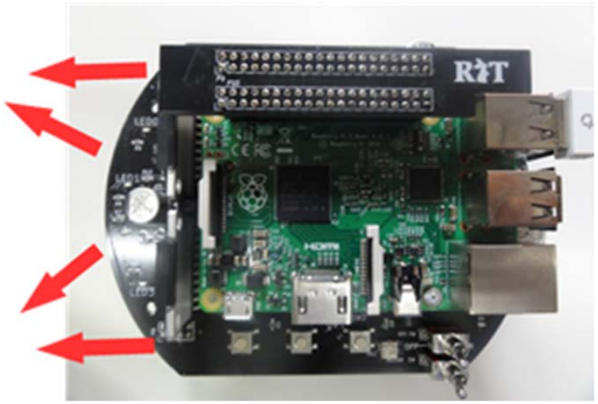
- 以下のコンフィギュレーション パラメータを追加
  - stop_d
    - データ型：int
    - デフォルト値：30
    - 他の項目は任意

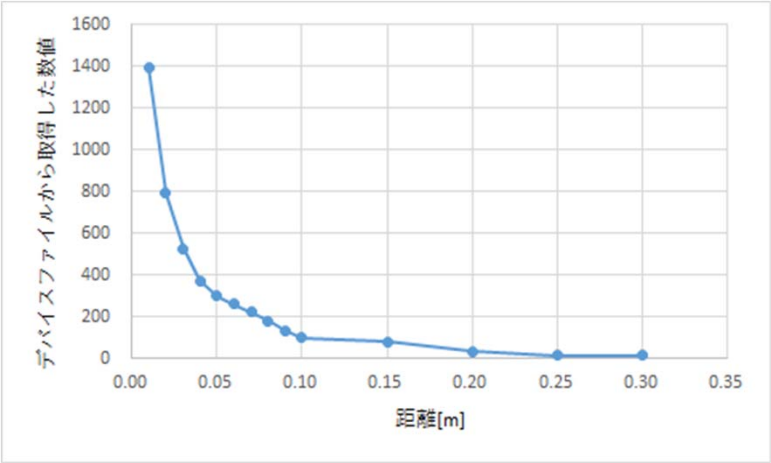
センサ値がこの値以上の場合に停止



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

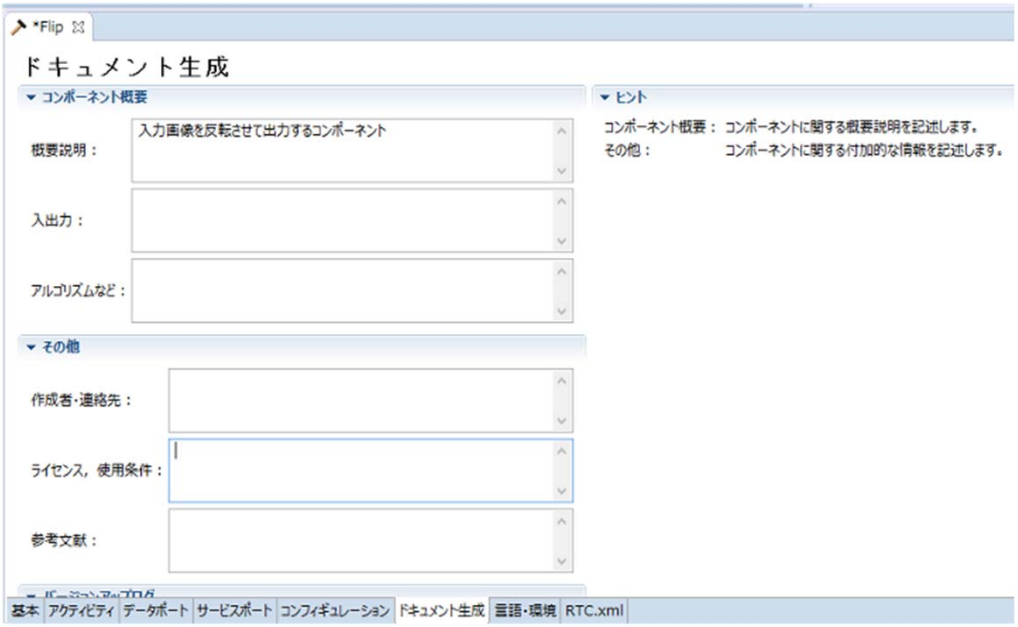




- シミュレータでもこのデータに近い値を計算して出力している

ドキュメントの設定

- 各種ドキュメント情報を設定

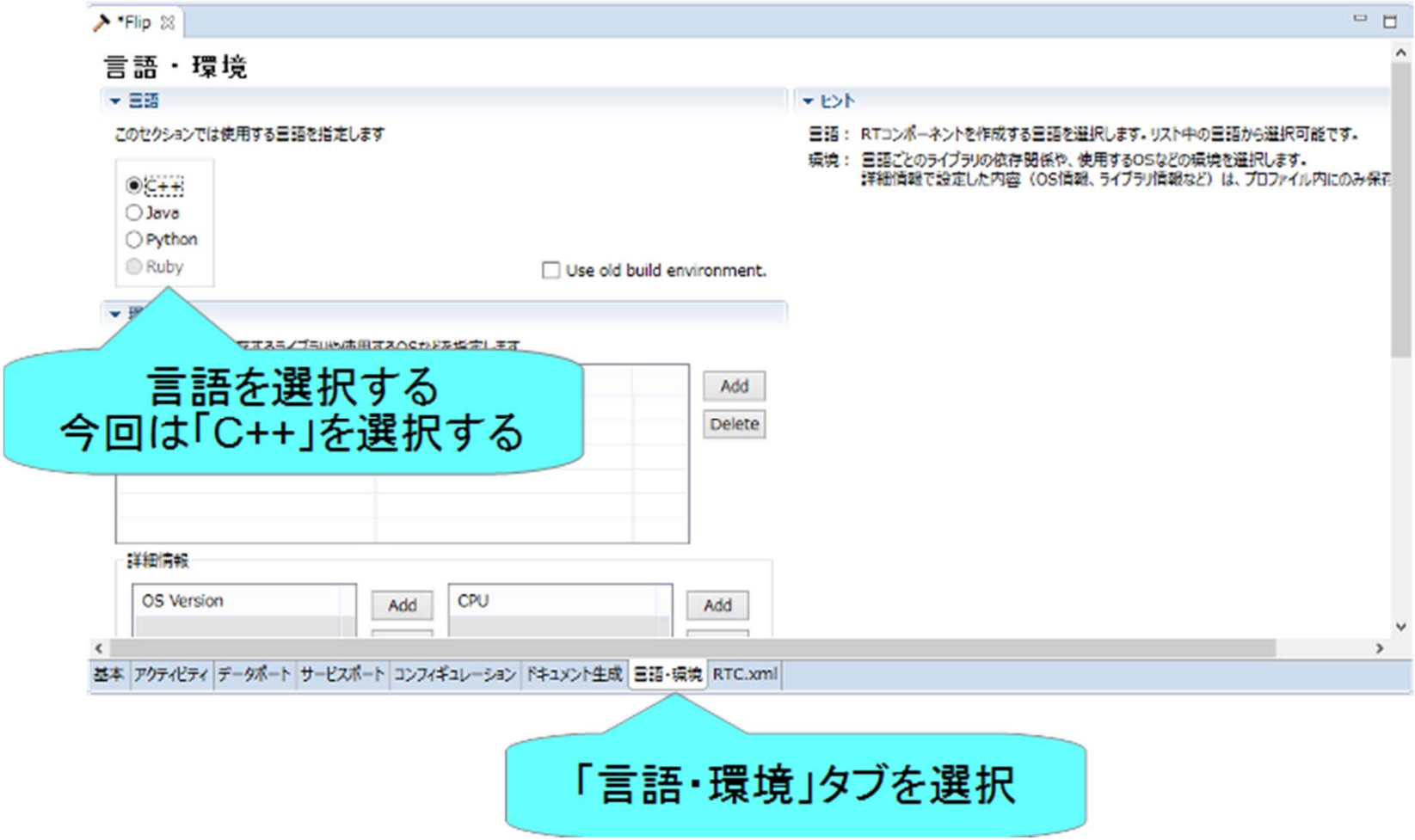


「ドキュメント生成」タブを選択

- 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

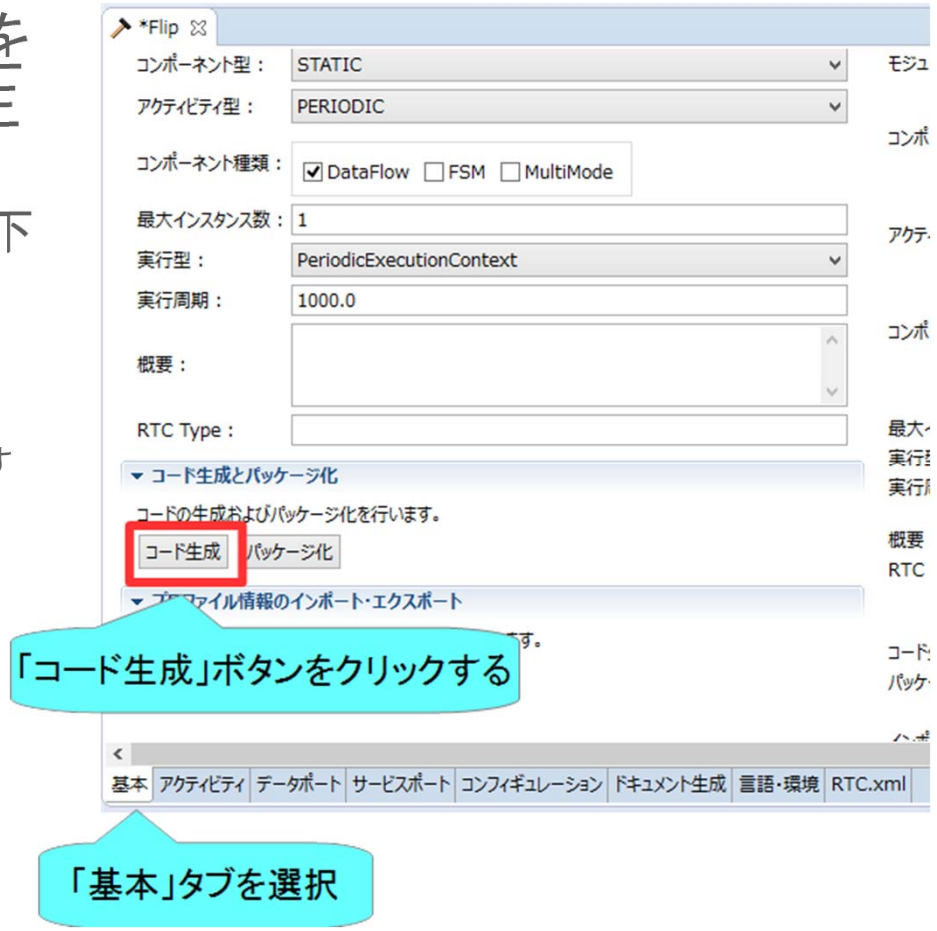
言語の設定

- 実装する言語，動作環境に関する情報を設定



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
  - ファイルが生成できているかを確認してください



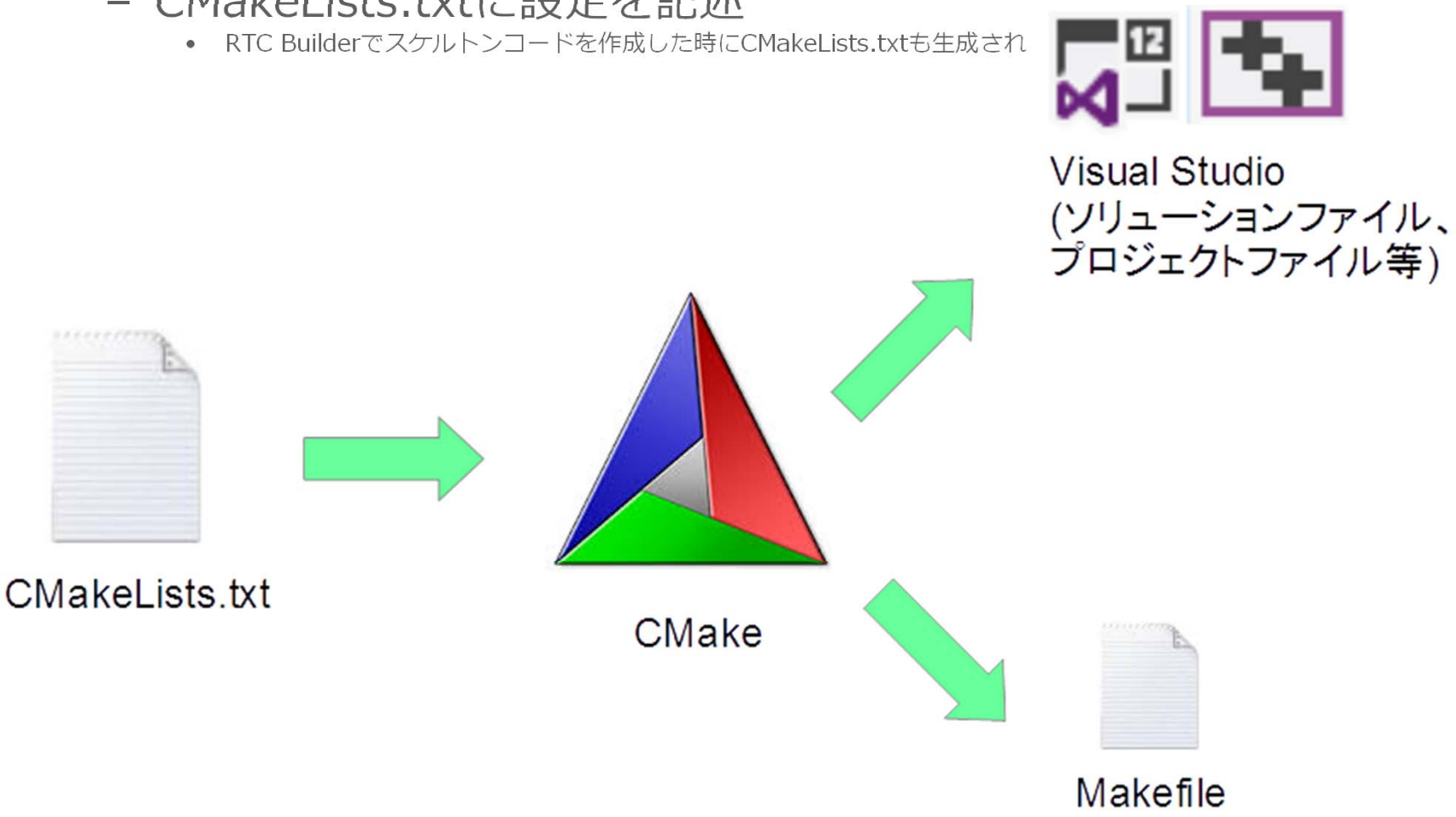
ソースコードの編集、RTCのビルド

手順

- ビルドに必要な各種ファイルを生成
  - 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
  - RobotController.hの編集
  - RobotController.cppの編集
- ビルド
  - Windows： Visual Studio
  - Ubuntu： Code::Blocks

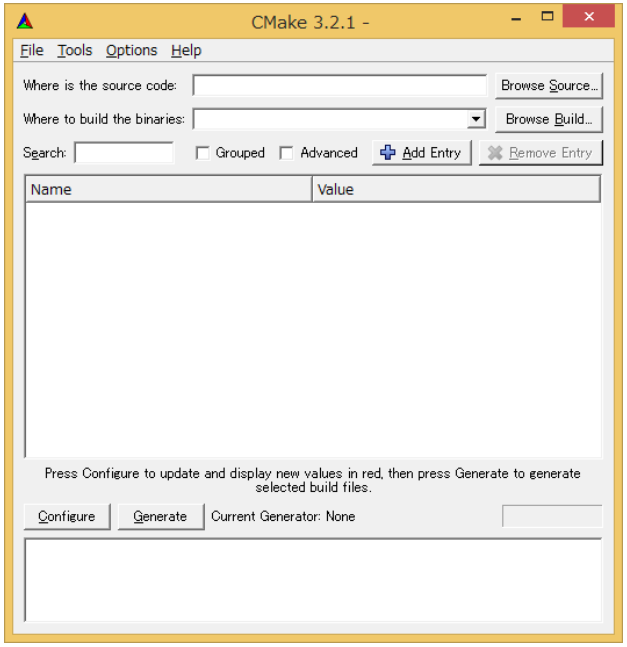
CMake

- 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



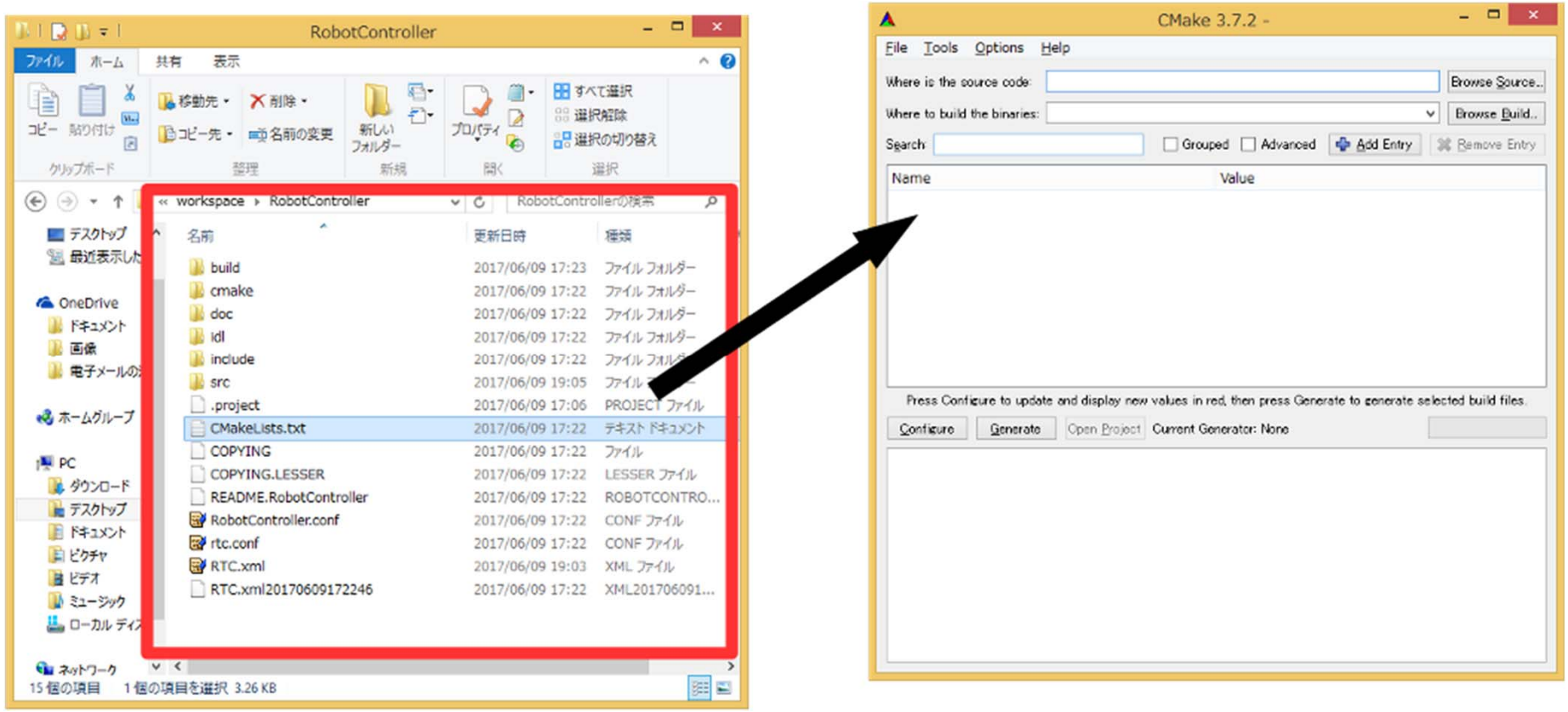
ビルドに必要なファイルの生成

- 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)」
  - Ubuntu
    - コマンドで「cmake-gui」を入力

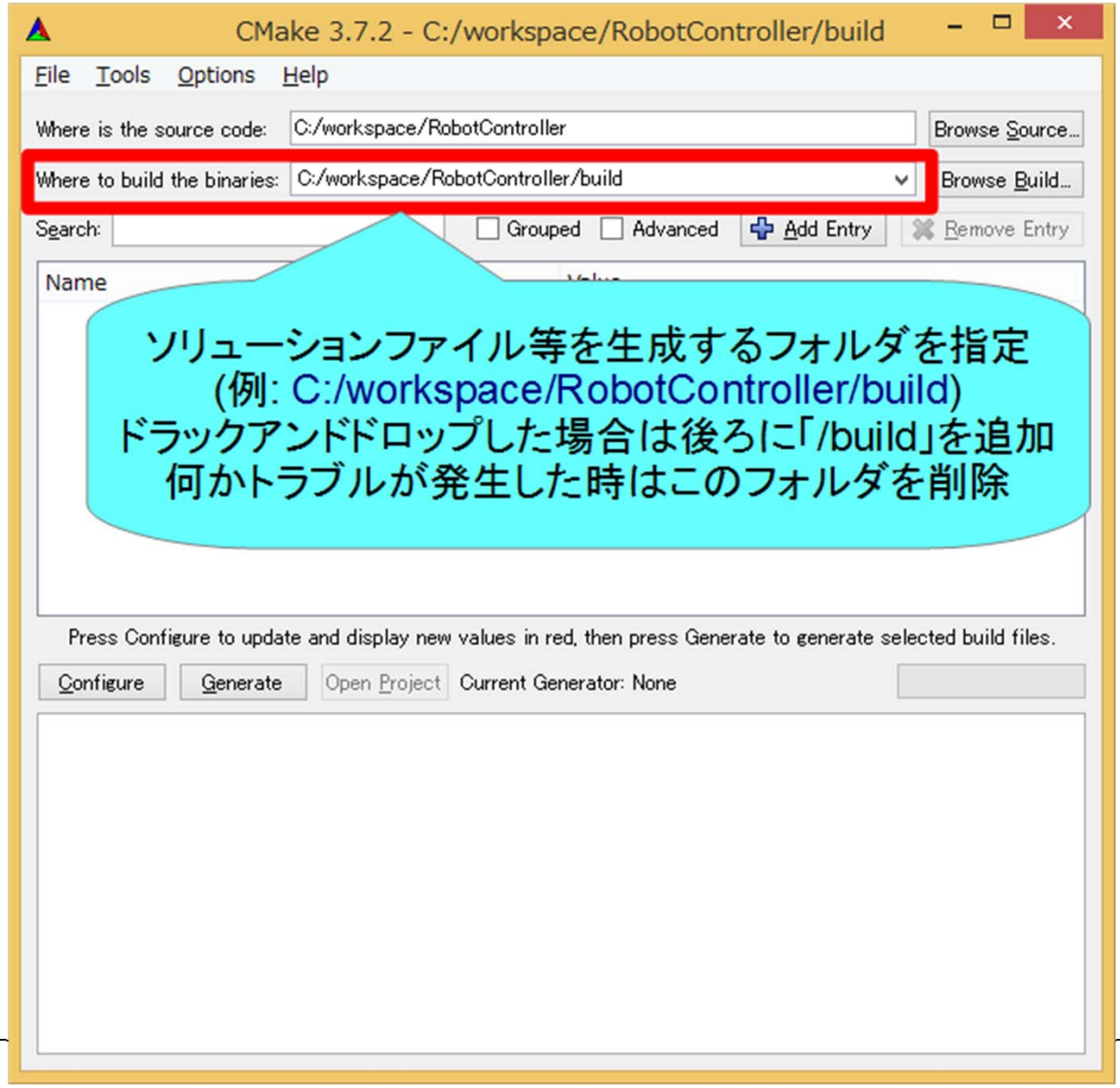


ビルドに必要なファイルの生成

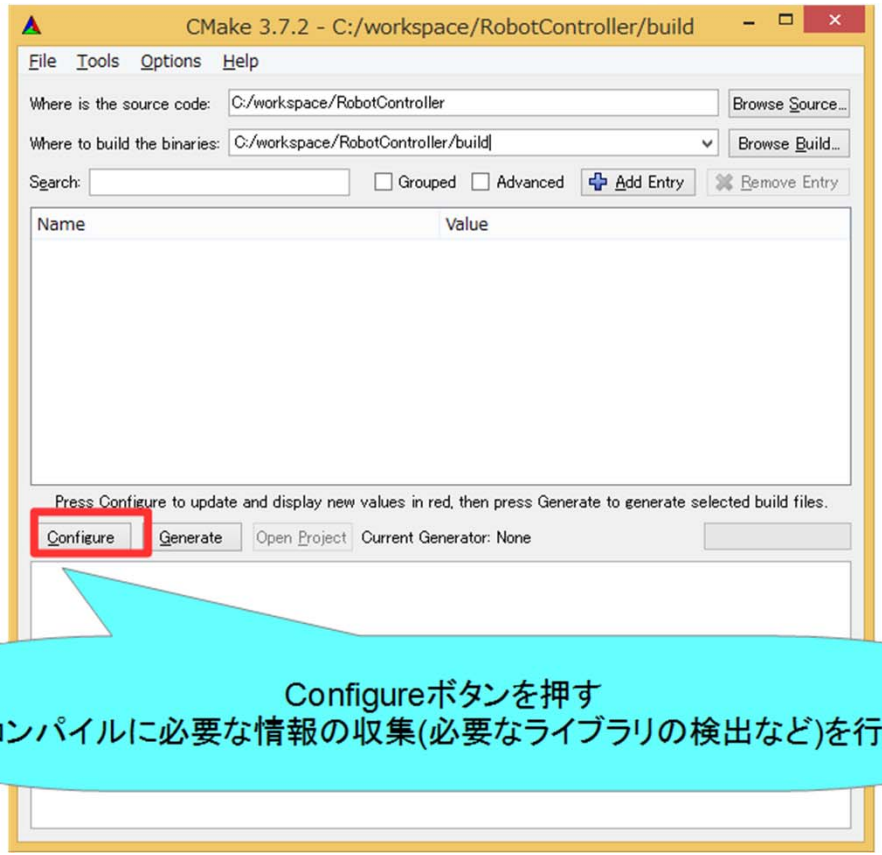
- **CMakeLists.txt**をcmake-guiにドラックアンドドロップ
  - CMakeLists.txtはRTC Builderで生成したプロジェクトのフォルダ
    (例： C:¥workspace¥RobotController)



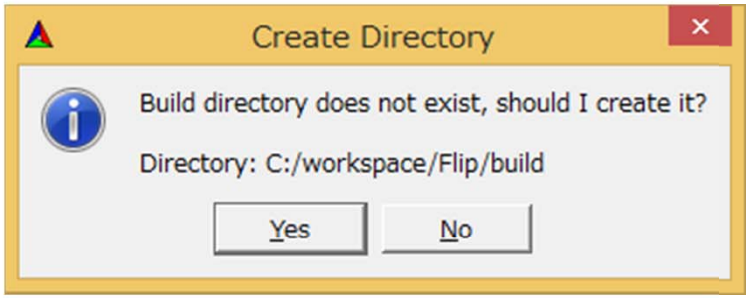
ビルドに必要なファイルの生成



ビルドに必要なファイルの生成



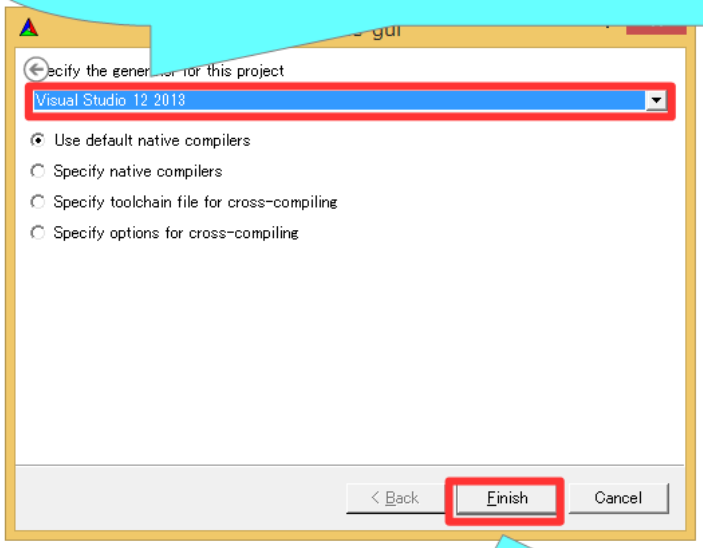
Configureボタンを押す
コンパイルに必要な情報の収集(必要なライブラリの検出など)を行う



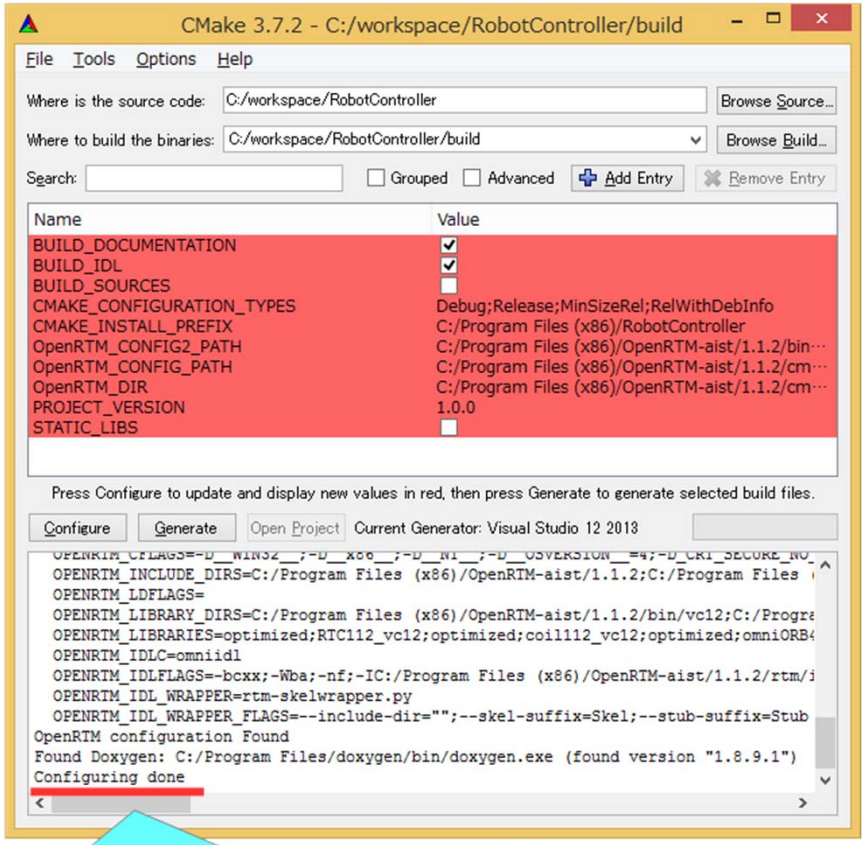
buildフォルダが存在しない場合は
作成するかどうかきかれるため「Yes」を選択

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

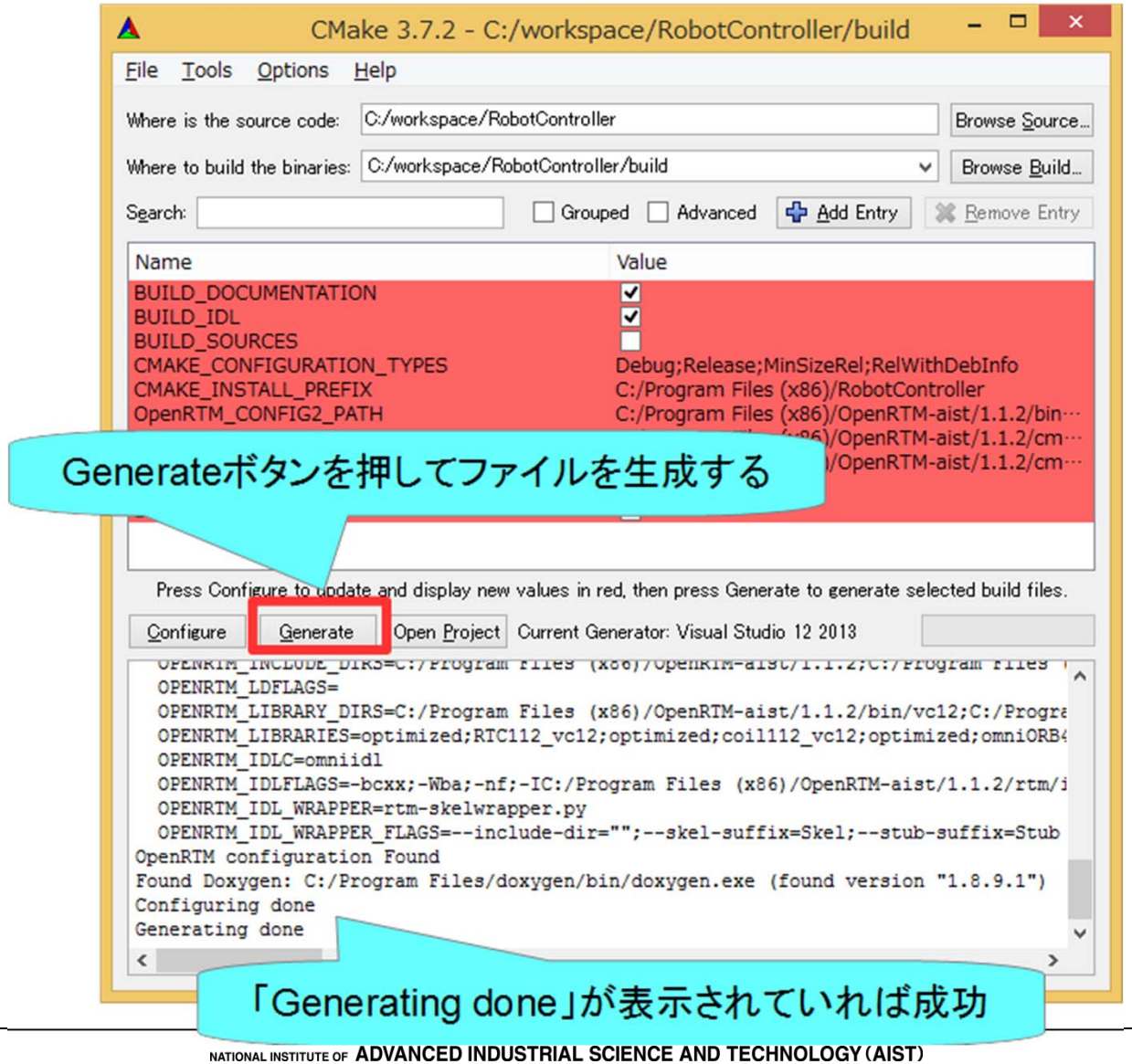


設定後、Finishボタンを押す



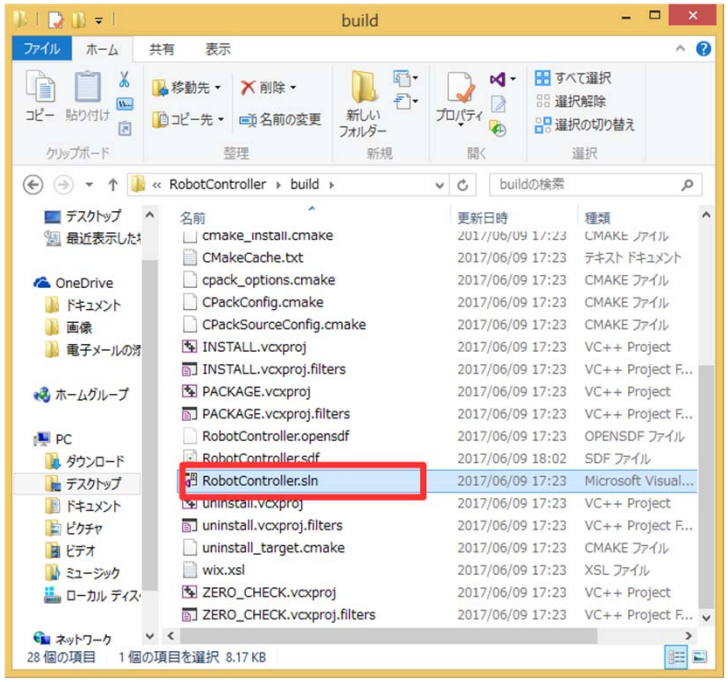
「Configure done」が表示されていれば成功

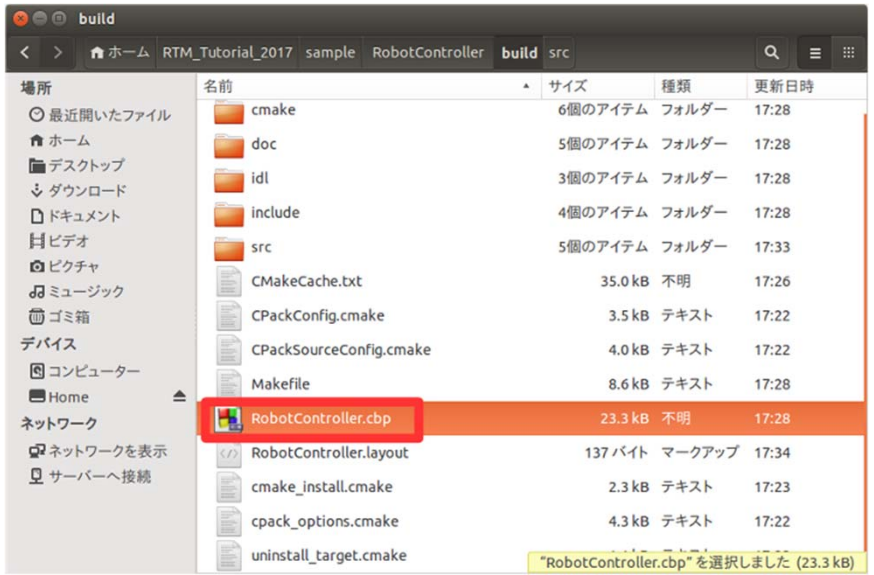
ビルドに必要なファイルの生成



ソースコードの編集

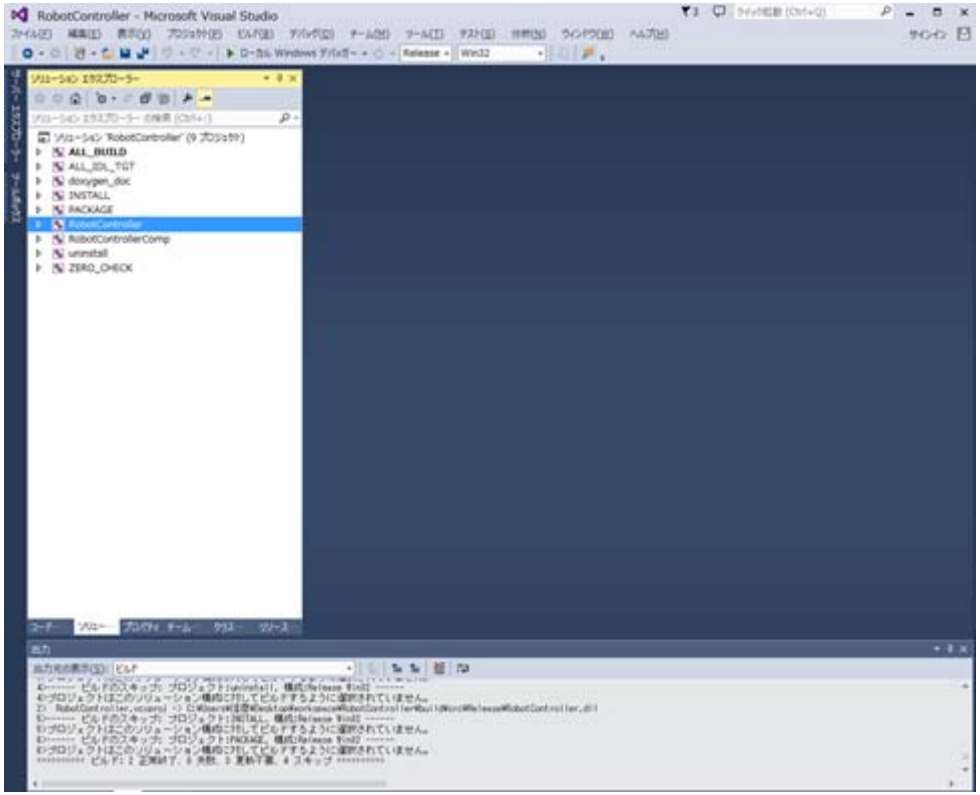
- Windows
  - buildフォルダの「RobotController.sln」をダブルクリックして開く
- Ubuntu
  - buildフォルダの「RobotController.cbp」をダブルクリックして開く

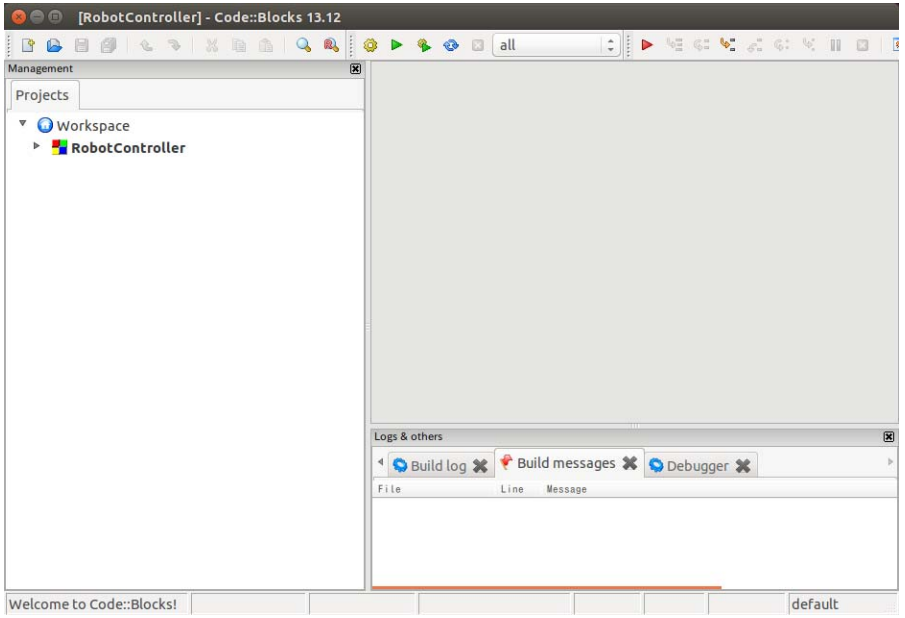




ソースコードの編集

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

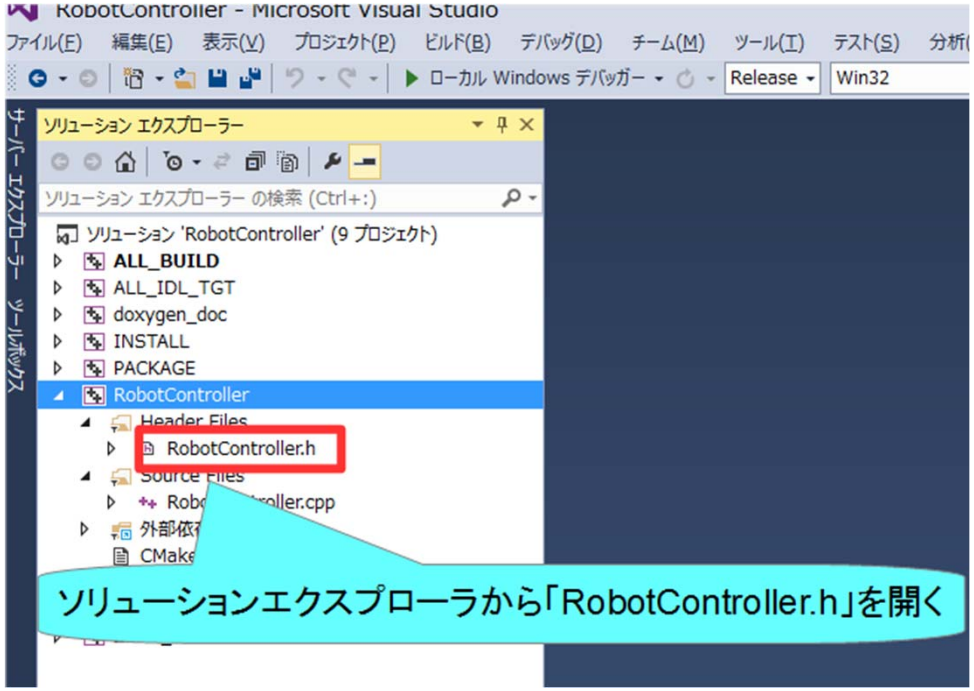




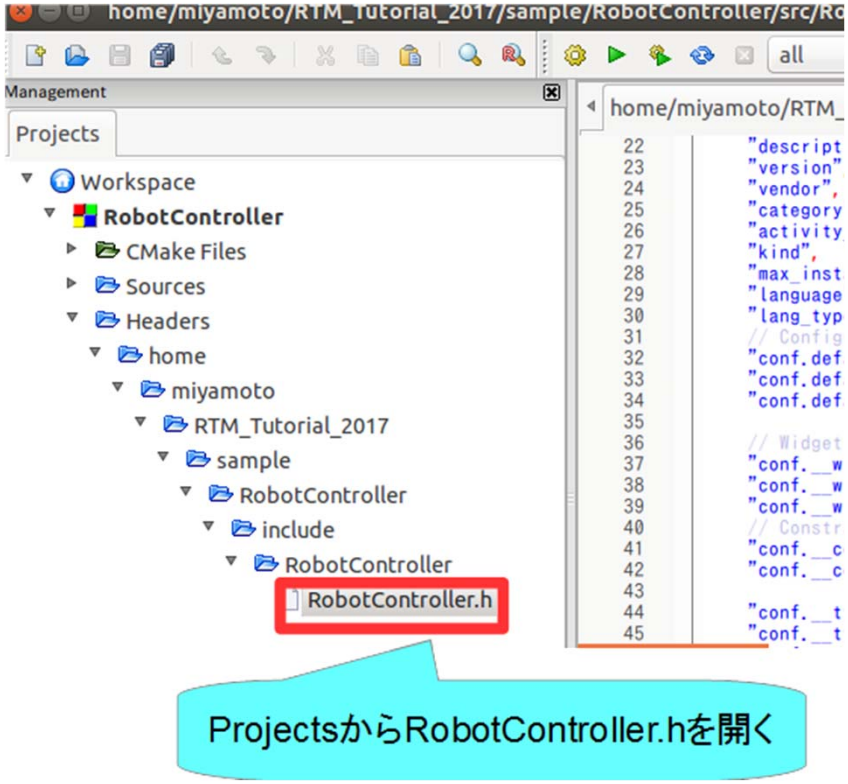
ソースコードの編集

- RobotController.hの編集

Visual Studio



Code::Blocks



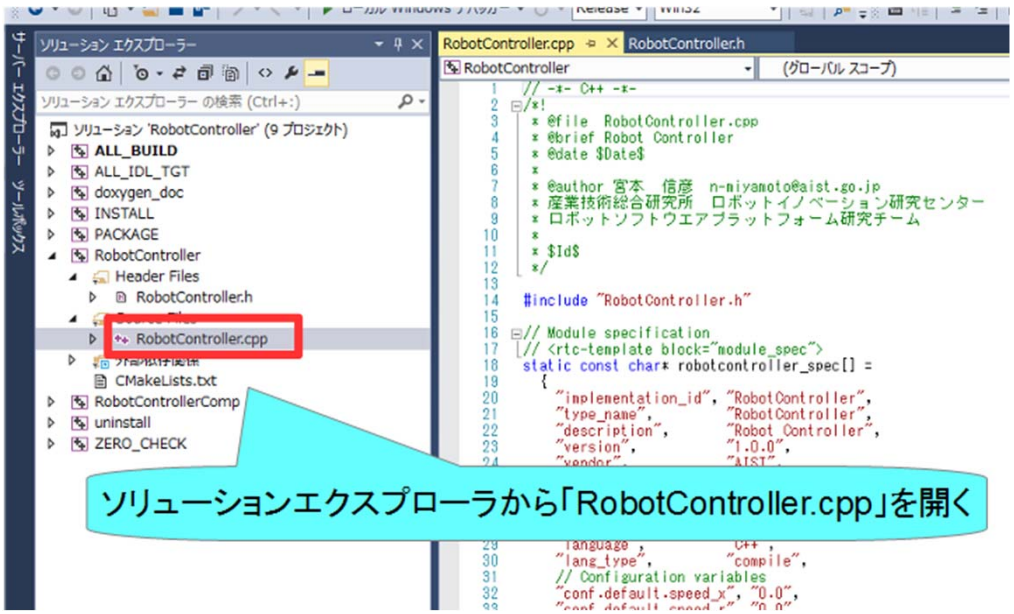
ソースコードの編集

- RobotController.hの編集

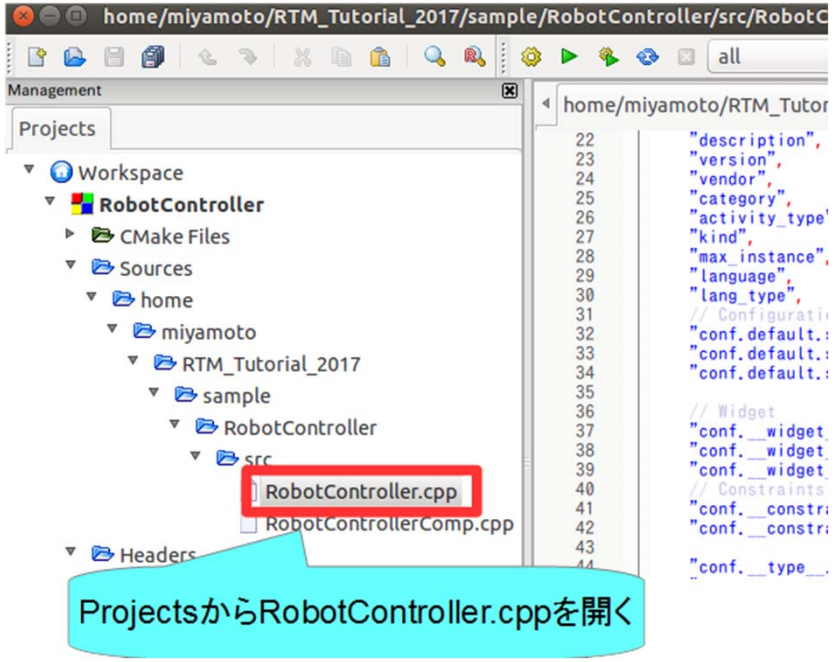
ソースコードの編集

- RobotController.cppの編集
  - 詳細はUSBメモリ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();
```

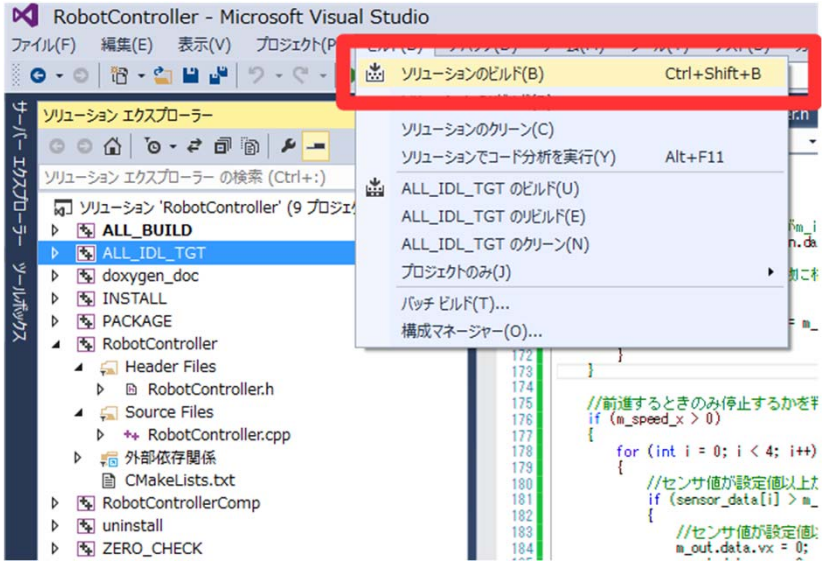
変数m_outにデータを格納する
TimedVelocity2D型のため、vxに直進速度、
vaに回転速度を格納する。

write関数でデータの書き込み

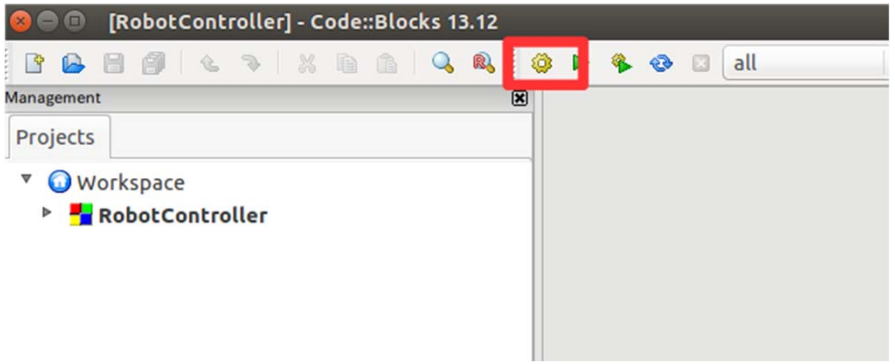
補足:コンフィギュレーションパラメータは変更すると
対応する変数(m_speed_x、m_speed_r、m_stop_d)
に値が格納される

ソースコードのコンパイル

Visual Studio



Code::Blocks

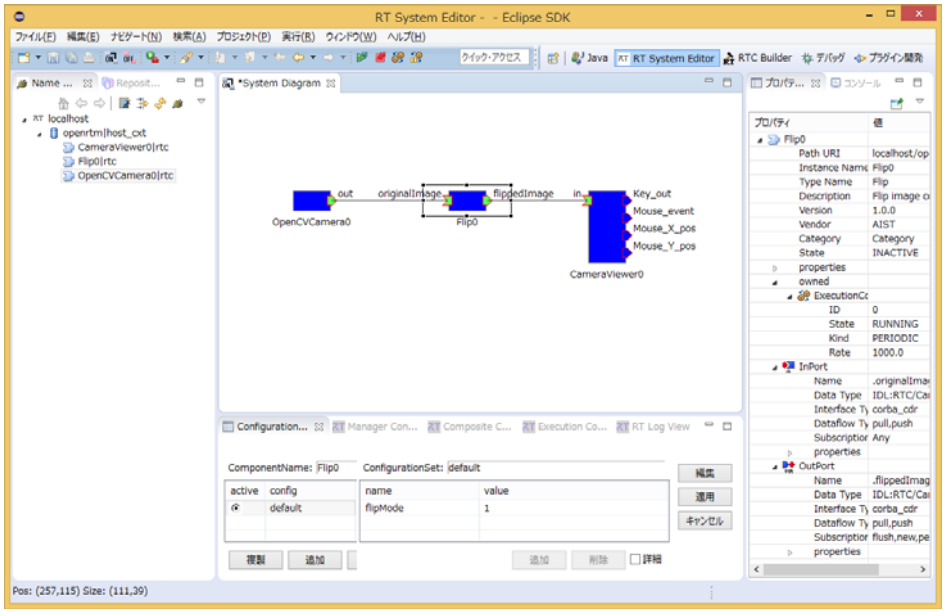


- 成功した場合、実行ファイルが生成される
  - Windows
    - **build¥src**フォルダの**Release**(もしくは**Debug**)フォルダ内にRobotControllerComp.exeが生成される
  - Ubuntu
    - **build/src**フォルダにRobotControllerCompが生成される

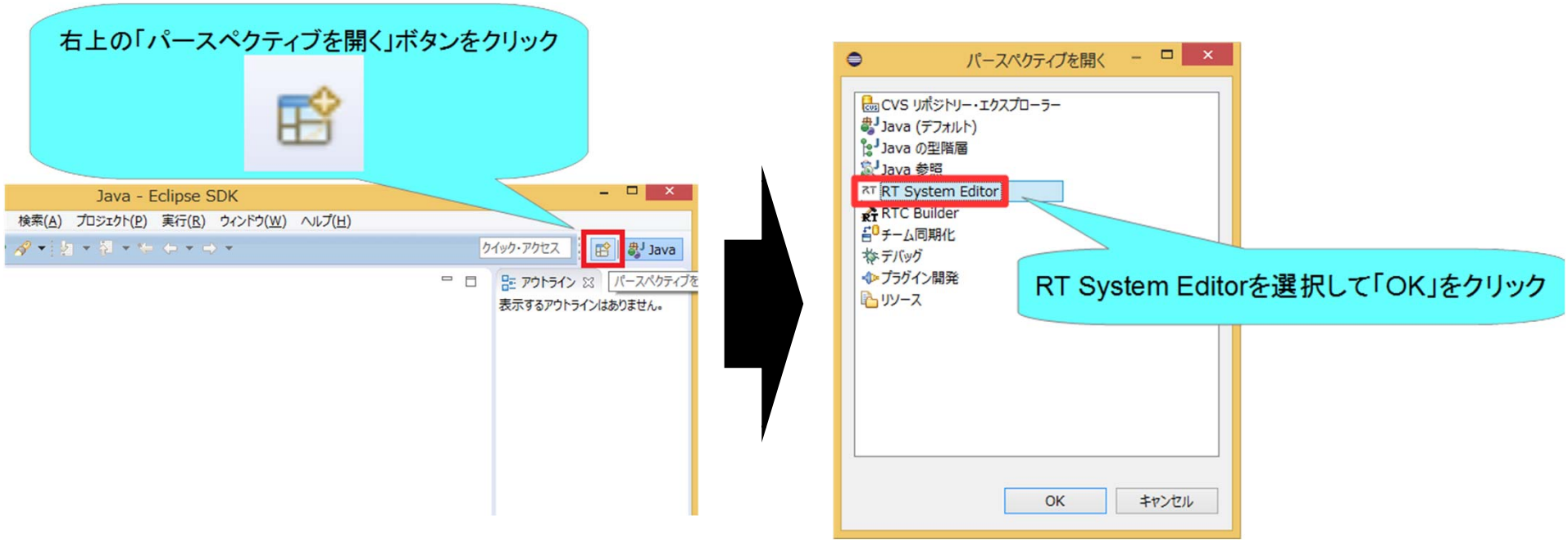
システム構築支援ツール
RTSystemEditorについて

RTSystemEditor

- 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



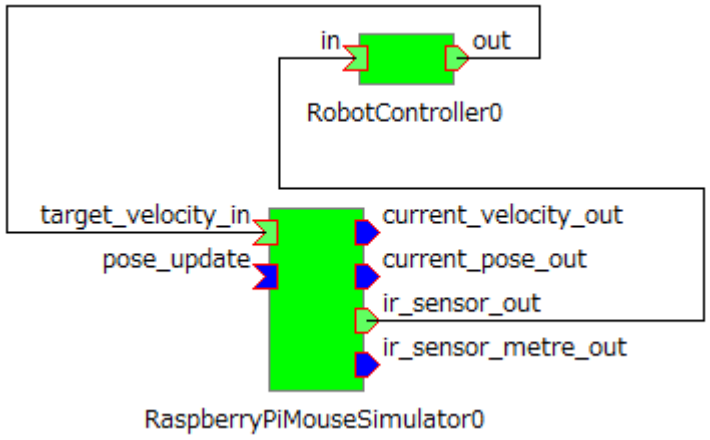
RT System Editorの起動



RT System Editorの画面構成

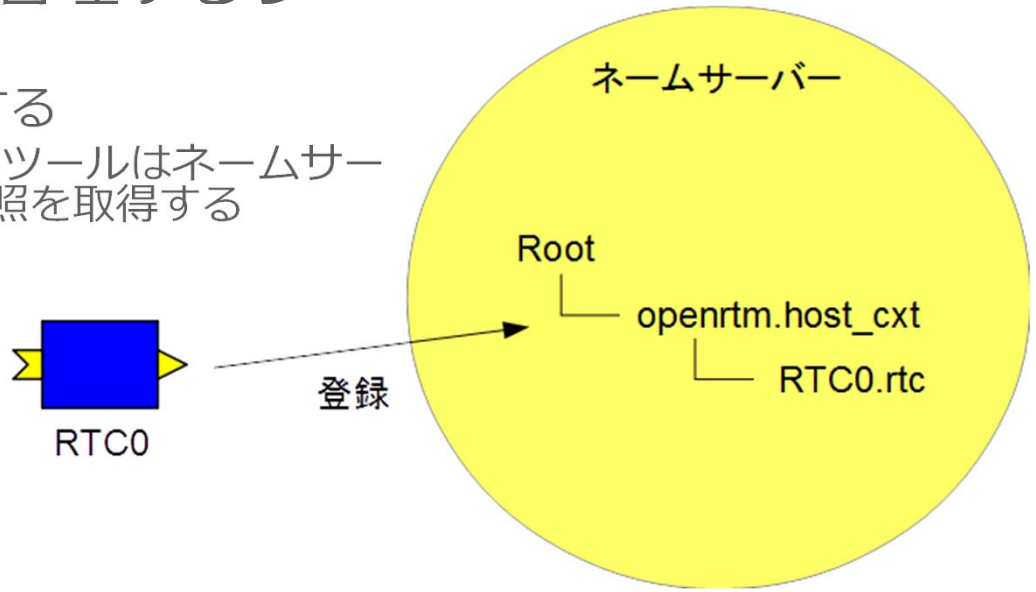
RobotControllerコンポーネントの動作確認

- 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
  - RasPiMouseSimulatorコンポーネントを起動する
    - Windows
      - 配布USBメモリのEXEフォルダ内
        「 RaspberryPiMouseSimulatorComp.exe 」をダブルクリック
    - Ubuntu
      - 配布USBメモリ内のスクリプトでインストール
        - » $ sh install_raspimouse_simulator_offline.sh
      - RasPiMouseSimulatorRTCに移動して以下のコマンドを実行
        - » $ build/src/RaspberryPiMouseSimulatorComp
  - RobotControllerコンポーネント起動
  - RasPiMouseSimulator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

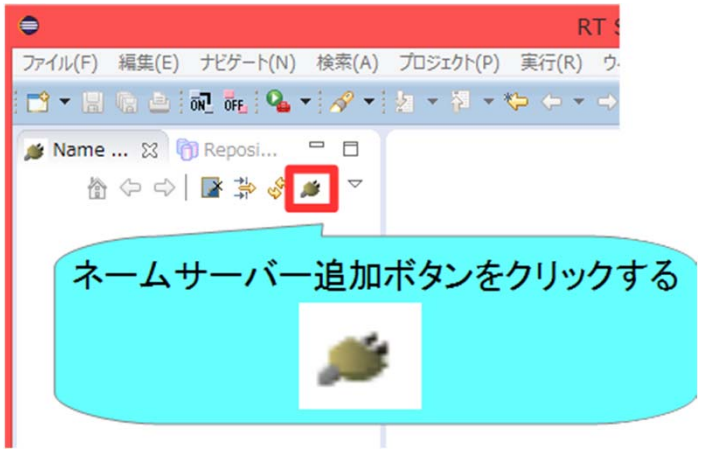


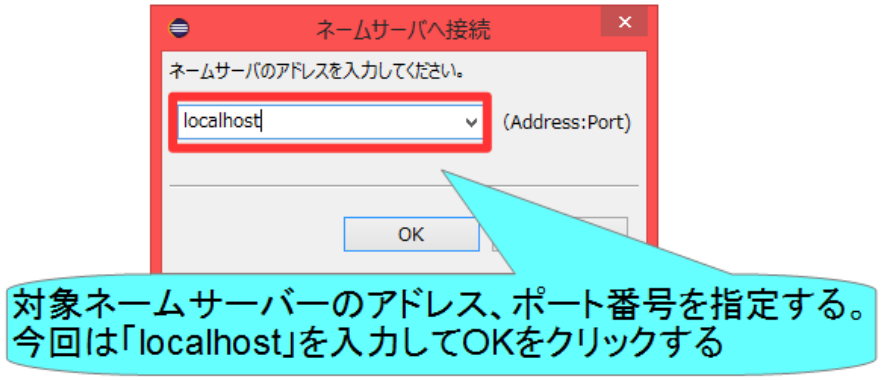
- 起動する手順
  - Windows 7
    - 「スタート」→「すべてのプログラム」→「OpenRTM-aist 1.1.2」→「Tools」→「Start Naming Service」
  - Windows 8.1
    - 「スタート」→「アプリビュー(右下矢印)」→「OpenRTM-aist 1.1.2」→「Start Naming Service」
  - Ubuntu
    - $ rtm-naming

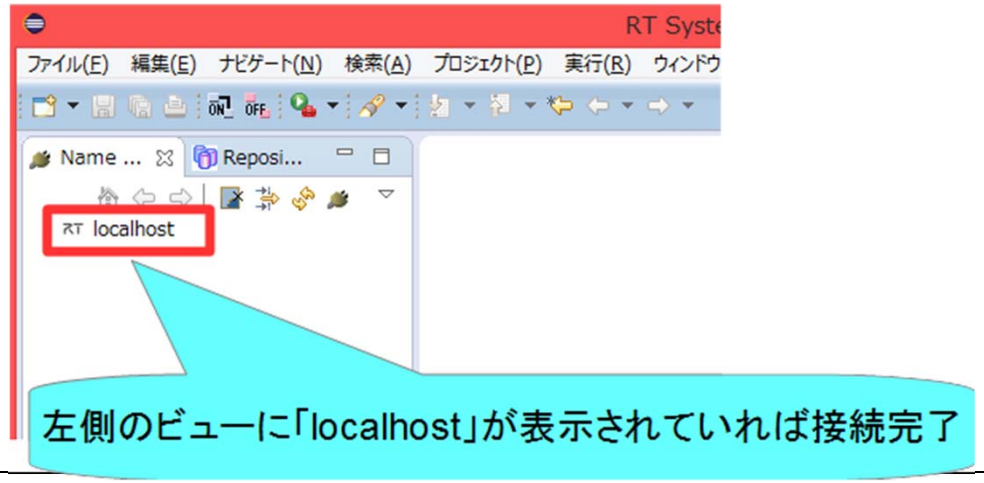
ネームサーバーの起動

- Windows 8.1

ネームサーバーへ接続


ファイル(F) 編集(E) ナビゲート(N) 検索(A) プロジェクト(P) 実行(R)
Name ...
Reposi...
ネームサーバー追加ボタンをクリックする

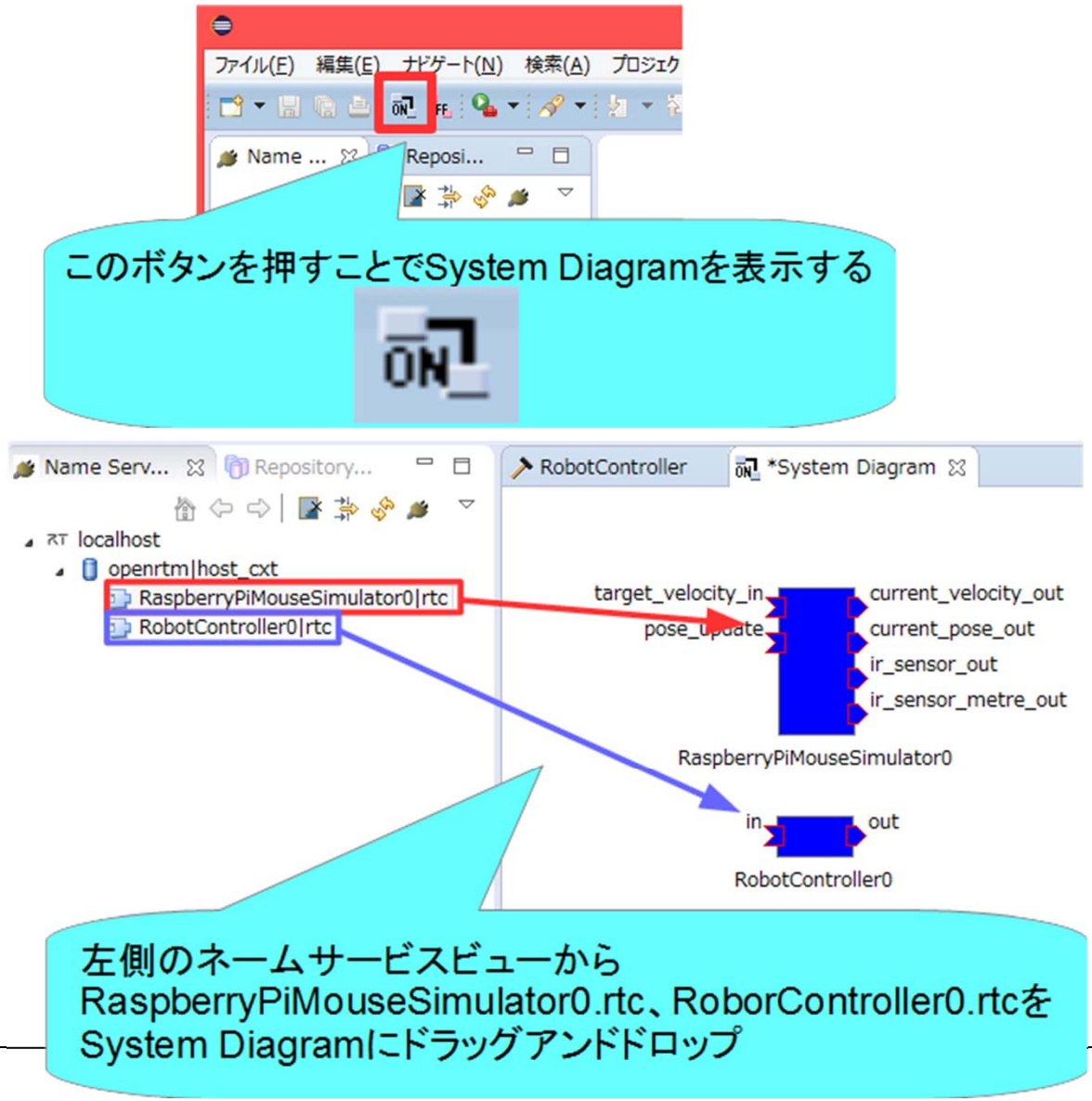

ネームサーバへ接続
ネームサーバのアドレスを入力してください。
localhost
(Address:Port)
OK
対象ネームサーバーのアドレス、ポート番号を指定する。
今回は「localhost」を入力してOKをクリックする


RT Syste
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
Name ...
Reposi...
localhost
左側のビューに「localhost」が表示されていれば接続完了

RobotControllerコンポーネントの動作確認

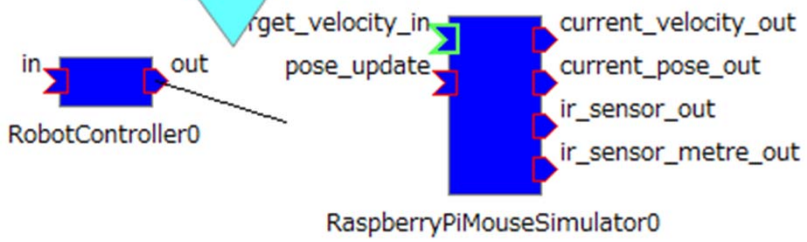
- シミュレータコンポーネントと接続してシミュレータ上のロボットを操作するRTシステムを作成する
  - ネームサーバーを起動する
  - RasPiMouseSimulatorコンポーネントを起動する
    - Windows
      - 配布USBメモリのEXEフォルダ内「 RaspberryPiMouseSimulatorComp.exe 」をダブルクリック
    - Ubuntu
      - 配布USBメモリ内のスクリプトでインストール
        - » $sudo sh install_raspimouse_simulator.sh
      - RasPiMouseSimulatorRTCに移動して以下のコマンドを実行
        - » build/src/RaspberryPiMouseSimulatorComp
  - RobotController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RobotControllerComp.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RobotControllerComp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

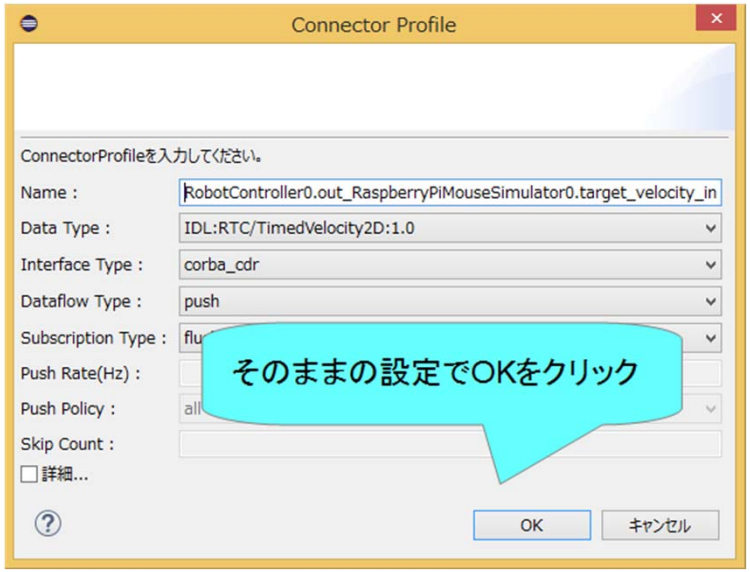
データポートの接続

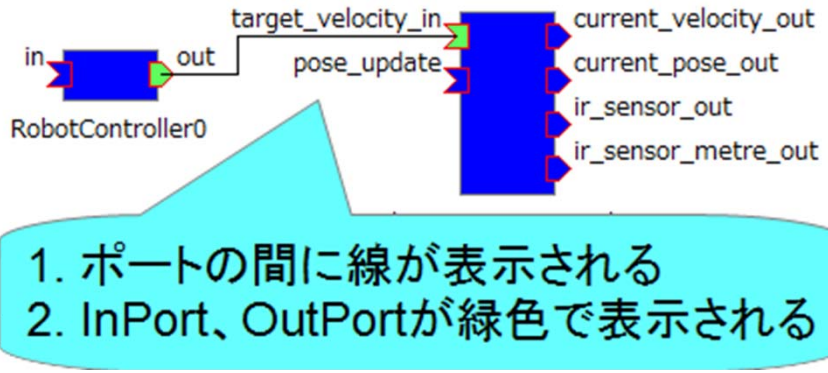

ファイル(F) 編集(E) ナビゲート(N) 検索(A) プロジェク
Name ...
Reposi...
このボタンを押すことでSystem Diagramを表示する
ON
Name Serv...
Repository...
RobotController
*System Diagram
localhost
openrtm|host_cxt
RaspberryPiMouseSimulator0|rtc
RobotController0|rtc
target_velocity_in
pose_update
current_velocity_out
current_pose_out
ir_sensor_out
ir_sensor_metre_out
RaspberryPiMouseSimulator0
in
out
RobotController0
左側のネームサービスビューから
RaspberryPiMouseSimulator0.rtc、RoborController0.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

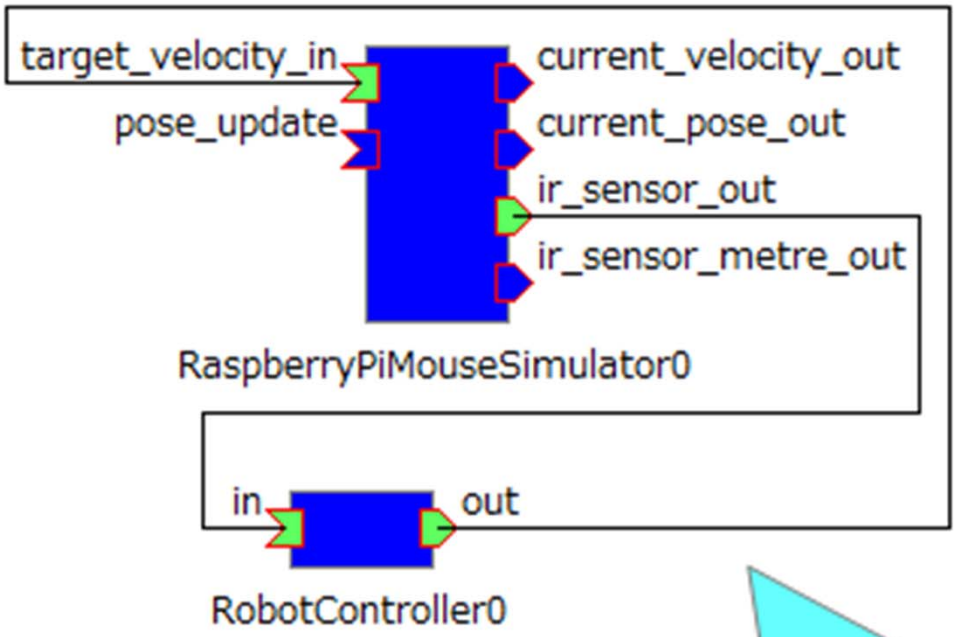
RobotController0の「out」を選択して、
RaspberryPiMouseSimulator0の
「target_velocity_in」にドラッグアンドドロップ





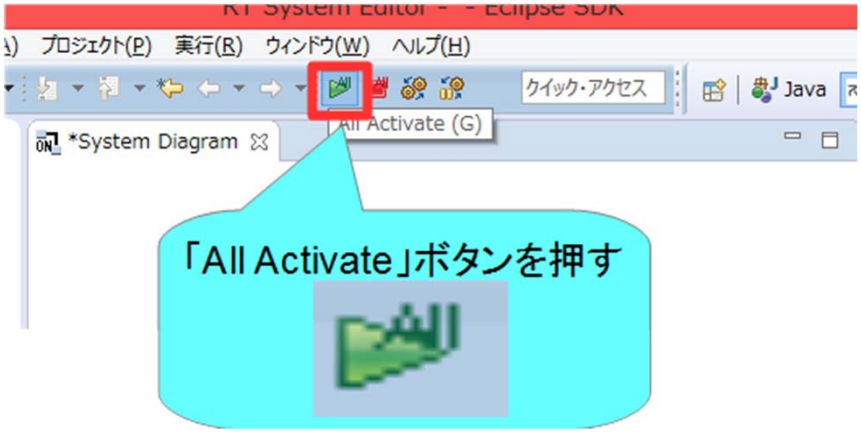


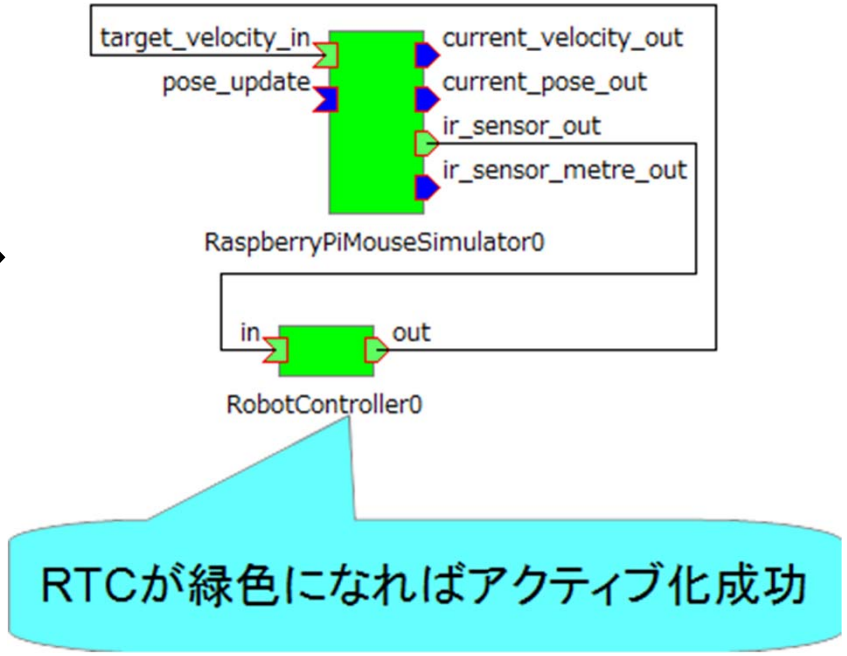
データポートの接続



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

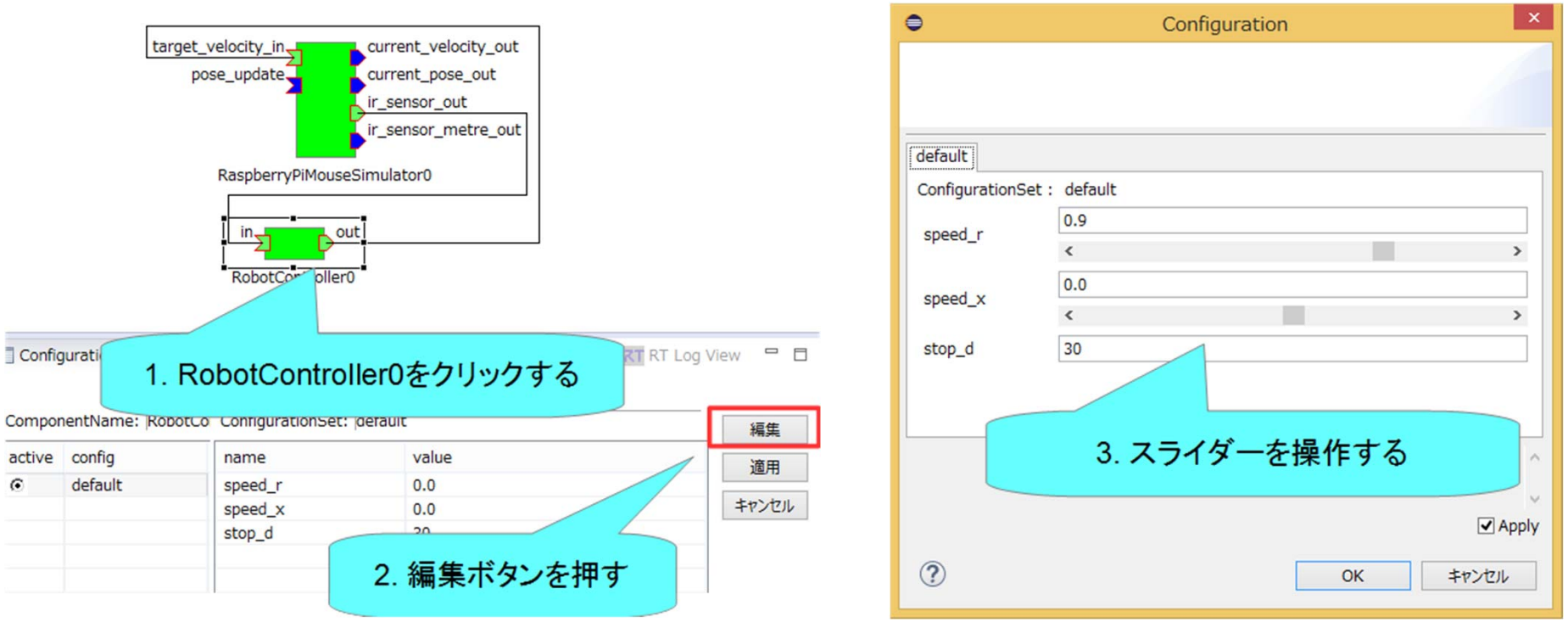
アクティブ化


プロジェクト(P)
実行(R)
ウィンドウ(W)
ヘルプ(H)
クイック・アクセス
Java
All Activate (G)
*System Diagram
「All Activate」ボタンを押す


target_velocity_in
pose_update
current_velocity_out
current_pose_out
ir_sensor_out
ir_sensor_metre_out
RaspberryPiMouseSimulator0
in
out
RobotController0
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

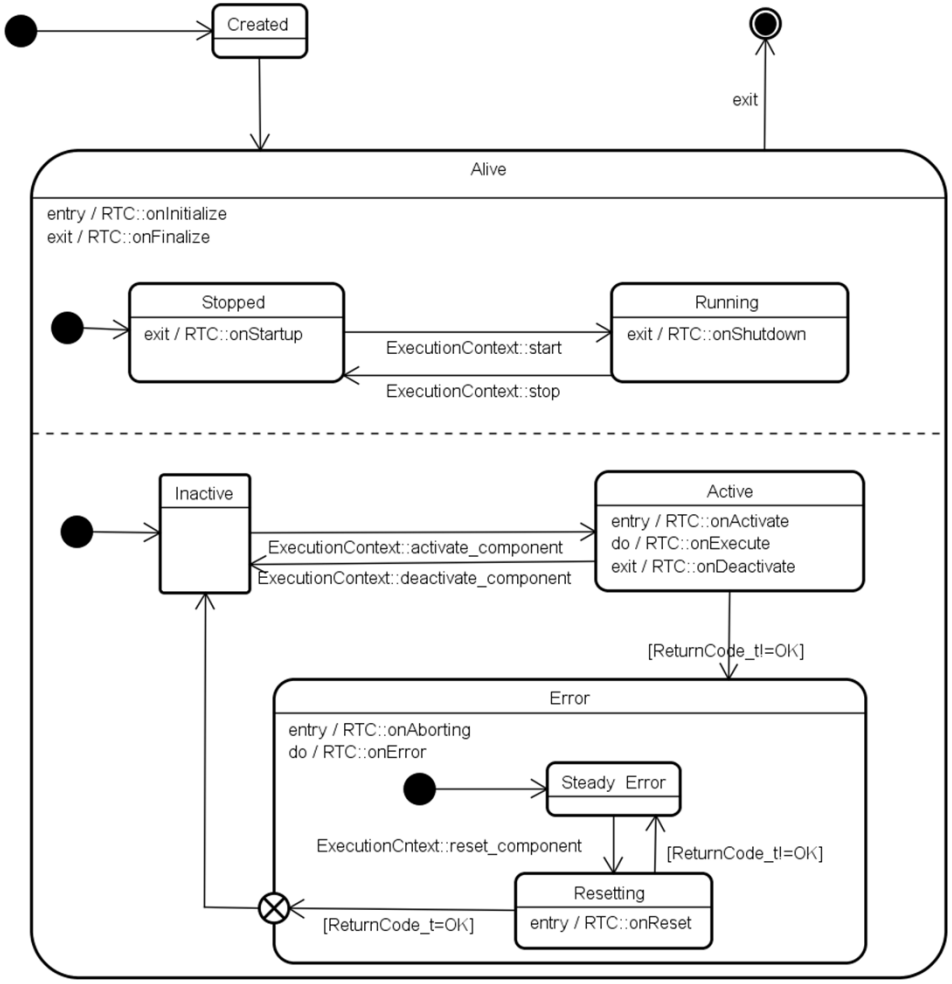
- コンフィギュレーションパラメータをRTシステムエディタから操



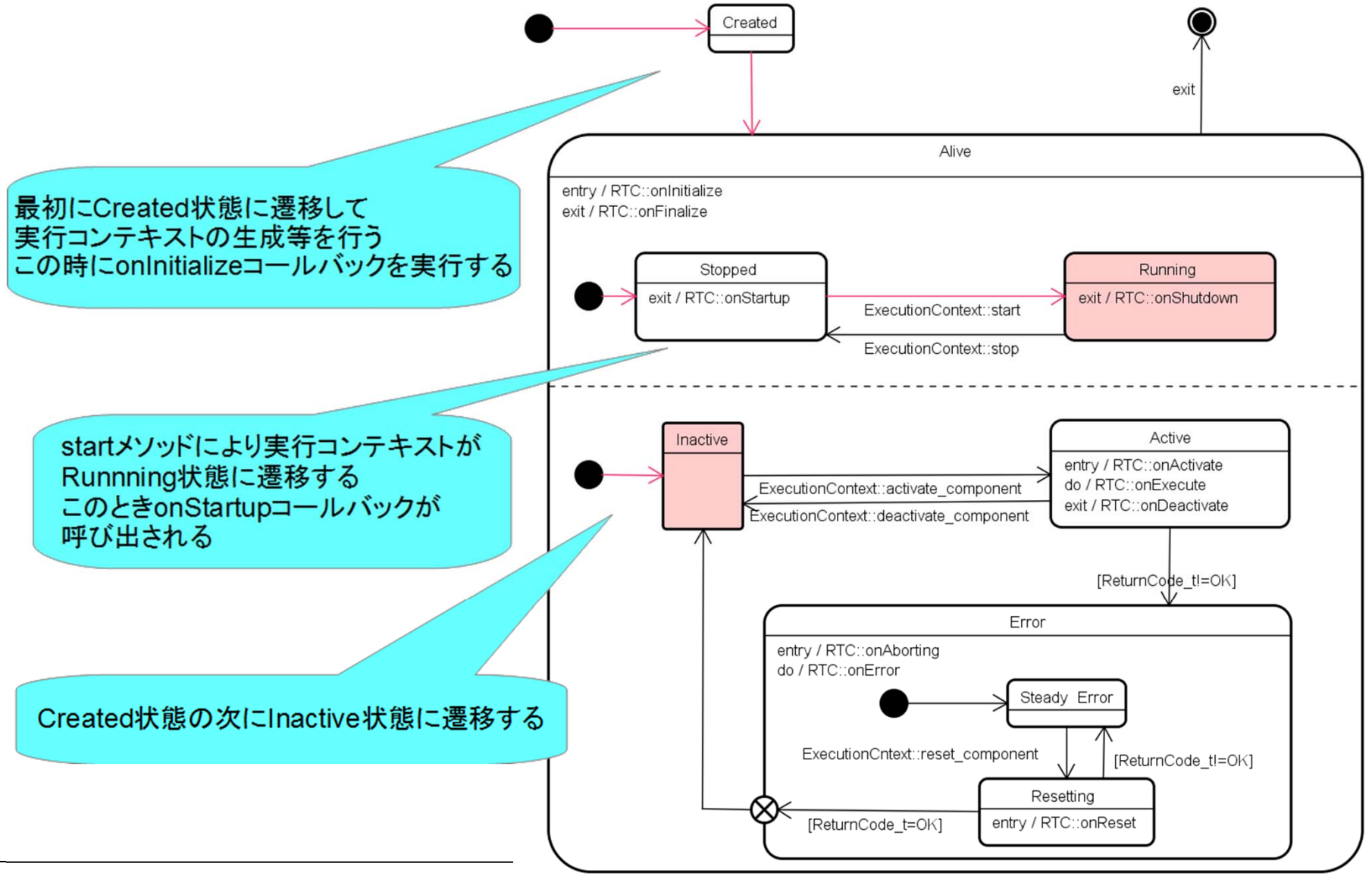
- 以下の動作ができるか確認
  - シミュレータ上のロボットがスライダーで操作できるか？
  - ロボットが障害物に近づくと停止するか？

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



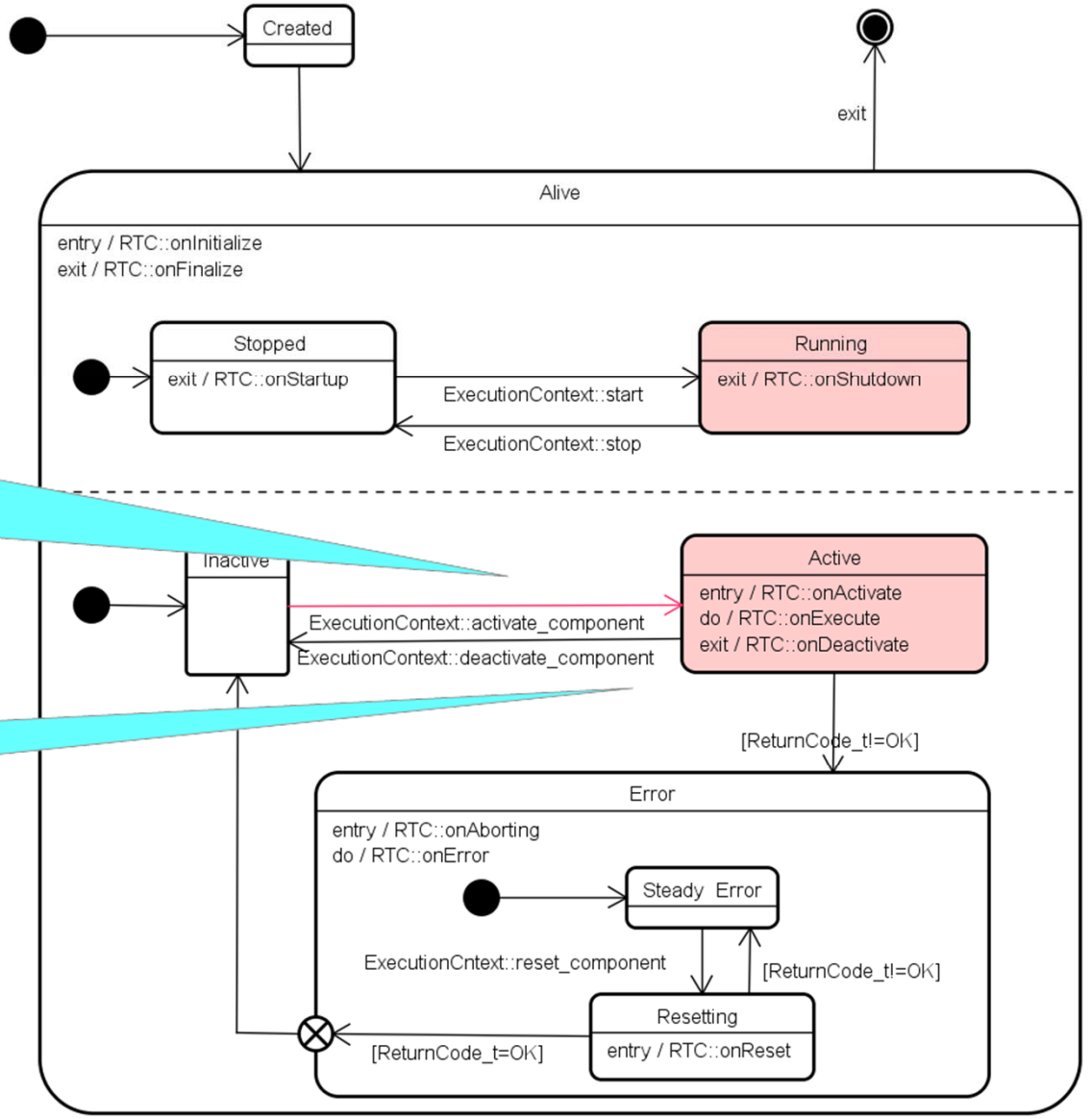
RTコンポーネントの状態遷移(生成直後)



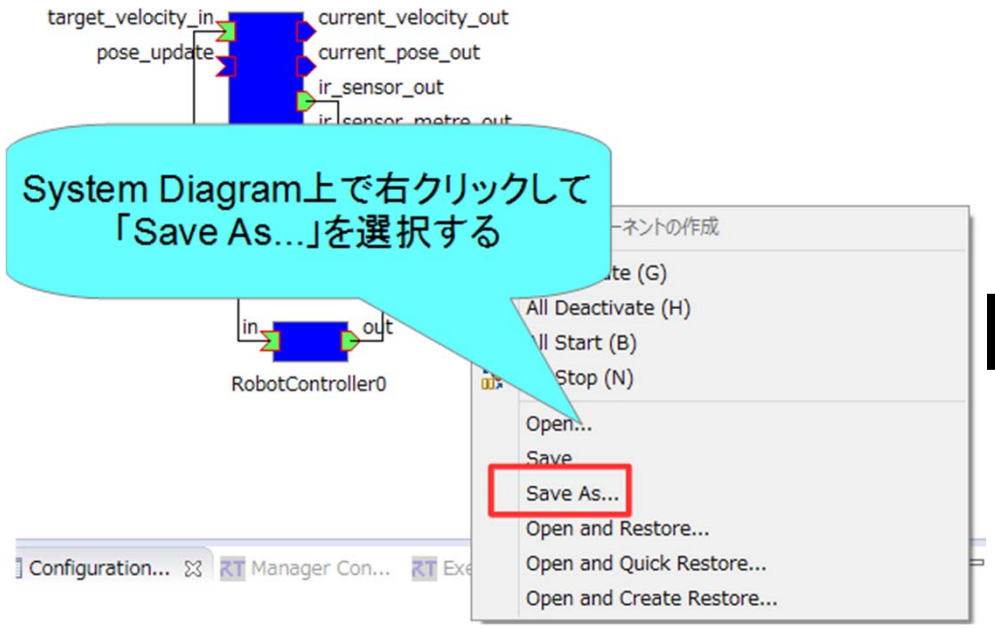
RTコンポーネントの状態遷移(アクティブ化)

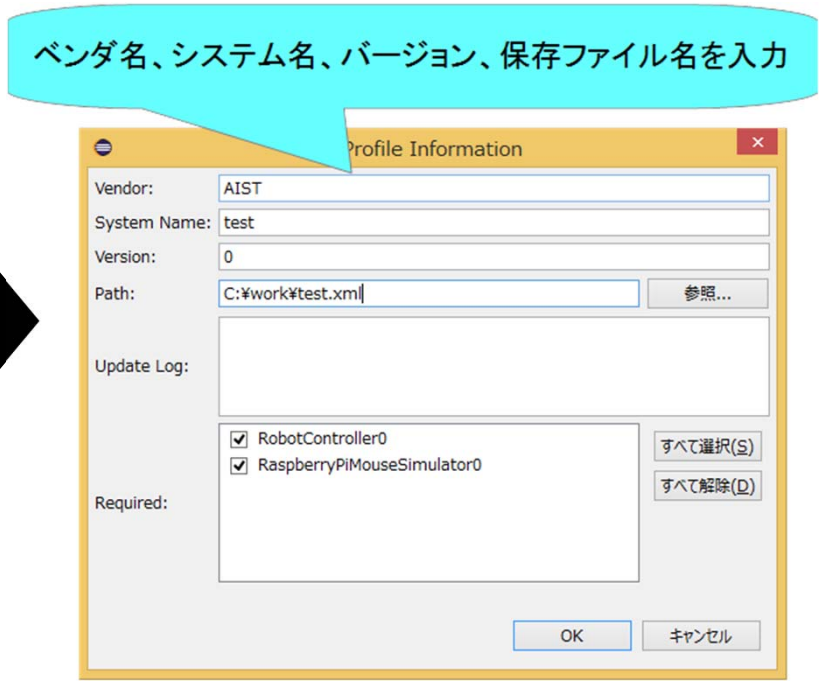
RTシステムエディタの操作によりRTコンポーネントのアクティブ化を行うとactivate_componentメソッドが呼び出される。
activate_componentメソッドによりコンポーネントがActive状態に遷移する。
この時onActivatedコールバックが実行される

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。

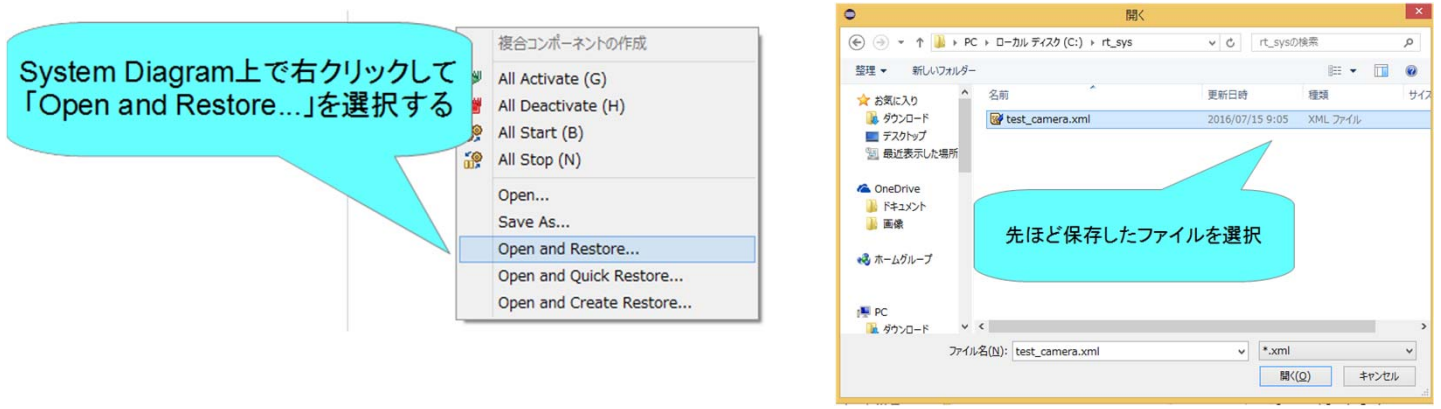


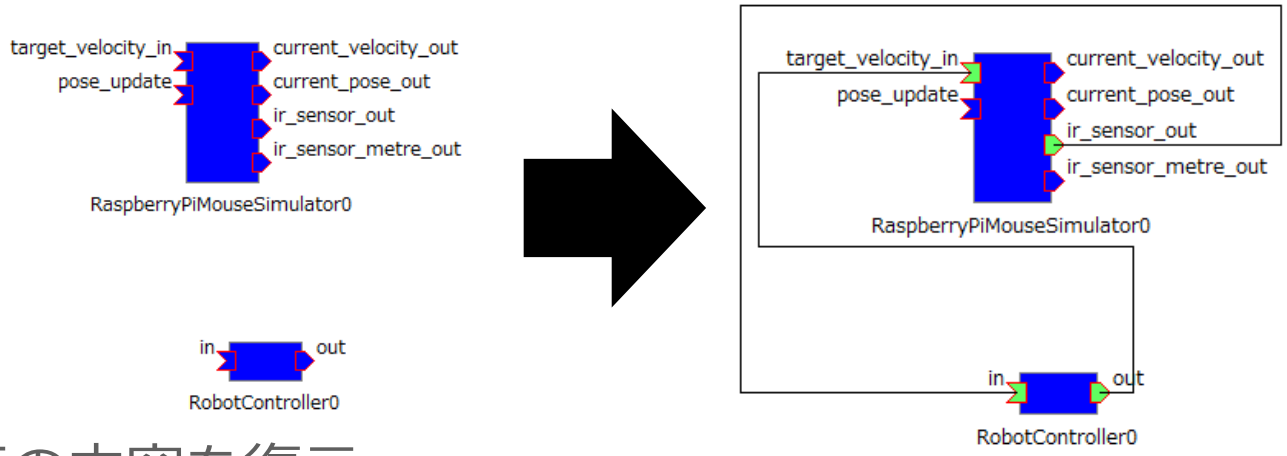
システムの保存


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



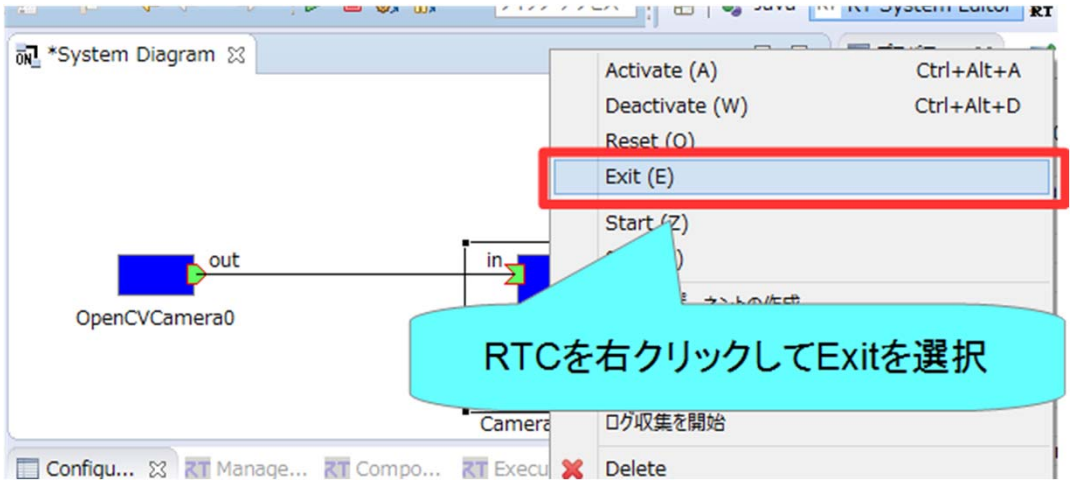


- 以下の内容を復元
  - ポート間の接続
  - コンフィギュレーション
  - 「Open and Create Restore」を選択した場合はマネージャからコンポーネント起動

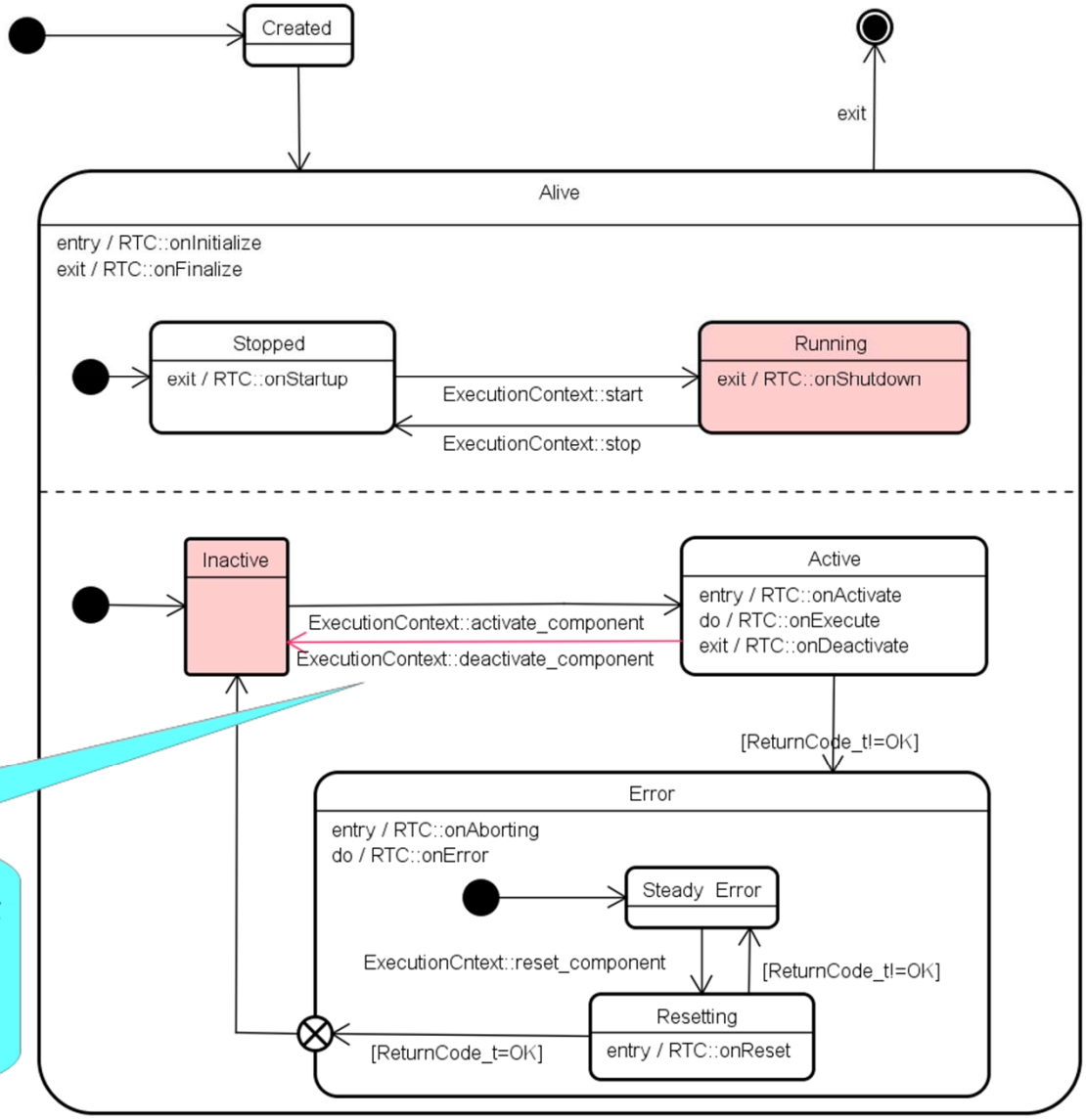
非アクティブ化、終了

- 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)

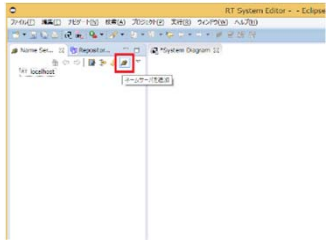


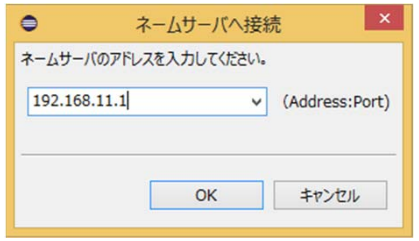
RTシステムエディタの操作によりRTコンポーネントの非アクティブ化を行うとdeactivate_componentメソッドが呼び出される。
deactivate_componentメソッドによりコンポーネントがInactive状態に遷移する。
この時onDeactivatedコールバックが実行される

実機との接続

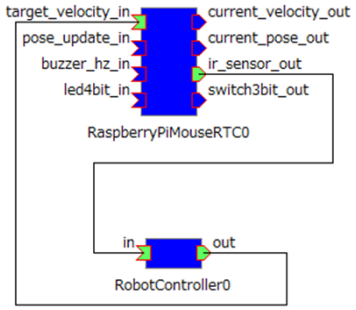
- 今回の講習会ではRaspberry Piマウス実機を2台用意
  - 動作までの手順
    - 無線LANによりRaspberry Piと接続(Raspberry Piがアクセスポイントとして動作)
      - 以下のSSIDに接続
        - » Raspberrypi_12
        - » Raspberrypi_13
          - » pi123openrtm
      - ネットワークを切り替えた際にはネームサーバー、起動中のRTCを再起動してください

- ネームサーバー接続
  - 192.168.11.1



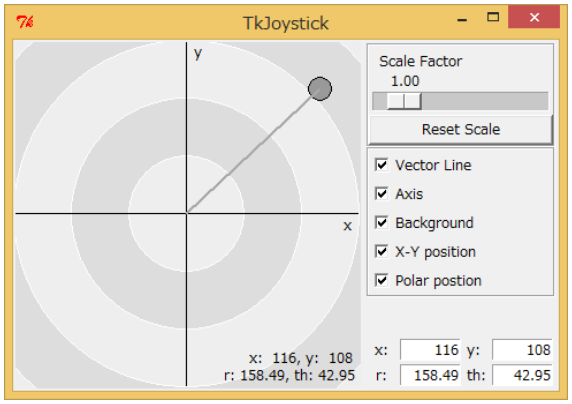


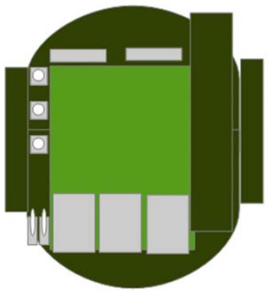
- ポート接続

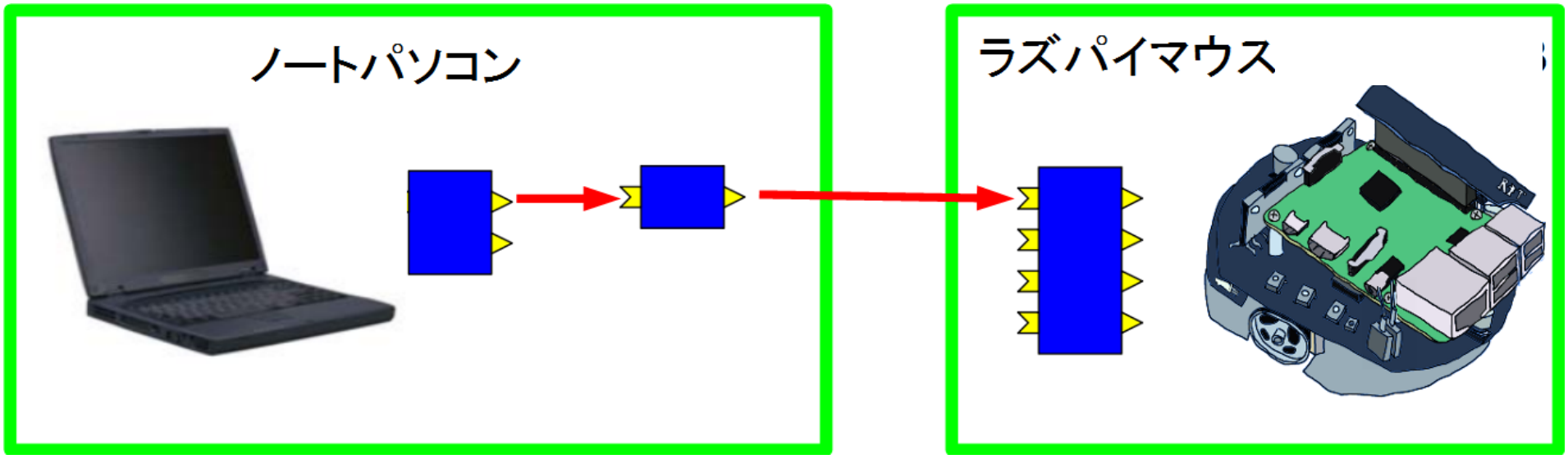


動作確認

- ジョイスティックで操作

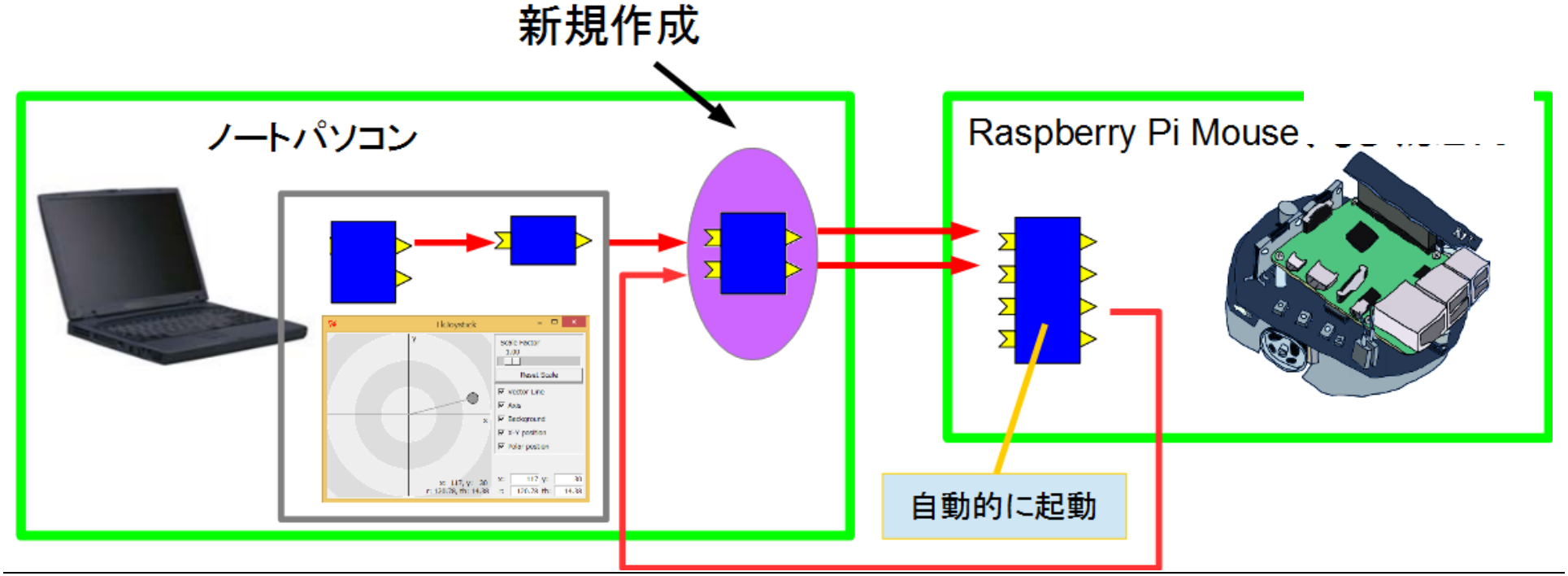






RTCの作成

- ジョイスティックコンポーネントとRaspberrry Pi Mouse(もしくはEV3)制御コンポーネントの間に新規作成したRTCを接続
  - チュートリアルでは簡単なRTCの作成手順を記載してありますが、ただの見本なので自由に作成してください。
  - ラズパイマウスは距離センサに物体がある程度近づくと停止して音を鳴らす動作を行います。



注意事項

- **有線での接続を推奨しています**
  - 一応、無線での接続も可
- ネットワークインターフェースが複数ある場合はトラブルが起こりやすいため、**無線LAN等はオフにしてください。**
- 無線LANをオフにするとインターネットに接続できなくなるため、**チュートリアルのページは保存しておくことをおすすめします。**