

第2部

RTコンポーネント作成入門

宮本 信彦

国立研究開発法人産業技術総合研究所
ロボットイノベーション研究センター
ロボットソフトウェアプラットフォーム研究チーム



資料(USBメモリで配布)

- ppt
 - 第2部
- WEBページ(手順を記載したページ)
 - 第2部(Windows)
 - 第2部(Ubuntu)
- installer
 - OpenRTM-aistのインストールに問題があった場合に使用してください
- script
 - この講習で使うシミュレータのインストールスクリプト(Ubuntu用)
- EXE
 - RaspberryPiMouseSimulatorComp.exe(シミュレータ)
- sample
 - RobotController(本実習で作成するコンポーネントの見本)

インストールの確認(Windows)

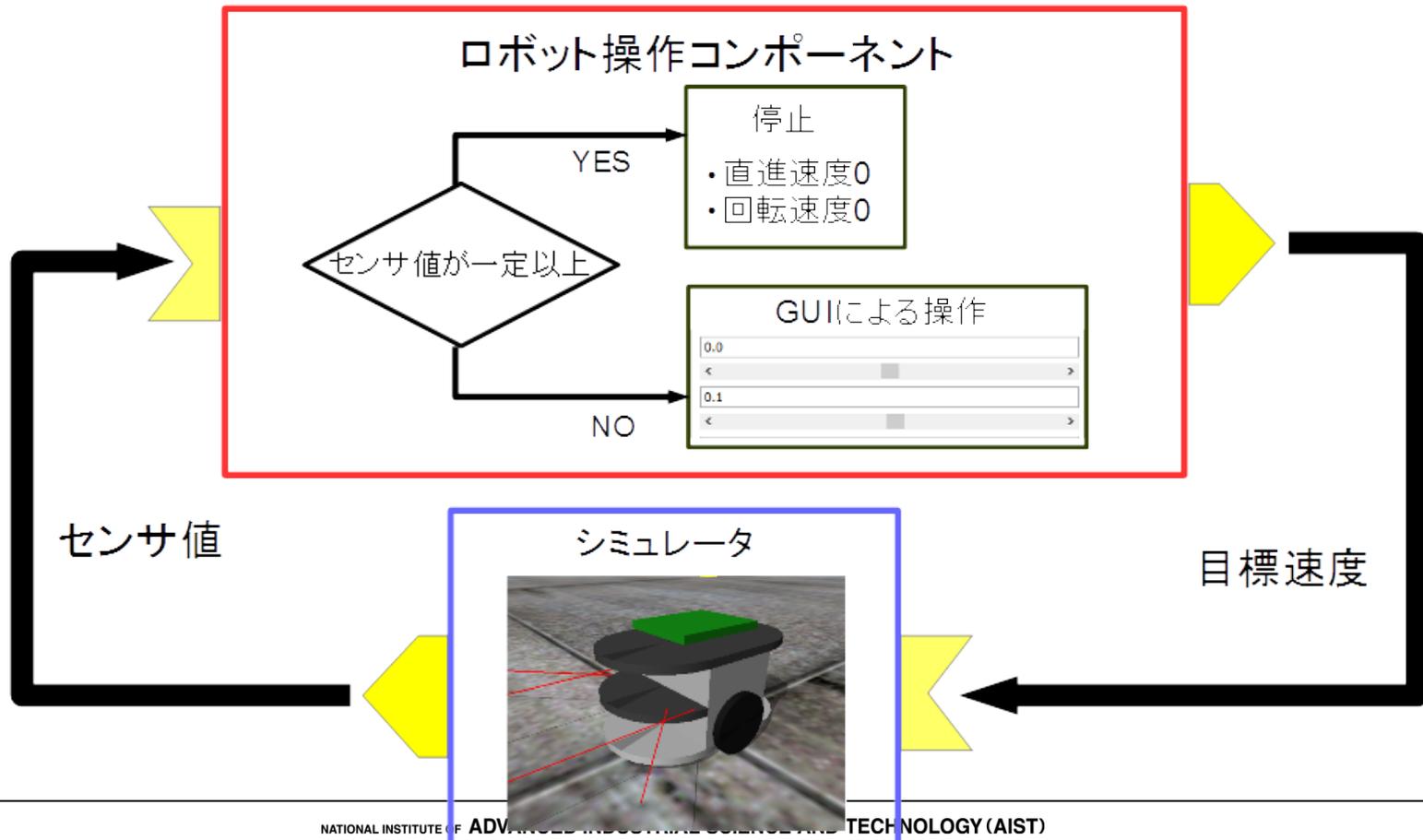
- OpenRTM-aist
 - OpenRTM-aist-1.2.0-RELEASE_x86_64.msi
 - Visual Studio 2019をインストールした場合はOpenRTM-aist-1.2.1-RC190514をインストールする
 - インストール後に再起動する(2回再起動を必要とする環境もある)
- Python
 - python-2.7.16.amd64.msi
 - OpenRTM-aistの64bit版をインストールする場合はPythonも64bit版をインストールする。
- CMake
 - cmake-3.14.1-win64-x64.msi
- Doxygen
 - doxygen-1.8.14-setup.exe
- Visual Studio
 - Visual Studio 2017 Community Edition
 - もしくはVisual Studio 2019
 - vs_installer_2019¥vs_community__2112361810.1542592799_2019.exe

インストールの確認(Ubuntu)

- OpenRTM-aist
 - \$ wget https://raw.githubusercontent.com/OpenRTM/OpenRTM-aist/master/scripts/pkg_install_ubuntu.sh
 - \$ sudo sh [pkg_install_ubuntu.sh](#)
- CMake
 - \$ sudo apt-get install cmake cmake-gui
- Doxygen
 - \$ sudo apt-get install doxygen
- Java
 - \$ sudo apt-get install openjdk-8-jdk
- Code::Blocks(任意)
 - \$ sudo apt-get install codeblocks
- RaspberryPiMouseSimulatorComp
 - \$ wget https://raw.githubusercontent.com/OpenRTM/RTM_Tutorial_RaspberryPiMouse/master/script/install_raspimouse_simulator.sh
 - sudo sh install_raspimouse_simulator.sh

実習内容

- 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

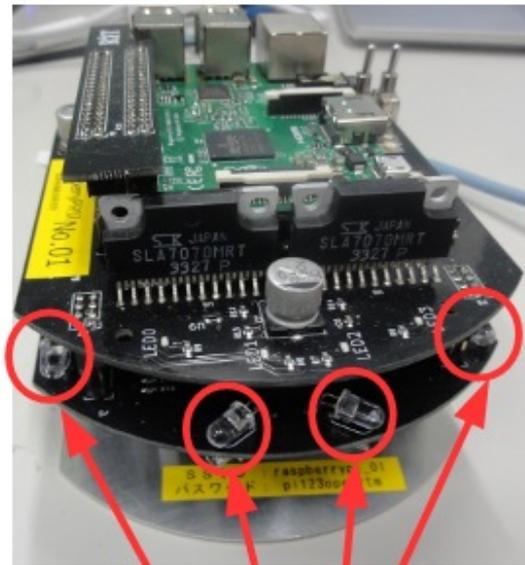


Raspberry Piマウス概要

- Raspberry Piマウスはアールティが販売している独立二輪駆動型の移動ロボット



前面



距離センサ

裏面



駆動車輪

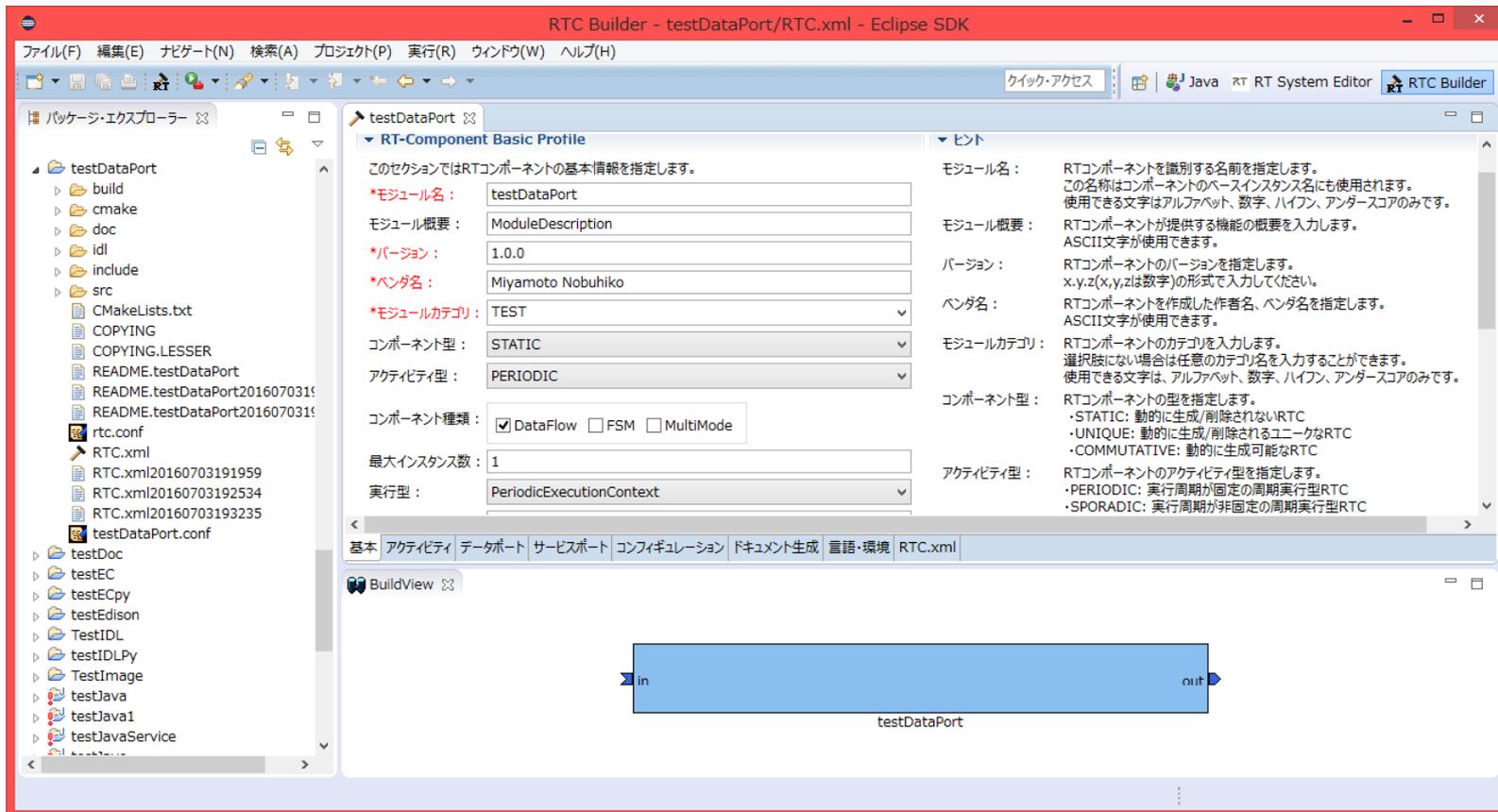
全体の手順

- RTC Builderによるソースコード等のひな型の作成
- ソースコードの編集、ビルド
 - ビルドに必要な各種ファイルを生成
 - CMakeにより各種ファイル生成
 - ソースコードの編集
 - 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、Luaのソースコードを出力



RTC Builderの起動

- 起動する手順

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



- デスクトップのショートカットがない場合

- Windows 7
 - 「スタート」→「すべてのプログラム」→「OpenRTM-aist 1.2.0」→「Tools」→「OpenRTP」
- Windows 8.1
 - 「スタート」→「アプリビュー(右下矢印)」→「OpenRTM-aist 1.2.0」→「OpenRTP」
 - ※同じフォルダに「RTSystemEditorRCP」がありますが、これはRTC Builderが使えないので今回は「OpenRTP」を起動してください。
- Windows 10
 - 左下の「ここに入力して検索」にOpenRTPと入力して、表示されたOpenRTPを起動

- Ubuntu

- 以下のコマンドを入力
- \$ openrtp

RTC Builderの起動

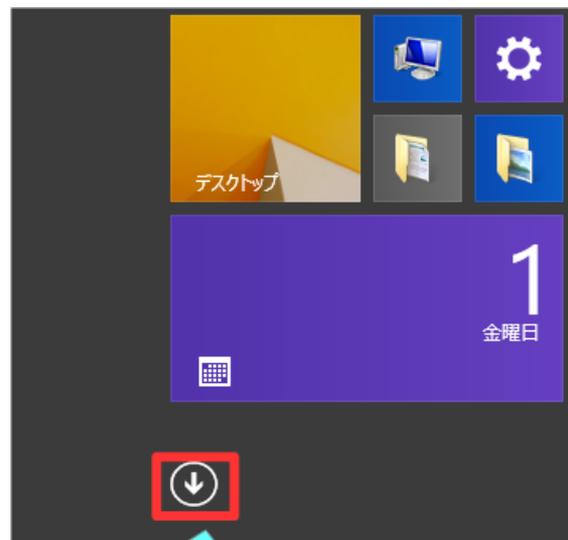
- Windows 8.1 (OpenRTM-aist 1.1.2以前)

デスクトップ



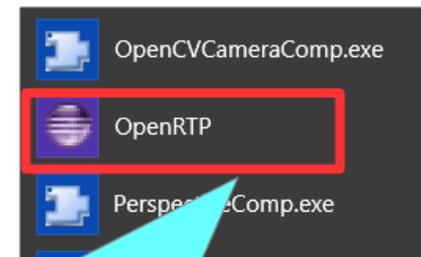
画面左下のアイコンをクリック

スタート画面



画面左下の矢印をクリック

アプリビュー



OpenRTPをクリック

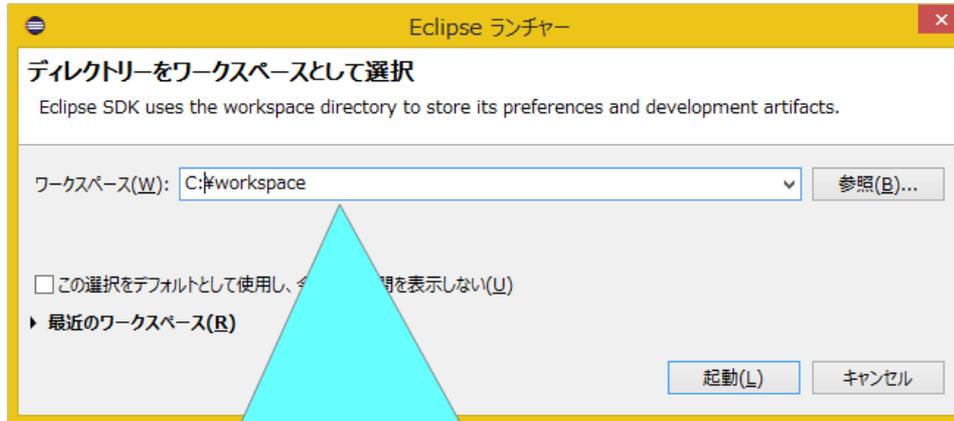
RTC Builderの起動

- Windows 10(OpenRTM-aist 1.1.2以前)



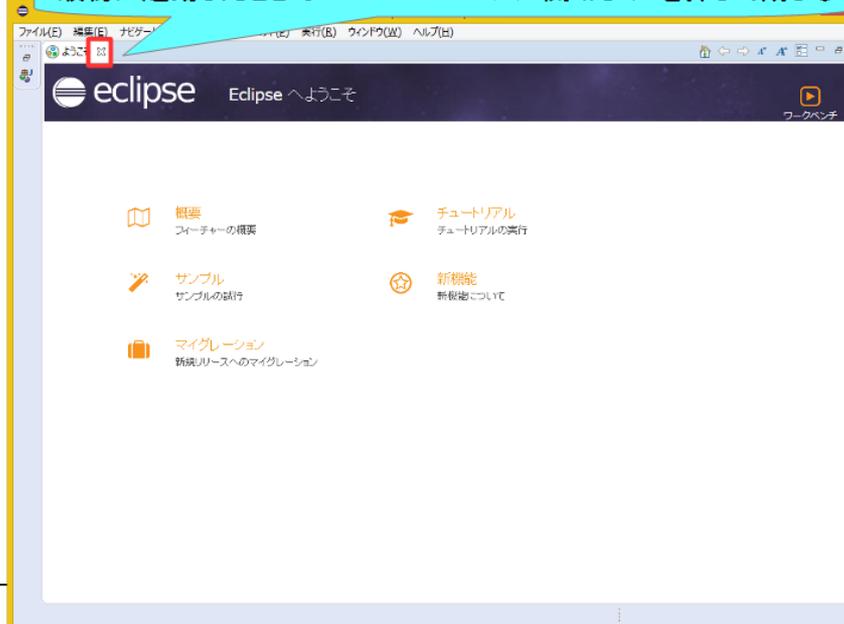
左下の「ここに入力して検索」に「OpenRTP」と入力

RTC Builderの起動



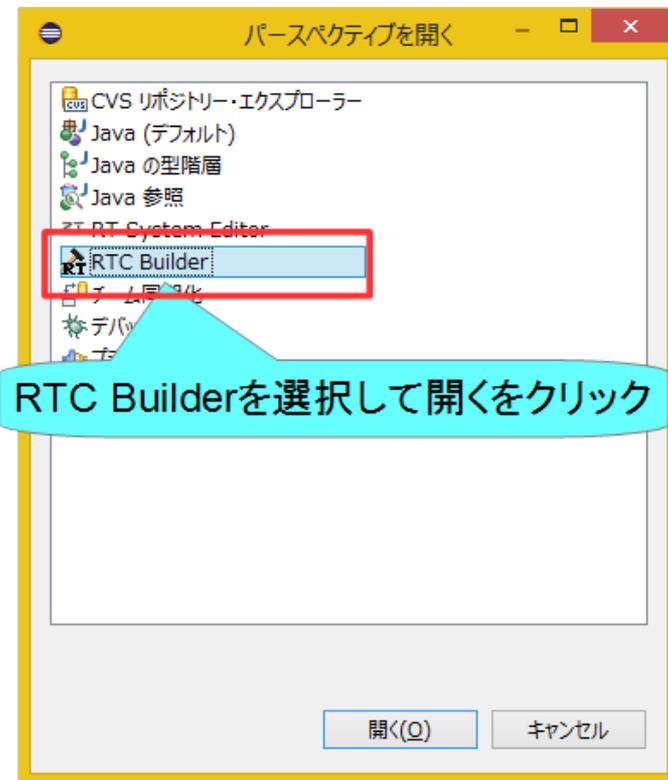
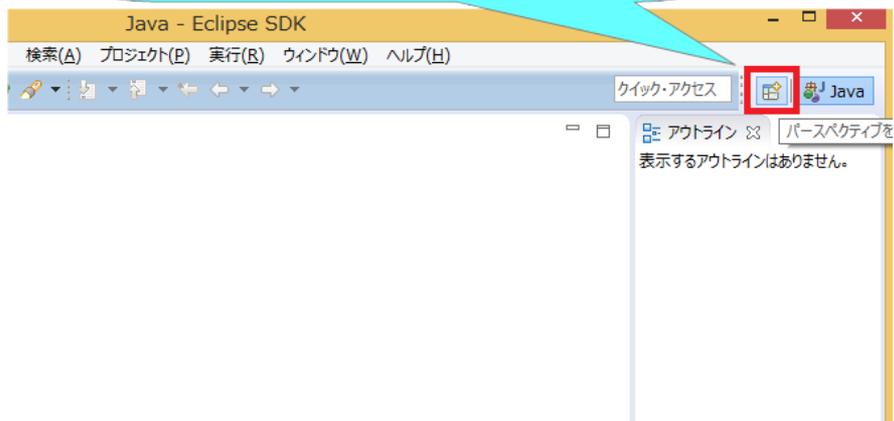
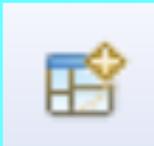
ワークスペースに適切な場所を指定して起動をクリックする

最初に起動したときはwelcomeページが開くため×を押して閉じる



RTC Builderの起動

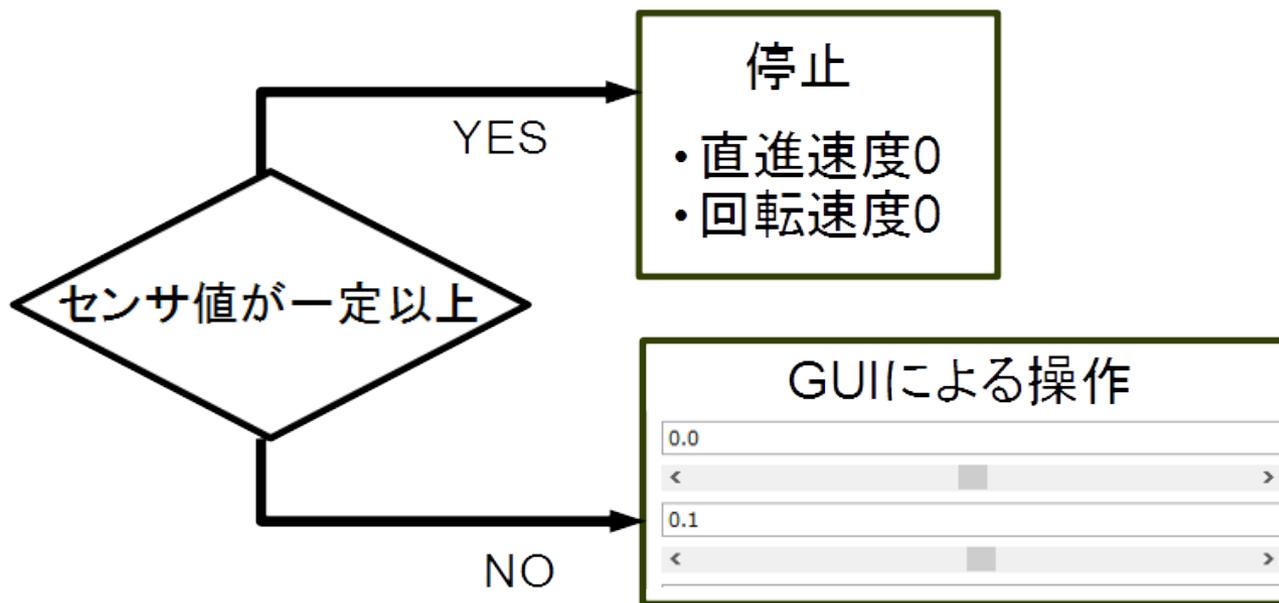
右上の「パースペクティブを開く」ボタンをクリック



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

ロボット操作コンポーネント



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
 - チュートリアル(Raspberry Pi Mouse、C++、Windows、強化月間用) _ OpenRTM-aist.html
 - チュートリアル(Raspberry Pi Mouse、C++、Windows、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(第2部、Windows)
 - チュートリアル(第2部、Ubuntu)

プログラム(プログラムは変更になる可能性があります)

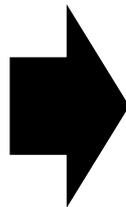
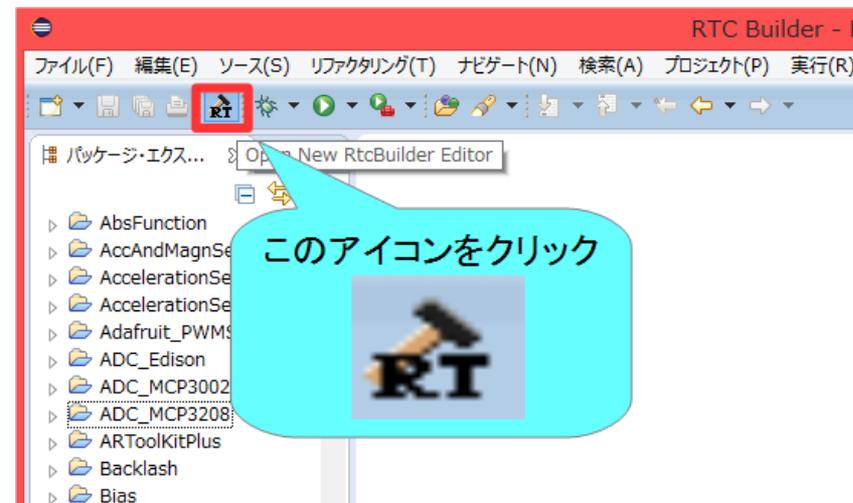
時間	概要
13:00 -14:00	<p>第1部：RTミドルウェア: OpenRTM-aist概要 担当：安藤慶昭(産総研) 概要：RTミドルウェアはロボットシステムをコンポーネント指向で構築するソフトウェア。RTミドルウェアを利用することで、既存のコンポーネントを再利用し、モジュール指向の柔軟なロボットシステムを構築することができます。RTミドルウェアの産総研 aistについてその概要について説明します。 講義資料：</p>
14:15 -17:00	<p>第2部: RTコンポーネントの作成入門 担当：宮本信彦(産総研) 概要： RTシステムを設計するツールRTSystemEditorおよび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、Windows編) チュートリアル (Raspberry Pi Mouse、Ubuntu編) 講義資料：</p>

資料

説明資料等を講習会当日にUSBメモリで配布する予定ですが、何らかの理由によりUSBメモリを利用できない場合は以下のZIPファイルをダウンロードしてください。

- RTM_Tutorial_RaspberryPiMouse.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

RTCプロファイルエディタ
ここに各項目を入力する

ヒント

基本

▼ RT-Component Basic Profile

このセクションではRTコンポーネントの基本情報を指定します。

*モジュール名: ModuleName

モジュール概要: ModuleDescription

*バージョン: 1.0.0

*ベンダ名: Miyamoto Nobuhiko

*モジュールカテゴリ: TEST

コンポーネント型: STATIC

アクティビティ型: PERIODIC

コンポーネント種類: DataFlow FSM MultiMode

最大インスタンス数: 1

▼ ヒント

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

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

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

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

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

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

アクティビティ型: RTコンポーネントのアクティビティ型を指定します。

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

「基本」タブを選択

基本プロファイルの入力

- **モジュール名**
 - **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

基本

▼ RT-Component Basic Profile

このセクションではRTコンポーネントの基本情報を指定します。

*モジュール名 : RobotController

モジュール概要 : Robot Controller Component

*バージョン : 1.0.0

*ベンダ名 : AIST

*モジュールカテゴリ : Controller ▼

コンポーネント型 : STATIC ▼

アクティビティ型 : PERIODIC ▼

コンポーネント種類 : DataFlow FSM MultiMode

最大インスタンス数 : 1

実行型 : PeriodicExecutionContext ▼

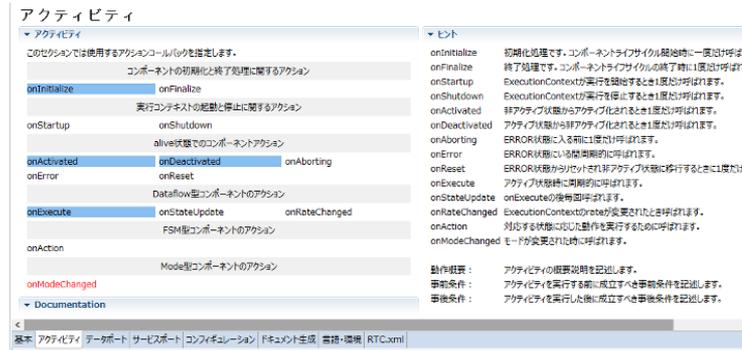
実行周期 : 1000.0

概要 : 講習会用Raspberry Piマウス操作コンポーネント ▲▼

RTC Type :

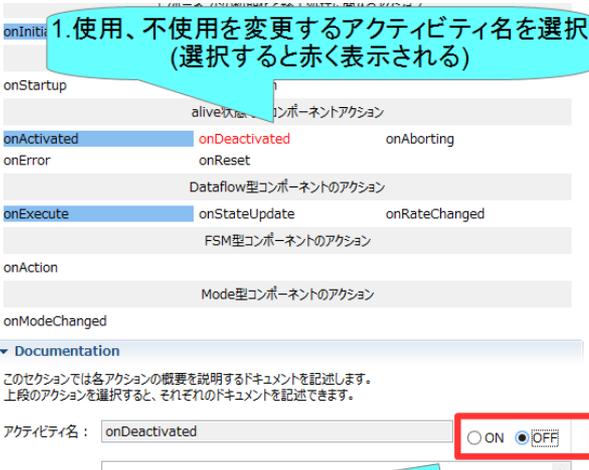
アクティビティの設定

- 使用するアクティビティを設定する

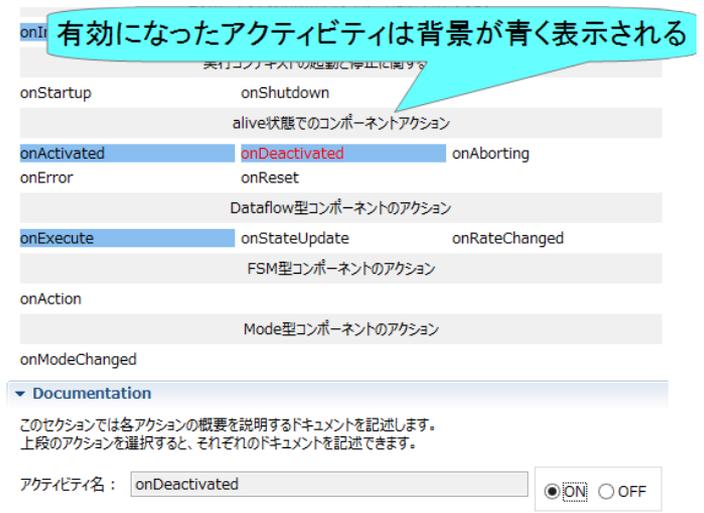


「アクティビティ」タブを選択

- 指定アクティビティを有効にする手順



2. アクティビティ名の選択後、ON・OFFを選択する



アクティビティの設定

コールバック関数	処理
onInitialize	初期化処理
onActivated	アクティブ化される時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時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

▼ アクティビティ

このセクションでは使用するアクションコールバックを指定します。

コンポーネントの初期化と終了処理に関するアクション

onInitialize onFinalize

実行コンテキストの起動と停止に関するアクション

onStartup onShutdown

alive状態でのコンポーネントアクション

onActivated onDeactivated onAborting

onError onReset

Dataflow型コンポーネントのアクション

onExecute onStateUpdate onRateChanged

FSM型コンポーネントのアクション

onAction

Mode型コンポーネントのアクション

onModeChanged

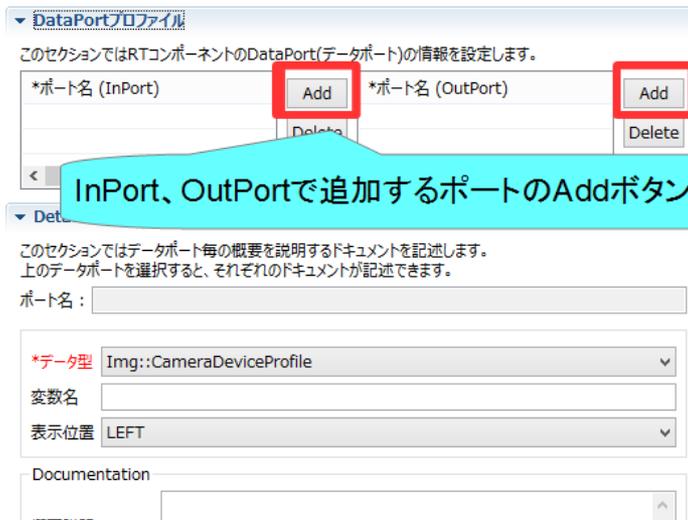
データポートの設定

- InPort、OutPortの追加、設定を行う

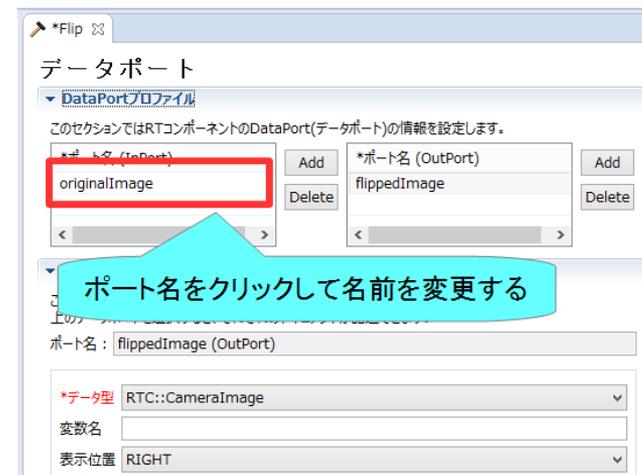


「データポート」タブを選択

- データポートを追加する手順



InPort、OutPortで追加するポートのAddボタンをクリック



ポート名をクリックして名前を変更する

各項目を設定する

データポートの設定

- 以下のOutPortを設定する

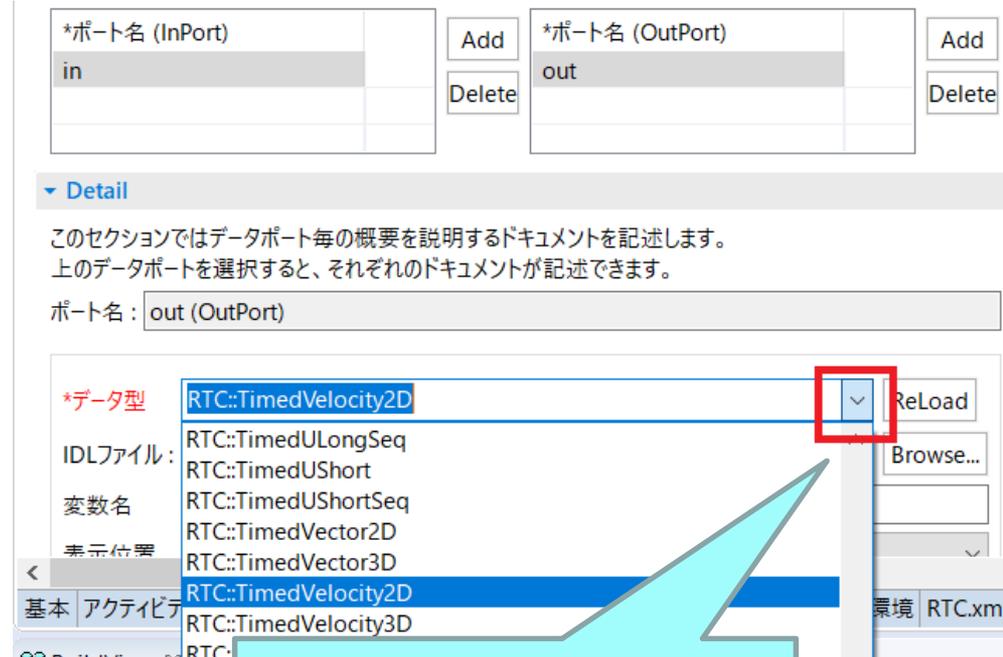
- out

- 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

- 以下のInPortを設定する

- in

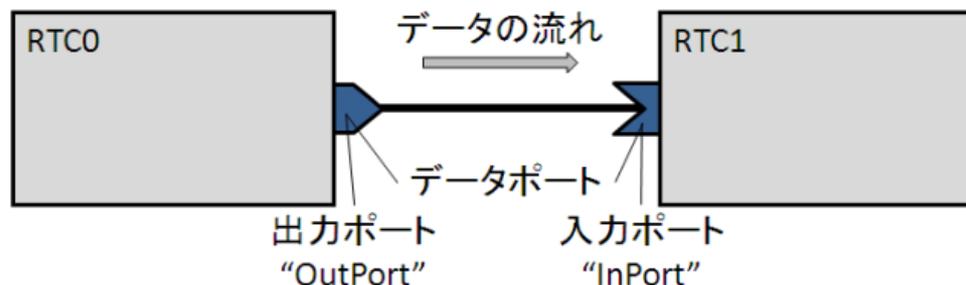
- データ型:
RTC::TimedShortSeq
 - 他の項目は任意
 - ※TimedShort型と間違えないようにしてください。



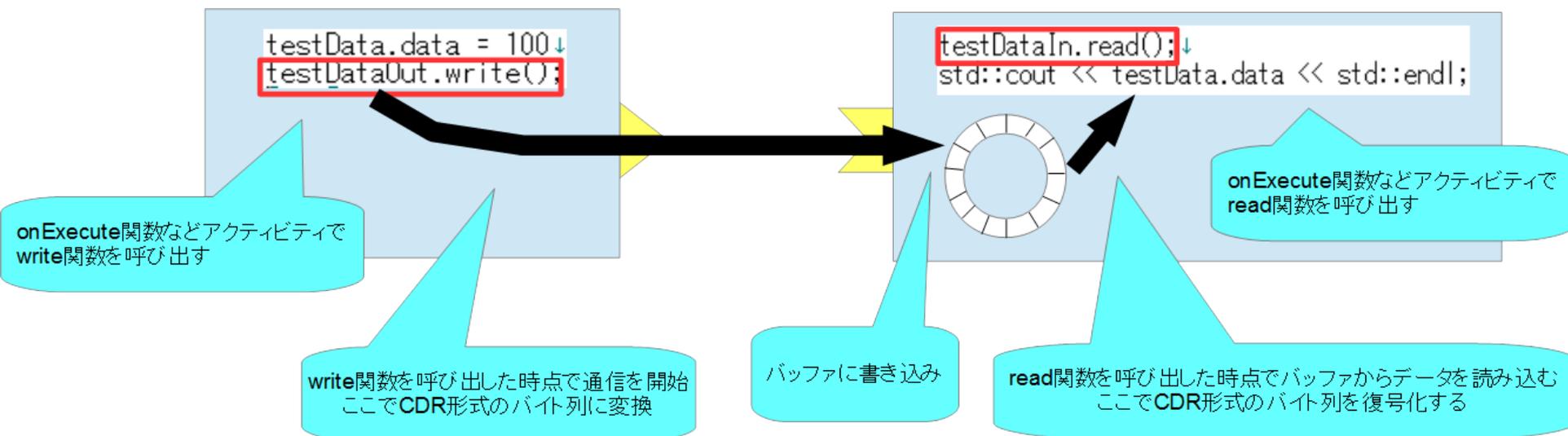
The screenshot displays the configuration interface for data ports. At the top, there are two tables: one for InPort (with 'in' listed) and one for OutPort (with 'out' listed), each with 'Add' and 'Delete' buttons. Below these is a 'Detail' section containing a document description and a text field for the port name, currently set to 'out (OutPort)'. A dropdown menu is open, showing a list of data types including 'RTC::TimedVelocity2D', 'RTC::TimedULongSeq', 'RTC::TimedUShort', 'RTC::TimedUShortSeq', 'RTC::TimedVector2D', 'RTC::TimedVector3D', 'RTC::TimedVelocity2D', and 'RTC::TimedVelocity3D'. A red box highlights the dropdown arrow, and a callout box states: 'データ型はドロップダウンリストから選択する' (Data type is selected from the dropdown list).

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

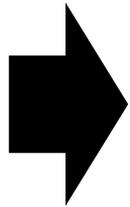


- 以下の例はデータフロー型がpush、サブスクリプション型がflush、インターフェース型がcorba_cdrの場合



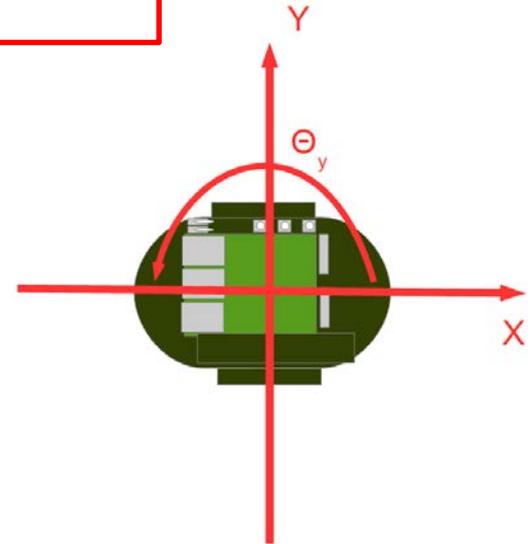
RTC::TimedVelocity2D型について

- 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



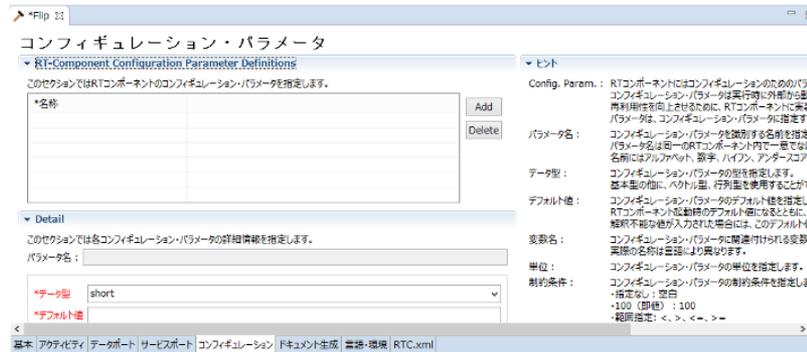
vxで直進速度、**va**で回転速度を設定

```
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;↓
};↓
```

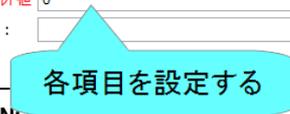
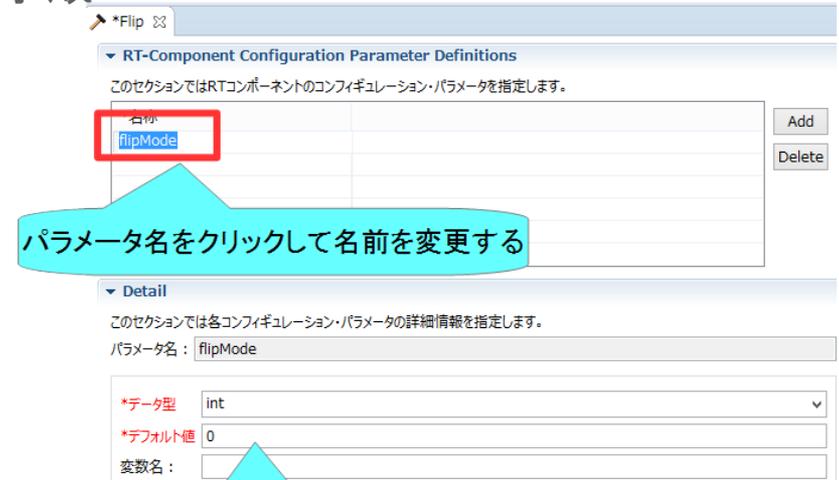
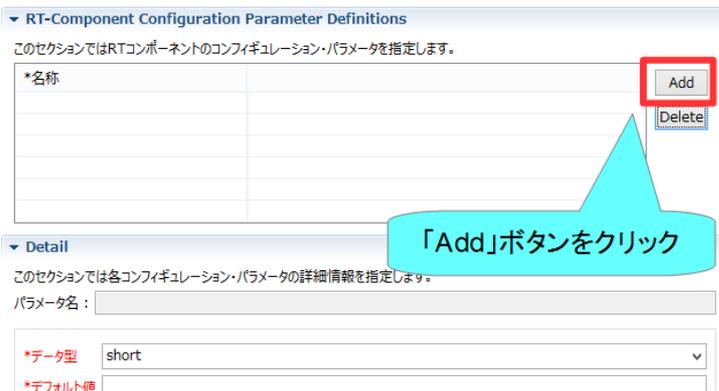


コンフィギュレーションの設定

- 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

RT-Component Configuration Parameter Definitions

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

*名称	Add
speed_x	
speed_r	
stop_d	

Delete

Detail

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

パラメータ名: speed_x

*データ型	double
*デフォルト値	0.0
変数名	
単位	m/s
制約条件	$-1.5 < x < 1.5$
Widget	slider
Step	0.01

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

0.0

< >

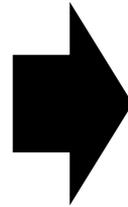
0.1

< >

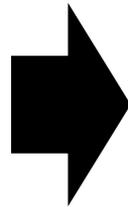
コンフィギュレーションパラメータの制約、Widgetの設定

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

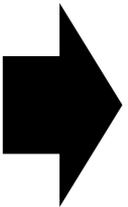
- Widget: text



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

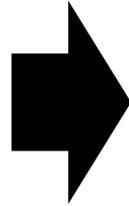


- 制約条件: $0 \leq x \leq 100$
- Widget: slider
- Step: 10

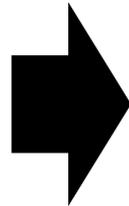


コンフィギュレーションパラメータの制約、Widgetの設定

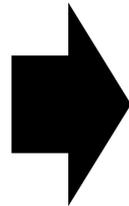
- 制約条件: (0,1,2,3)
- Widget: radio


 0 1 2
 3

- 制約条件: (0,1,2,3)
- Widget: checkbox


 0 1 2
 3

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



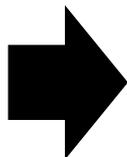
0			0
1		>	1
2		<	
3			^
			v

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

コンフィギュレーション・パラメータ

▼ RT-Component Configuration Parameter Definitions

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

*名称		
speed_x		
speed_r		
stop_d		

Add
Delete

▼ Detail

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

パラメータ名: stop_d

*データ型: int

*デフォルト値: 30

変数名:

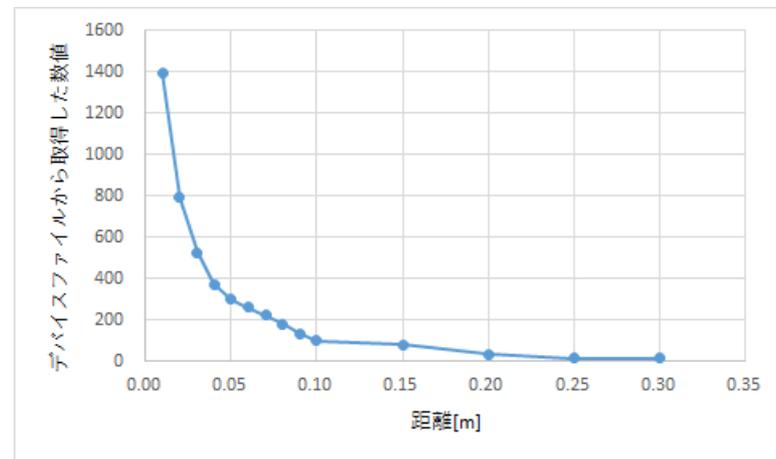
単位:

制約条件:

Widget: text

Raspberry Piマウスの距離センサ

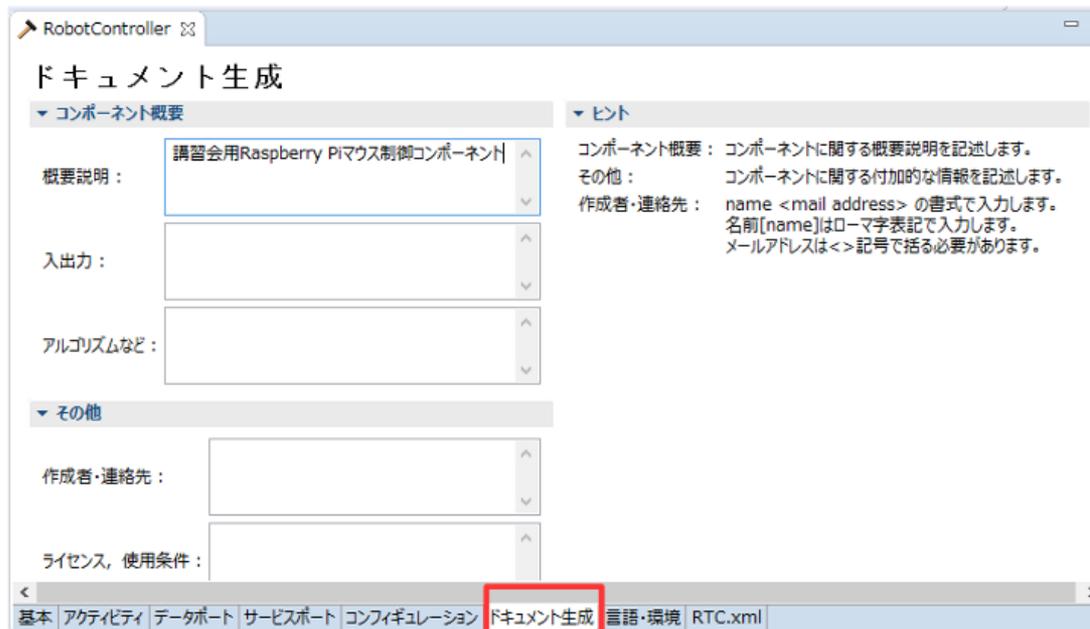
- Raspberry Piマウス実機には距離センサが搭載されている
 - 計測した値は物体までの**距離が近いほど大きな値**となる



- 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

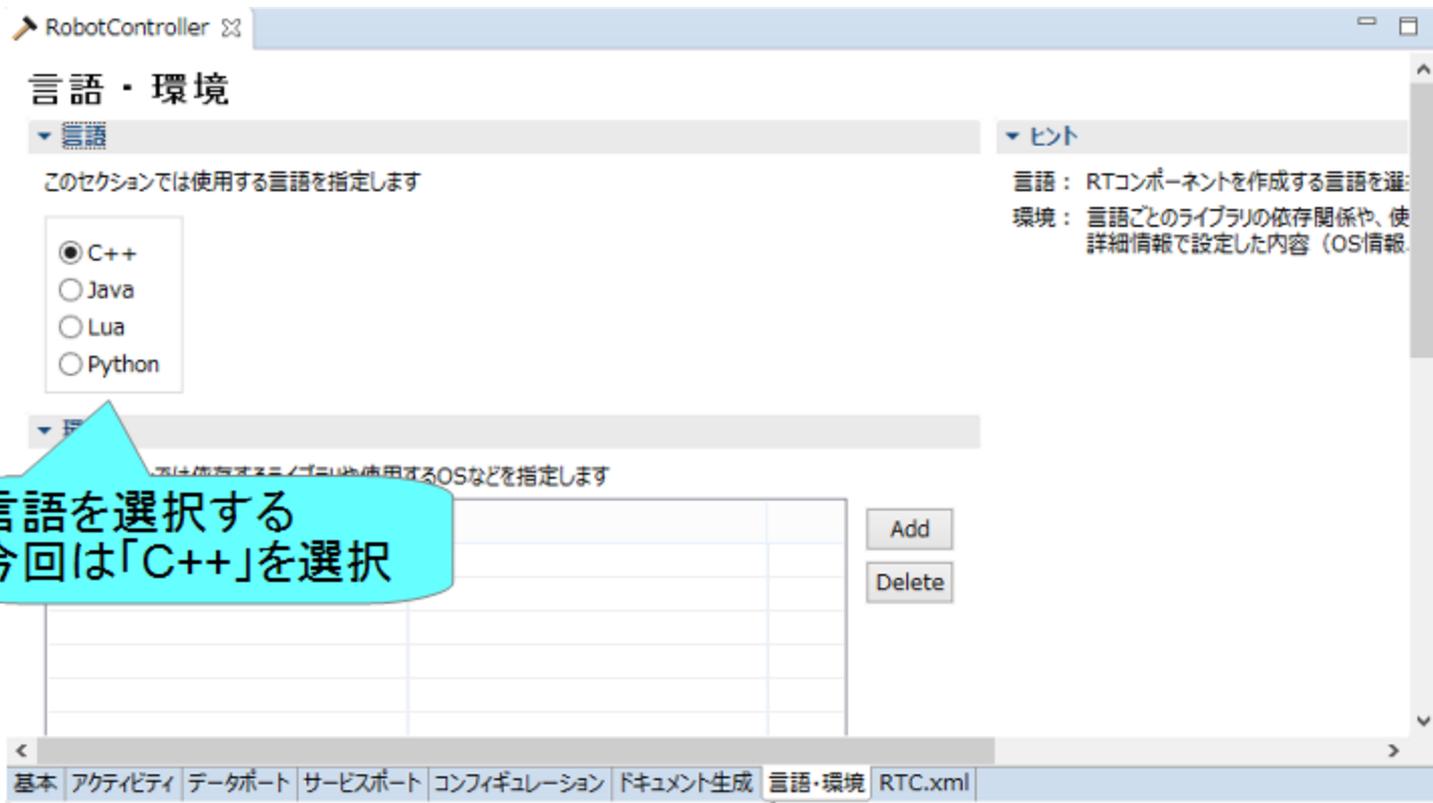


「ドキュメント生成」タブを選択

- 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

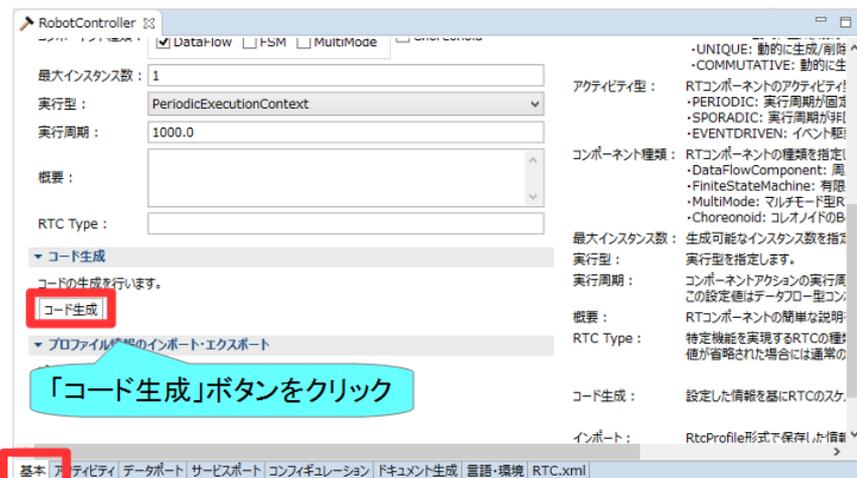


言語を選択する
今回は「C++」を選択

「言語・環境」タブを選択

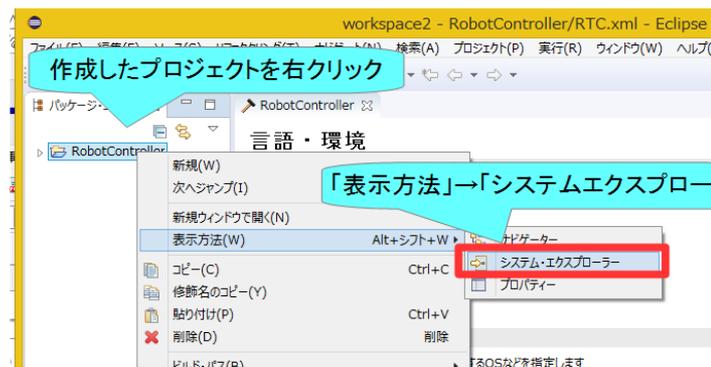
スケルトンコードの生成

- 基本タブからコード生成ボタンを押すことでスケルトンコードが生成される
 - Workspace¥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



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

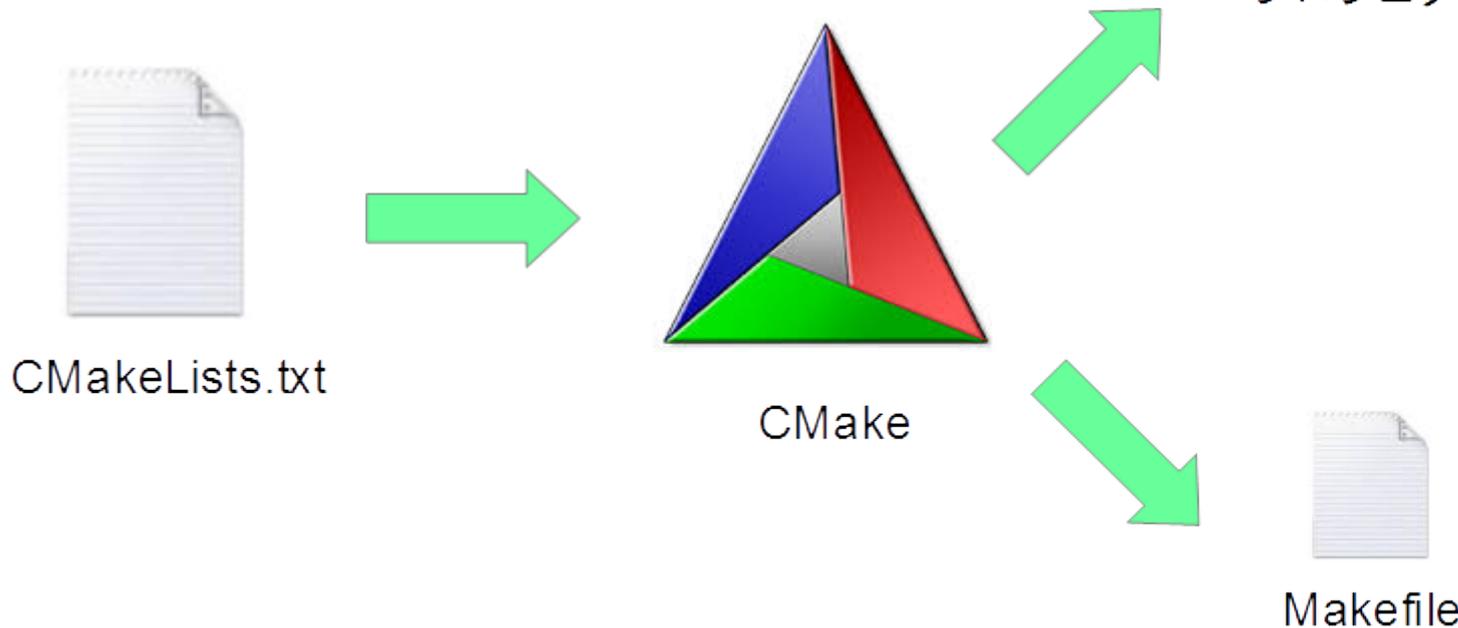
ソースコードの編集、RTCのビルド

CMake

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

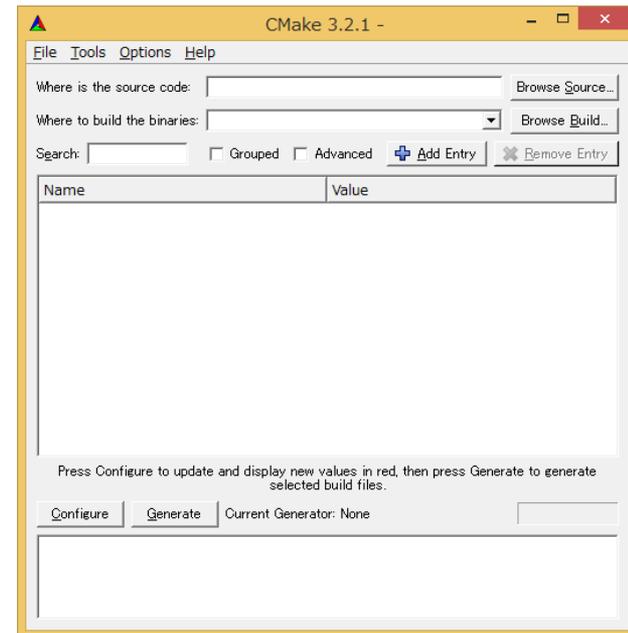


Visual Studio
(ソリューションファイル、
プロジェクトファイル等)



ビルドに必要なファイルの生成

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



cmake-guiの起動

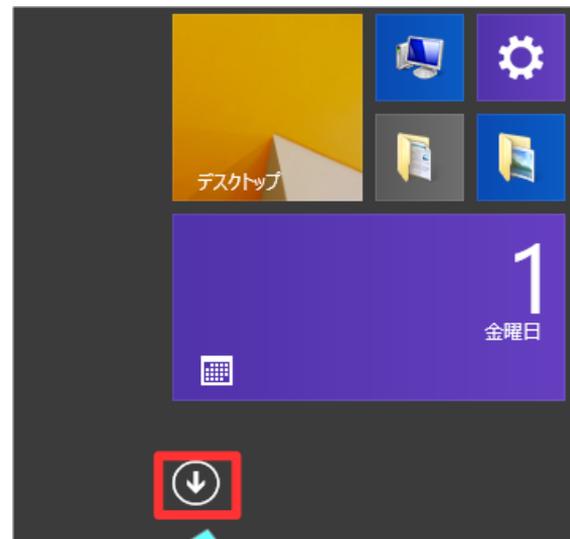
- Windows 8.1

デスクトップ



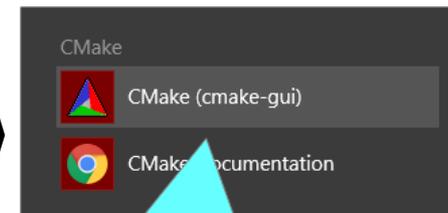
画面左下のアイコンをクリック

スタート画面



画面左下の矢印をクリック

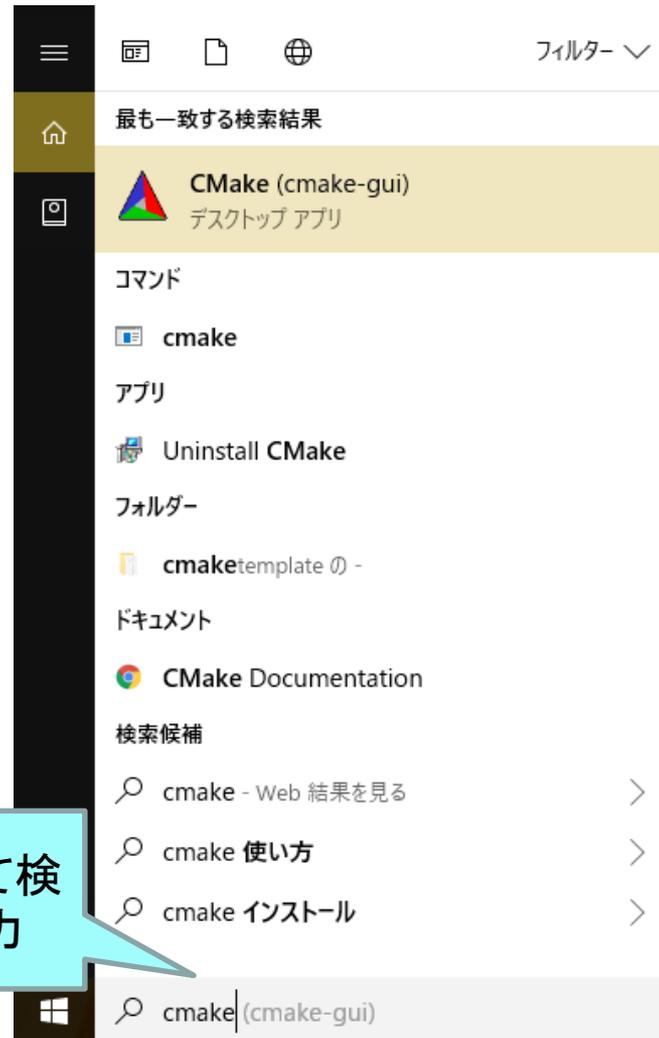
アプリビュー



CMake(cmake-gui)をクリック

cmake-guiの起動

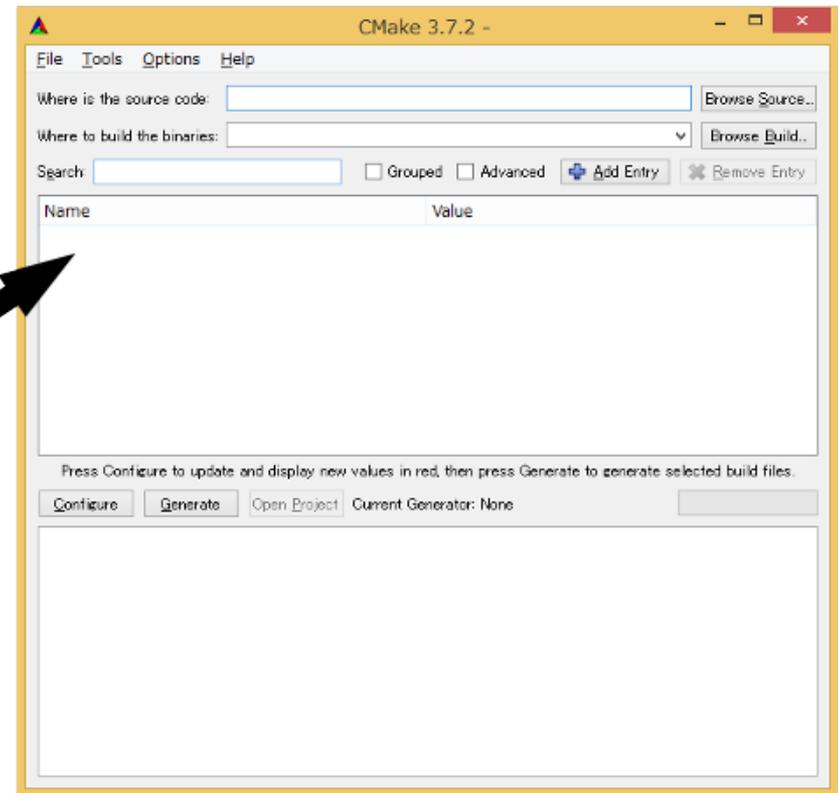
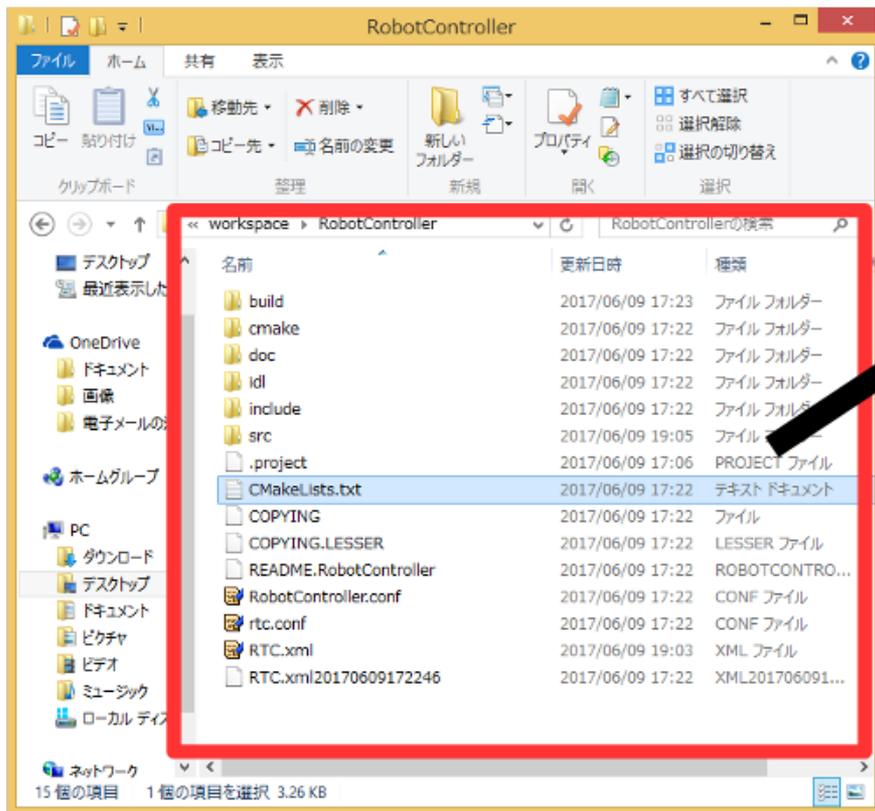
- Windows 10



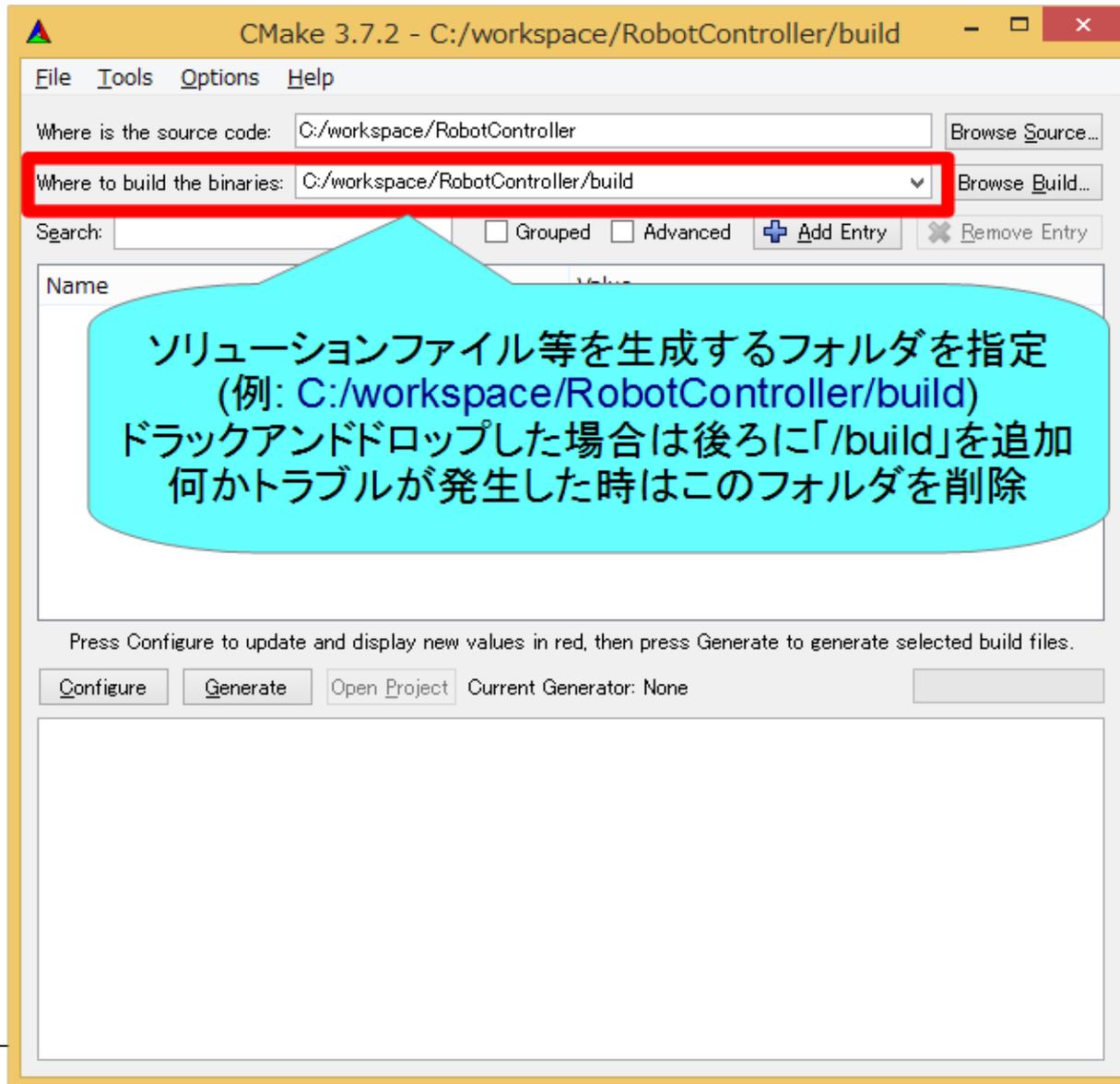
左下の「ここに入力して検索」に「cmake」と入力

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



CMake 3.7.2 - C:/workspace/RobotController/build

File Tools Options Help

Where is the source code: C:/workspace/RobotController Browse Source...

Where to build the binaries: C:/workspace/RobotController/build Browse Build...

Search: Grouped Advanced + Add Entry ✖ Remove Entry

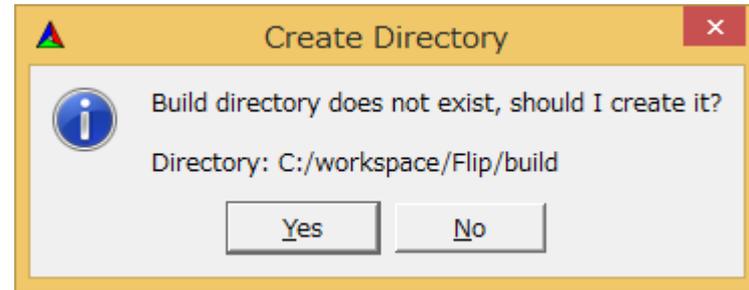
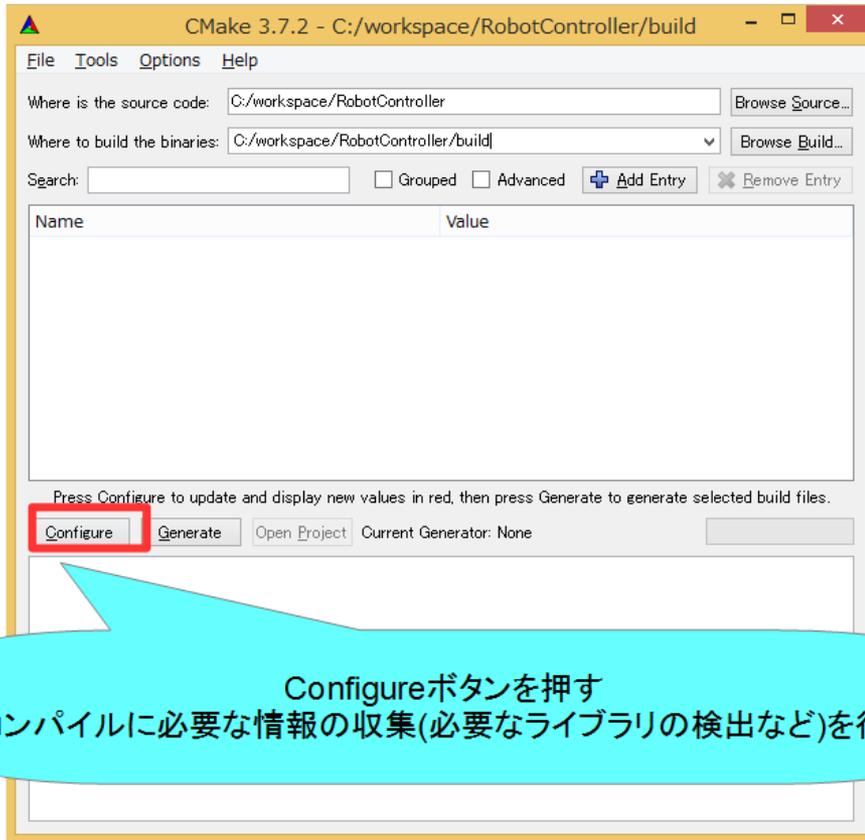
Name Value

ソリューションファイル等を生成するフォルダを指定
 (例: C:/workspace/RobotController/build)
 ドラックアンドドロップした場合は後ろに「/build」を追加
 何かトラブルが発生した時はこのフォルダを削除

Press Configure to update and display new values in red, then press Generate to generate selected build files.

Configure Generate Open Project Current Generator: None

ビルドに必要なファイルの生成



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

CMake 3.14以降の場合

ビルド環境の設定

Visual Studio 2019 → Visual Studio 16 2019

Visual Studio 2013 → Visual Studio 12 2013

※貸し出したPCでは「Visual Studio 14 2015」を指定

Specify the generator for this project

Visual Studio 15 2017

Optional platform for generator(if empty, generator uses: Win32)

x64

Optional toolset to use (argument to -T)

Specify native compilers
 Specify toolchain file for cross-compiling
 Specify options for cross-compiling

Finish Cancel

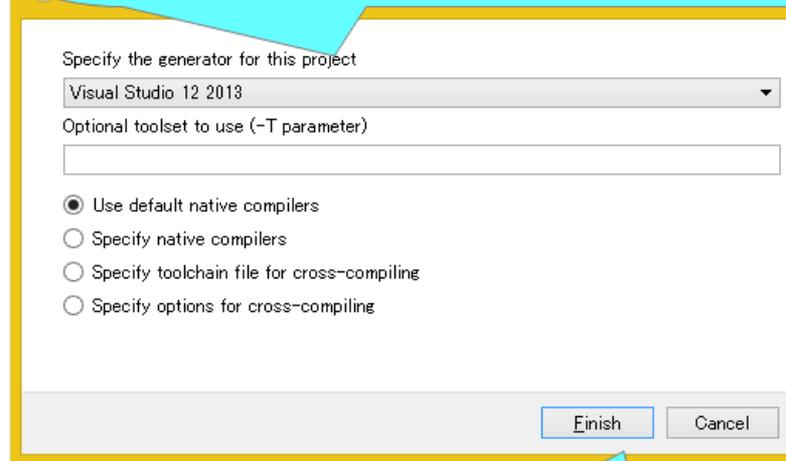
X64を選択する。

設定後、Finishボタンを押す

CMake 3.13以前の場合

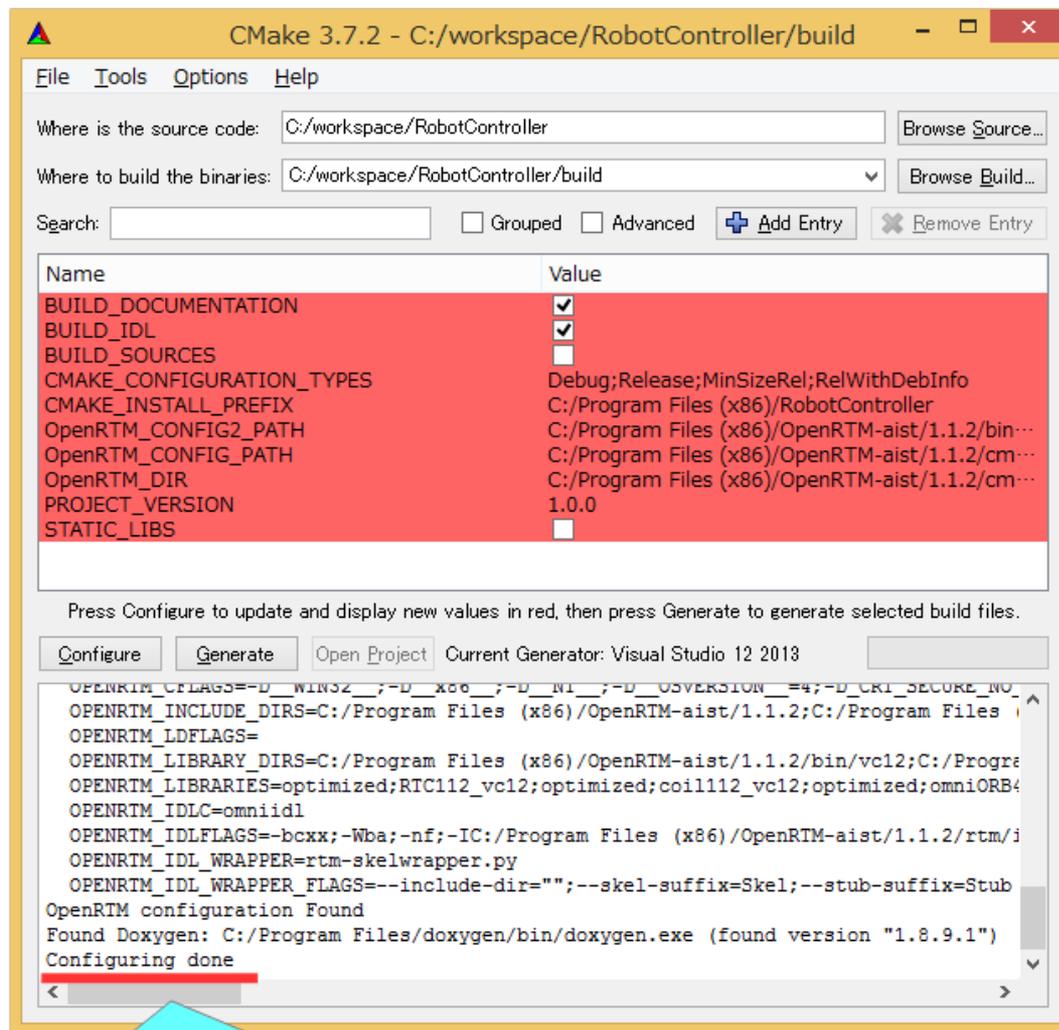
ビルド環境の設定

Visual Studio 2013 32bit → Visual Studio 12 2013
 Visual Studio 2013 64bit → Visual Studio 12 2013 Win64
 Visual Studio 2017 32bit → Visual Studio 15 2017
 Visual Studio 2017 64bit → Visual Studio 15 2017 Win64
 Code::Blocks → CodeBlocks-Uinx Makefiles
 ※32bitか64bitかはインストールした
 OpenRTM-aistが32bitか64bitかで選択



設定後、Finishボタンを押す

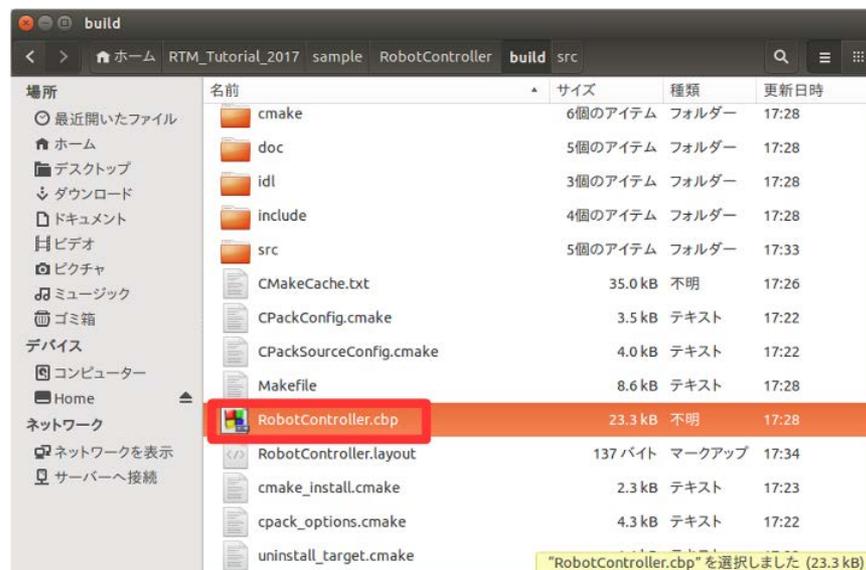
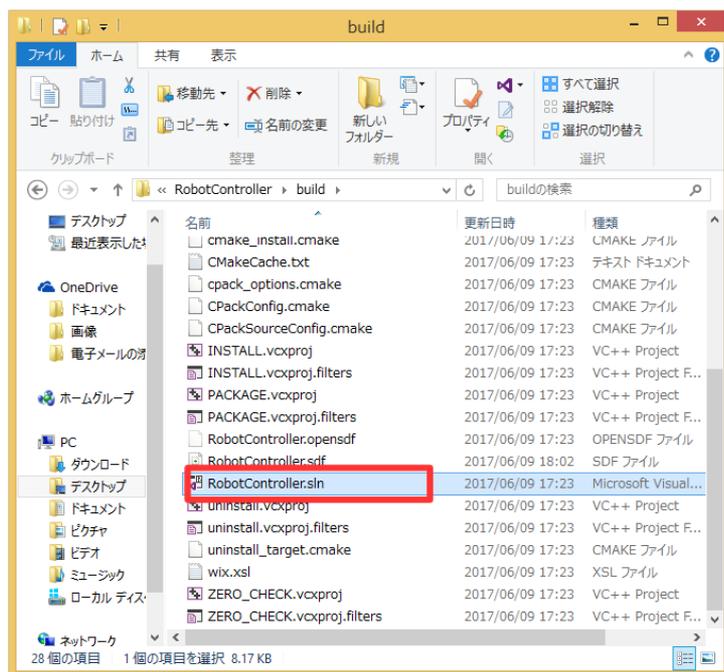
ビルドに必要なファイルの生成



「Configure done」が表示されていれば成功

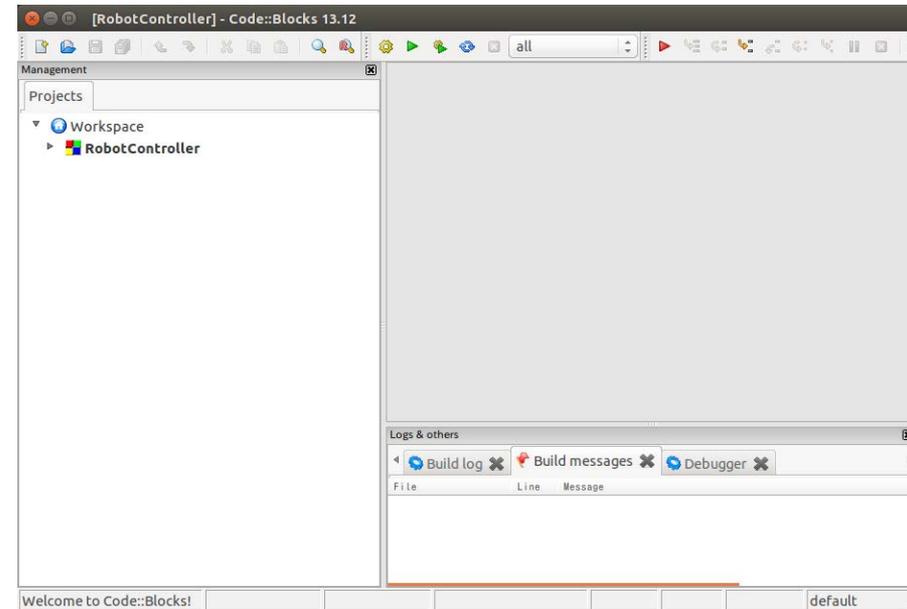
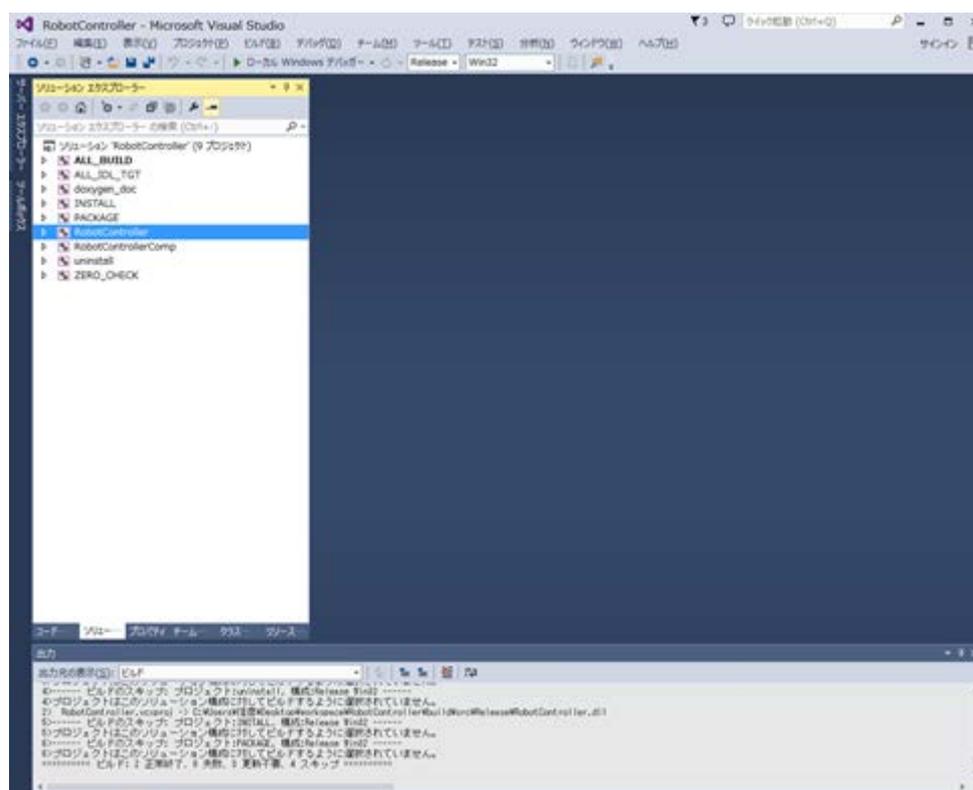
ソースコードの編集

- 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

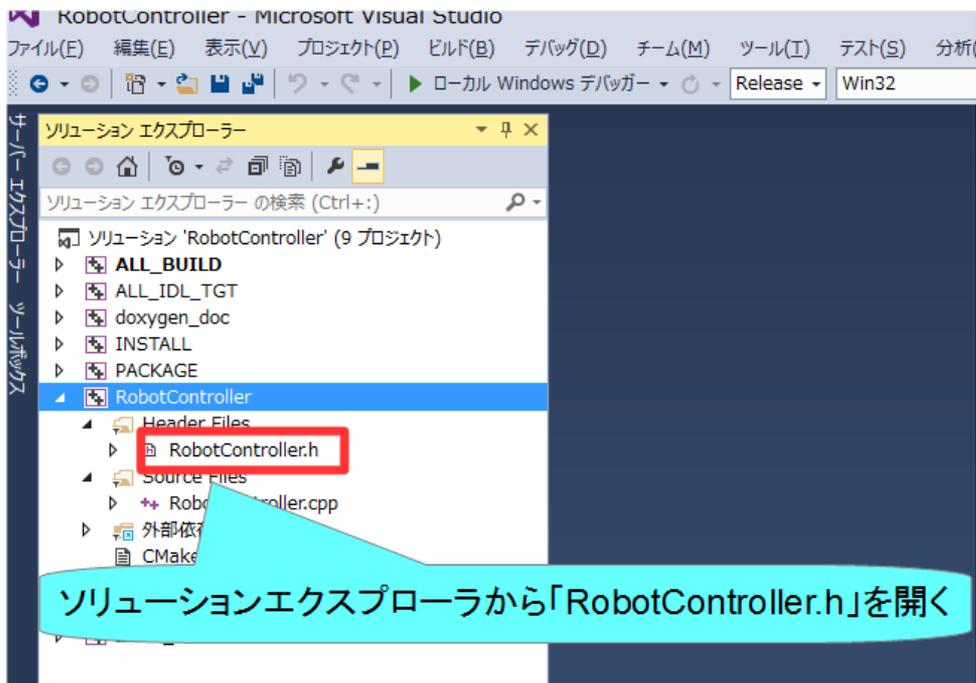
- Windows
 - Visual Studioが起動
- Ubuntu
 - Code::Blocksが起動



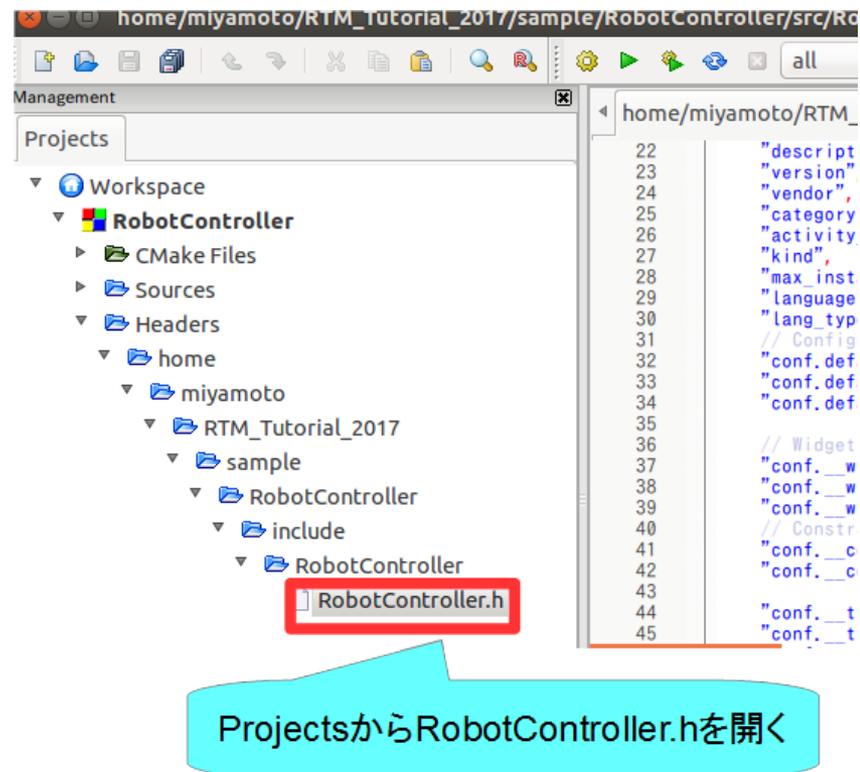
ソースコードの編集

- RobotController.hの編集

Visual Studio



Code::Blocks



ソースコードの編集

- RobotController.hの編集

センサ値を一時格納する変数の宣言
int sensor_data[4];

```

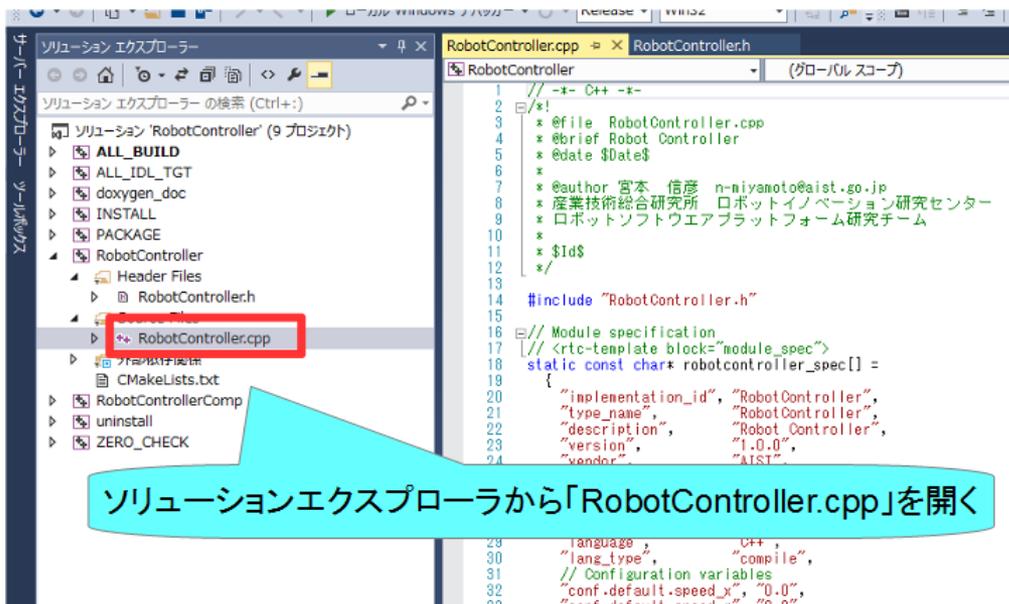
297
298 // <rtc-template block="private_attribute" >
299
300 // </rtc-template>
301
302 private:
303 int sensor_data[4]; //センサ値を一時格納する変数
304
305 // <rtc-template block="private_attribute" >
306
307 // </rtc-template>
308
309 // <rtc-template block="private_operation" >
310
311 // </rtc-template>
312
313 };
314
315
316 extern "C"
317 {

```

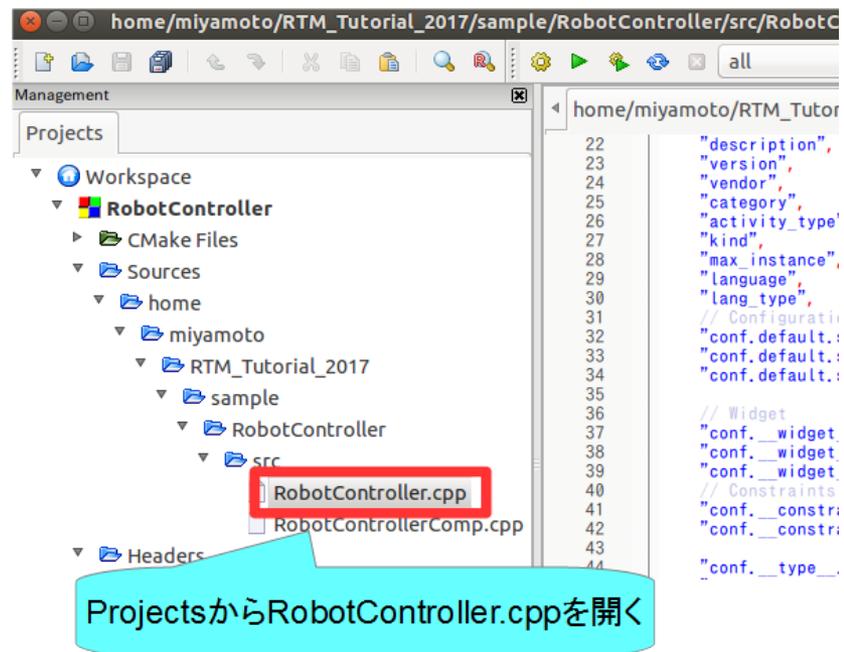
ソースコードの編集

- 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
141      //ロボットを停止する
142      m_out.data.vx = 0;
143      m_out.data.va = 0;
144      m_outOut.write();
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に格納
        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             //変数m_outにデータを格納する
185             //TimedVelocity2D型のため、vxに直進速度、
186             //vaに回転速度を格納する。
187             m_out.data.vx = m_speed_x;
188             m_out.data.va = m_speed_r;
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();

```

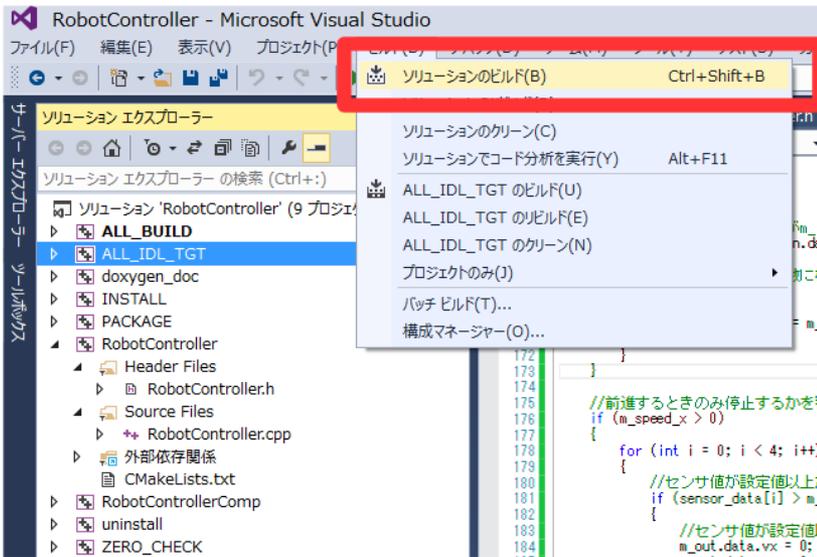
変数m_outにデータを格納する
TimedVelocity2D型のため、vxに直進速度、
vaに回転速度を格納する。

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

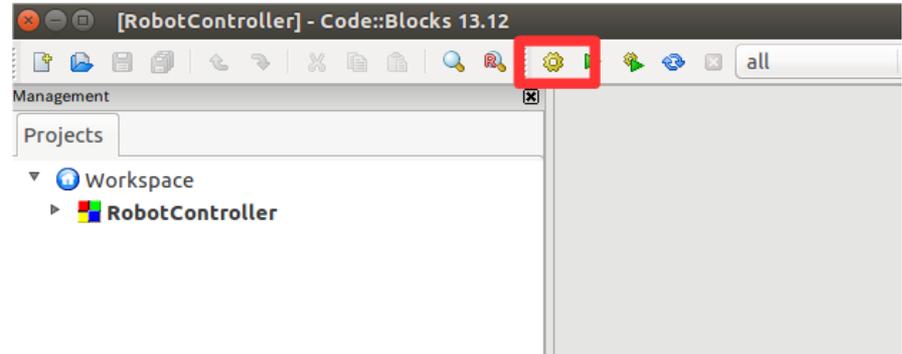
write関数でデータの書き込み

ソースコードのコンパイル

Visual Studio



Code::Blocks

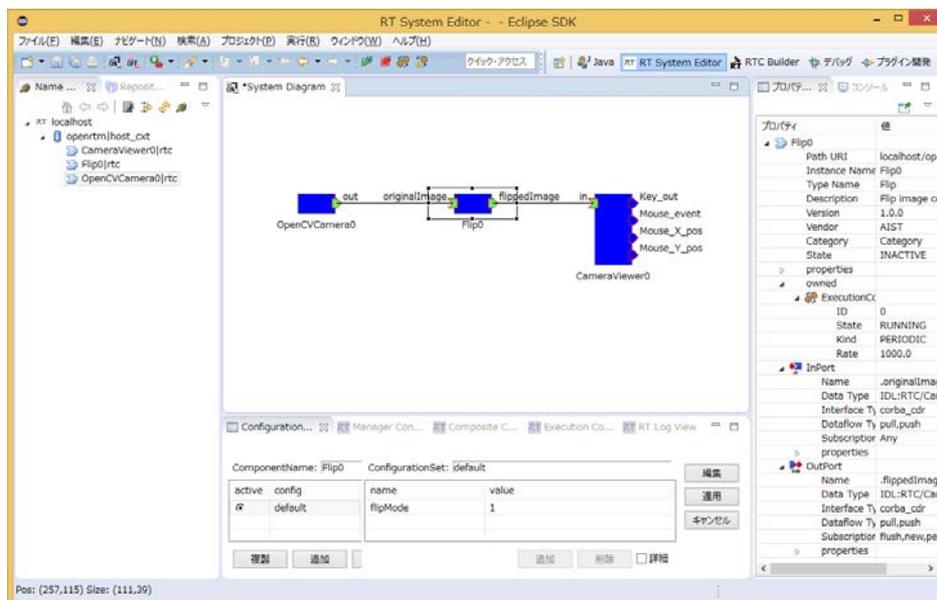


- 成功した場合、実行ファイルが生成される
 - Windows
 - **build¥src**フォルダの**Release**(もしくは**Debug**)フォルダ内に RobotControllerComp.exeが生成される
 - Ubuntu
 - **build/src**フォルダにRobotControllerCompが生成される

システム構築支援ツール RT System Editorについて

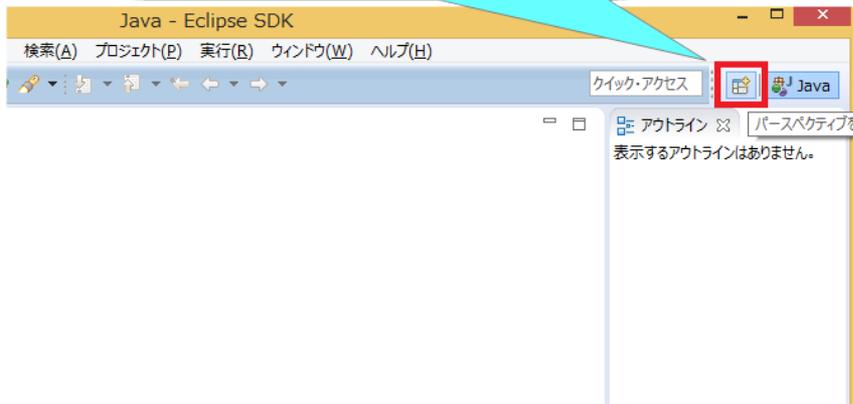
RT System Editor

- 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

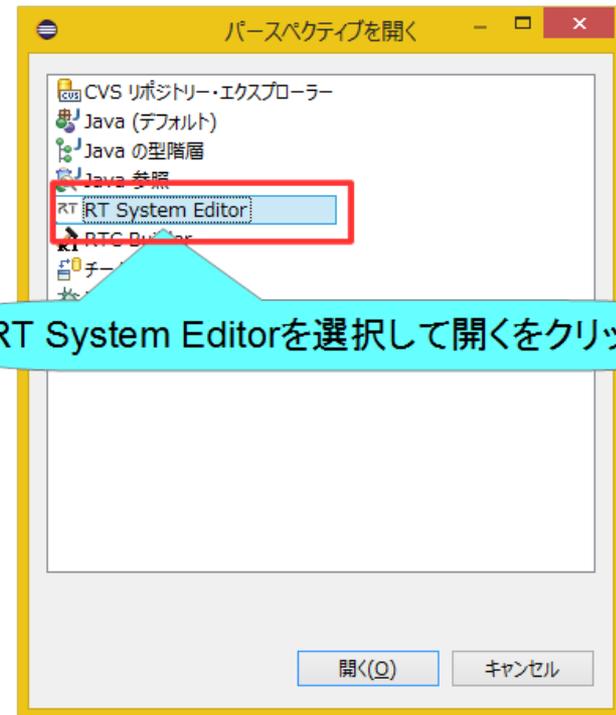


RT System Editorの起動

右上の「パースペクティブを開く」ボタンをクリック



RT System Editorを選択して開くをクリック

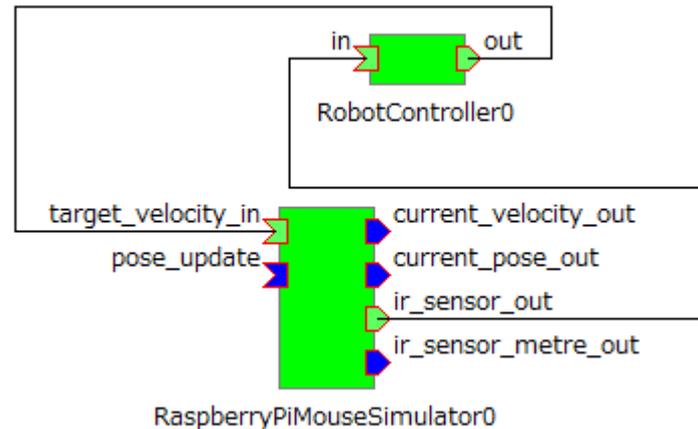


RT System Editorの画面構成



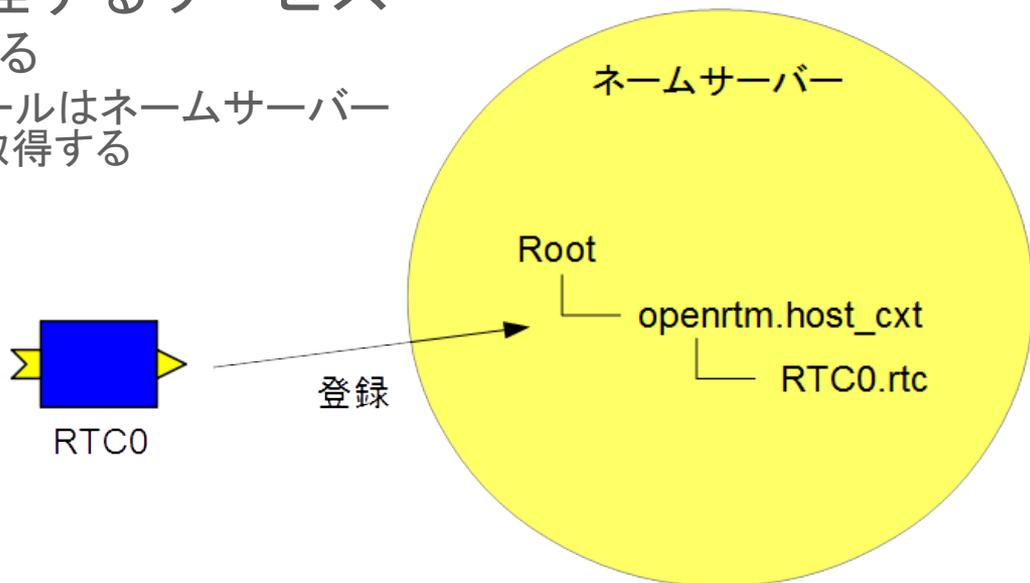
RobotControllerコンポーネントの動作確認

- 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
 - 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(インストールしていない場合)
 - » \$ sh install_raspimouse_simulator_usb.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
 - 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

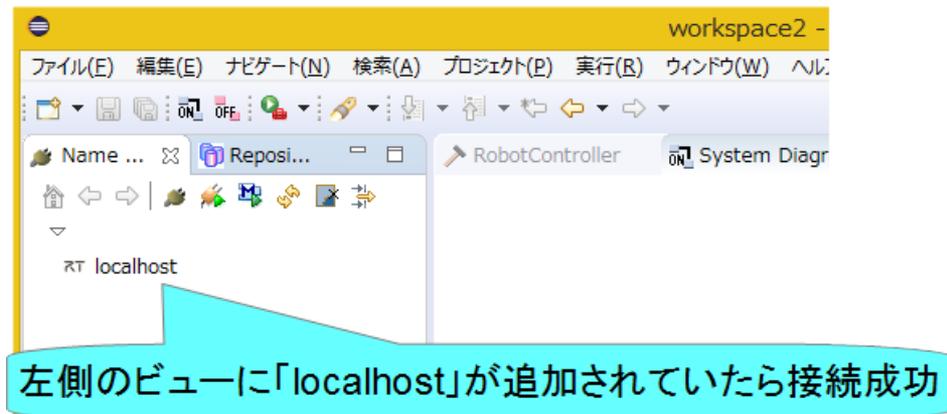
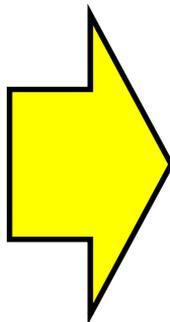
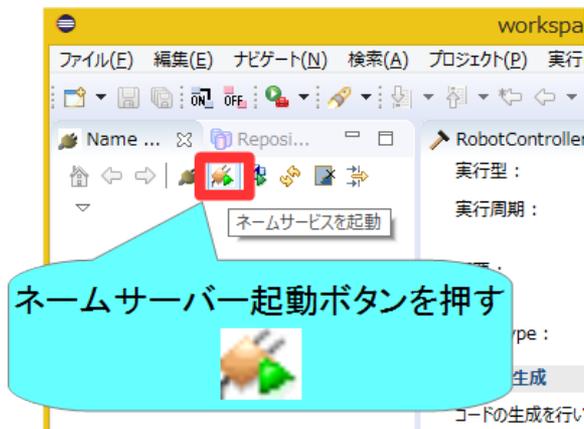


ネームサーバーの起動

- 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管理でRTCの参照を取得する

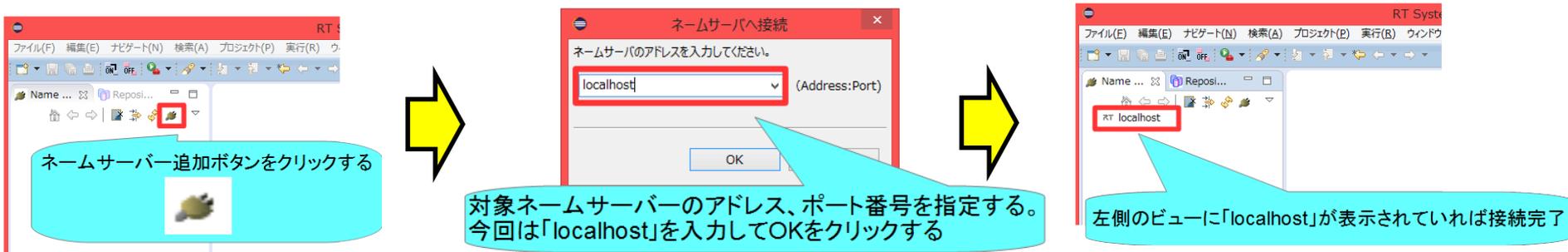


起動する手順



ネームサーバーの起動

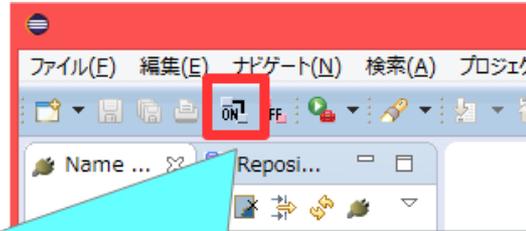
- OpenRTM-aist 1.1.2以前の手順
 - Windows 7
 - 「スタート」→「すべてのプログラム」→「OpenRTM-aist 1.2.0」→「Tools」→「Start Naming Service」
 - Windows 8.1
 - 「スタート」→「アプリビュー(右下矢印)」→「OpenRTM-aist 1.2.0」→「Start Naming Service」
 - Windows 10
 - 左下の「ここに入力して検索」にStart Naming Serviceと入力して起動
 - Ubuntu
 - \$ rtm-naming



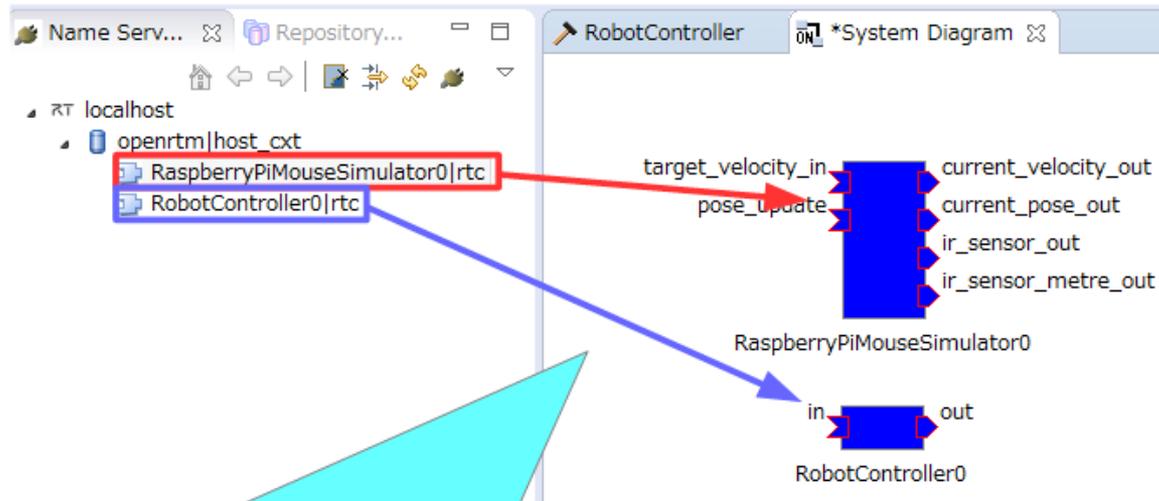
RobotControllerコンポーネントの動作確認

- 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
 - 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(インストールしていない場合)
 - » \$sudo sh install_raspimouse_simulator_usb.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



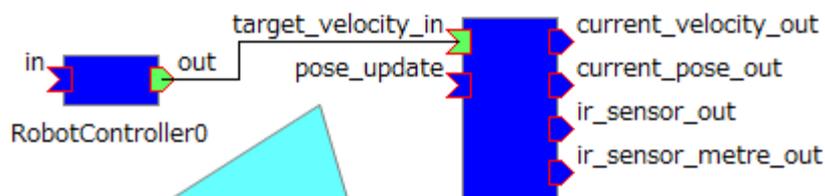
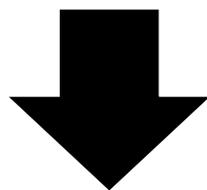
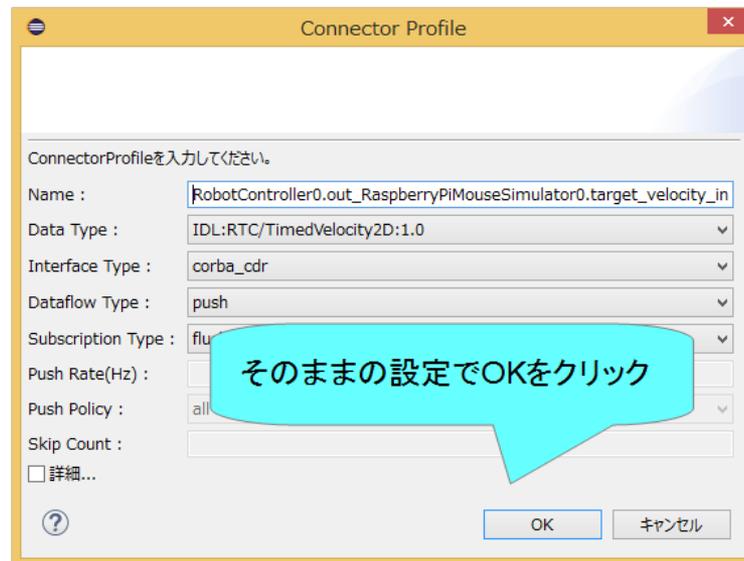
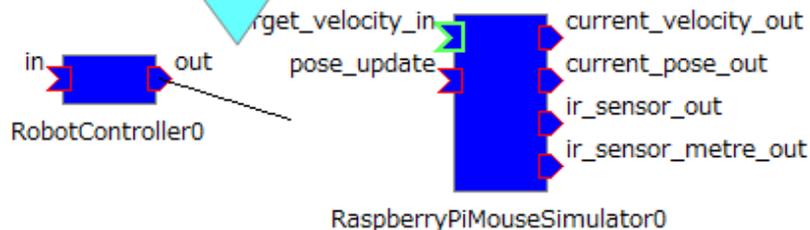
このボタンを押すことでSystem Diagramを表示する



左側のネームサービスビューから
RaspberryPiMouseSimulator0.rtc、RobotController0.rtcを
System Diagramにドラッグアンドドロップ

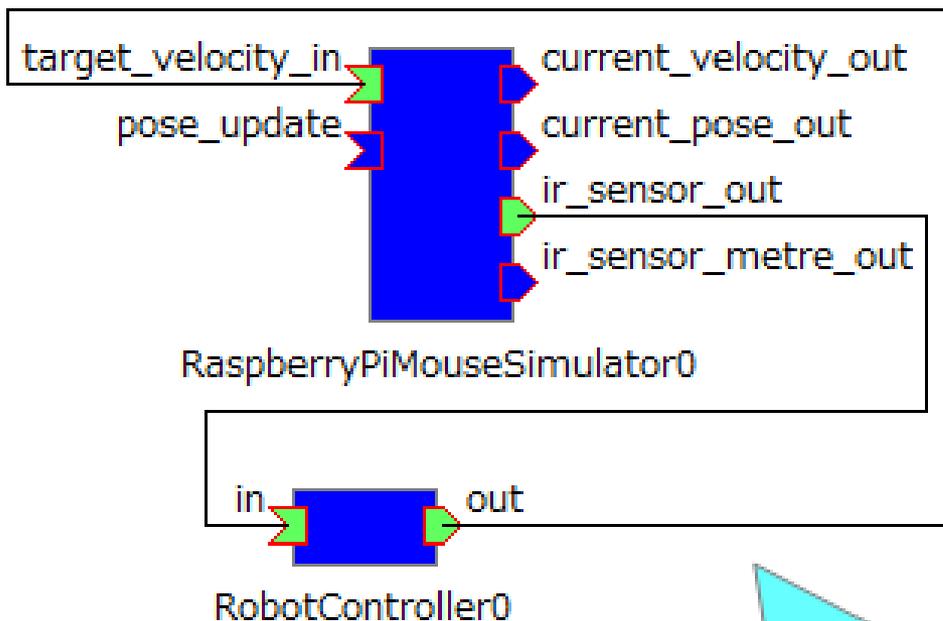
データポートの接続

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



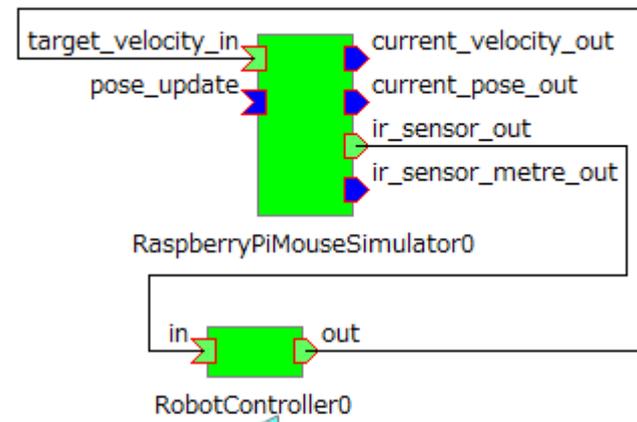
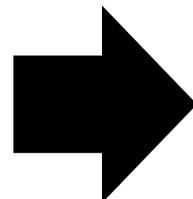
1. ポート間に線が表示される
2. InPort、OutPortが緑色で表示される

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

1. RobotController0をクリックする

active	config	name	value
<input checked="" type="radio"/>	default	speed_r	0.0
		speed_x	0.0
		stop_d	30

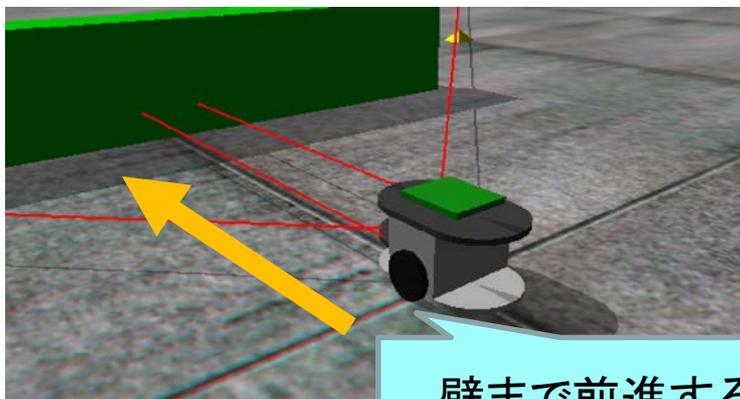
2. 編集ボタンを押す

3. スライダーを操作する

- 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

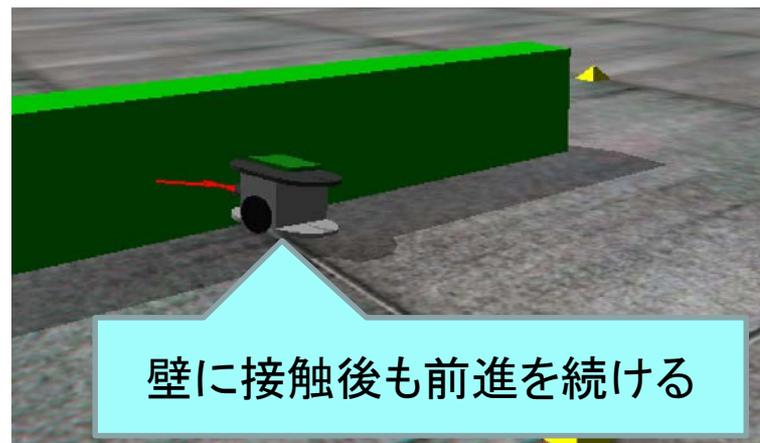
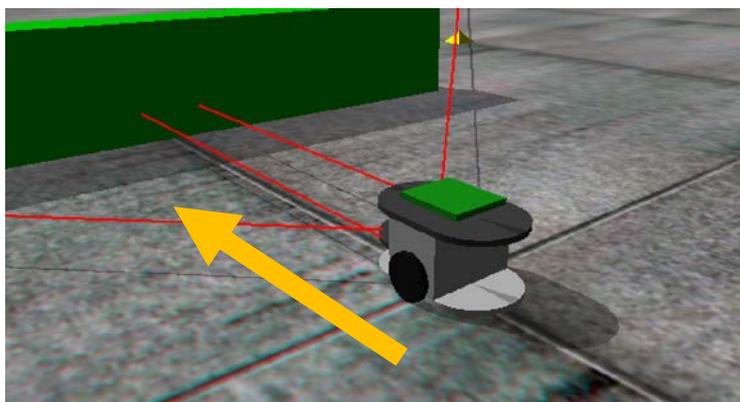
- タッチセンサがオンになった時に停止した場合



壁まで前進する

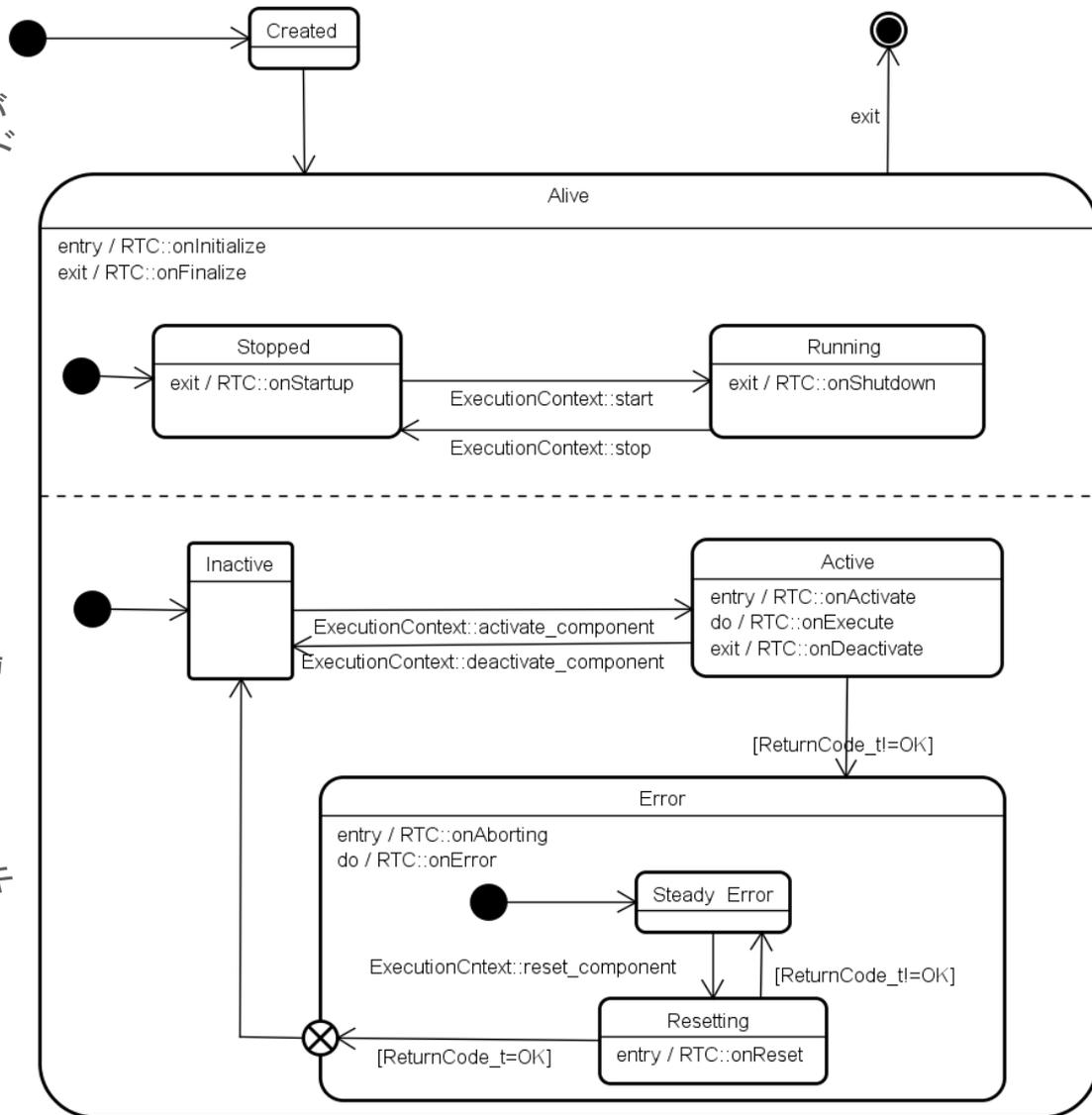


- タッチセンサがオンになっても停止しない場合



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(Running)状態になる
 - 自動的にInactive状態に遷移する
 - Inactive
 - 非活性状態
 - activate_componentメソッドを呼び出すと活性状態に遷移する
 - RT System Editor上での表示は青
 - Active
 - 活性状態
 - onExecuteコールバックが実行コンテキストにより実行される
 - リターンコードがRTC_OK以外の場合はエラー状態に遷移する
 - RT System Editor上での表示は緑
 - Error
 - エラー状態
 - onErrorコールバックが実行コンテキストにより実行される
 - reset_componentメソッドを呼び出すと非活性状態に遷移する
 - RT System Editor上での表示は赤
 - 終了状態

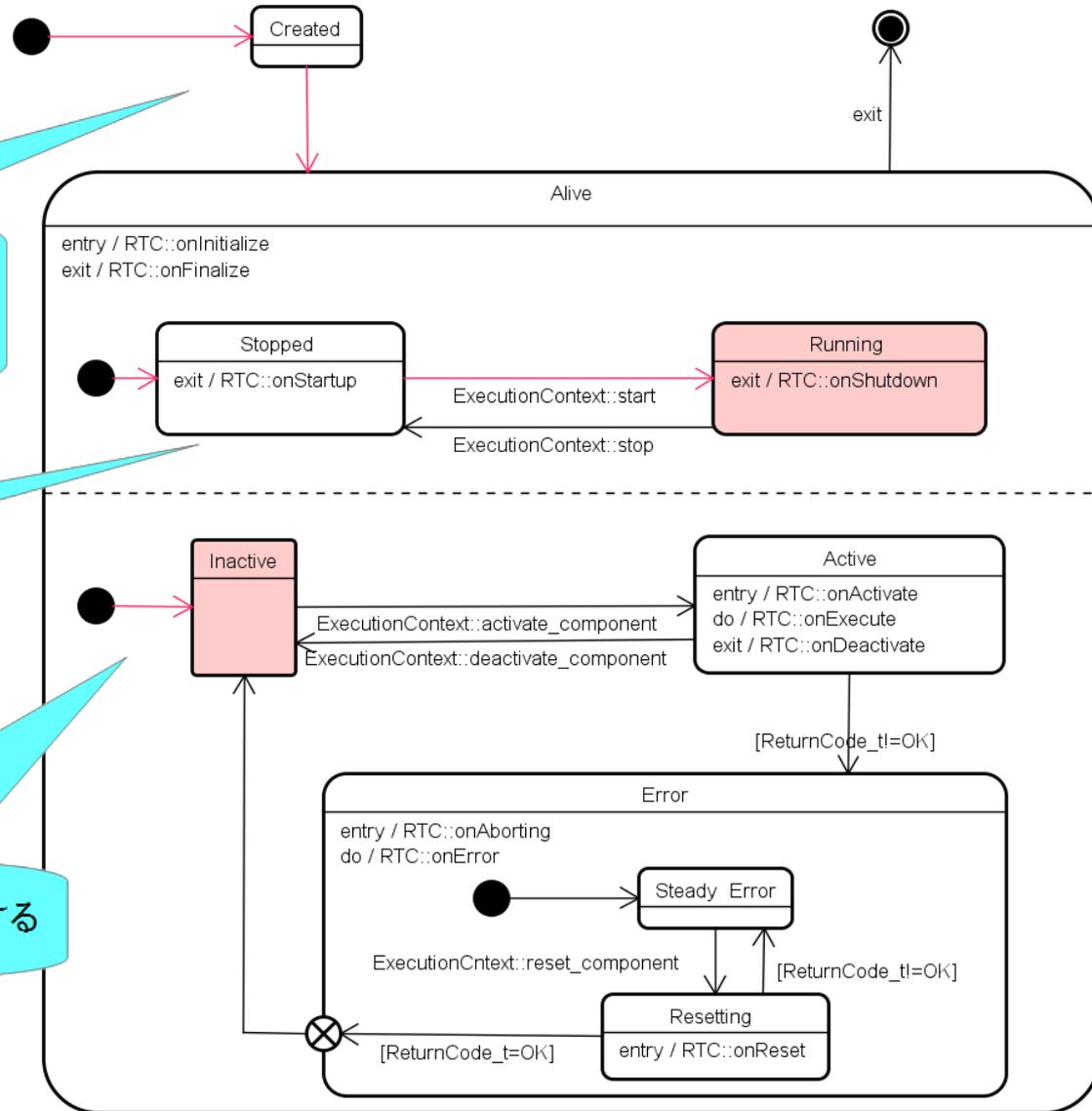


RTコンポーネントの状態遷移(生成直後)

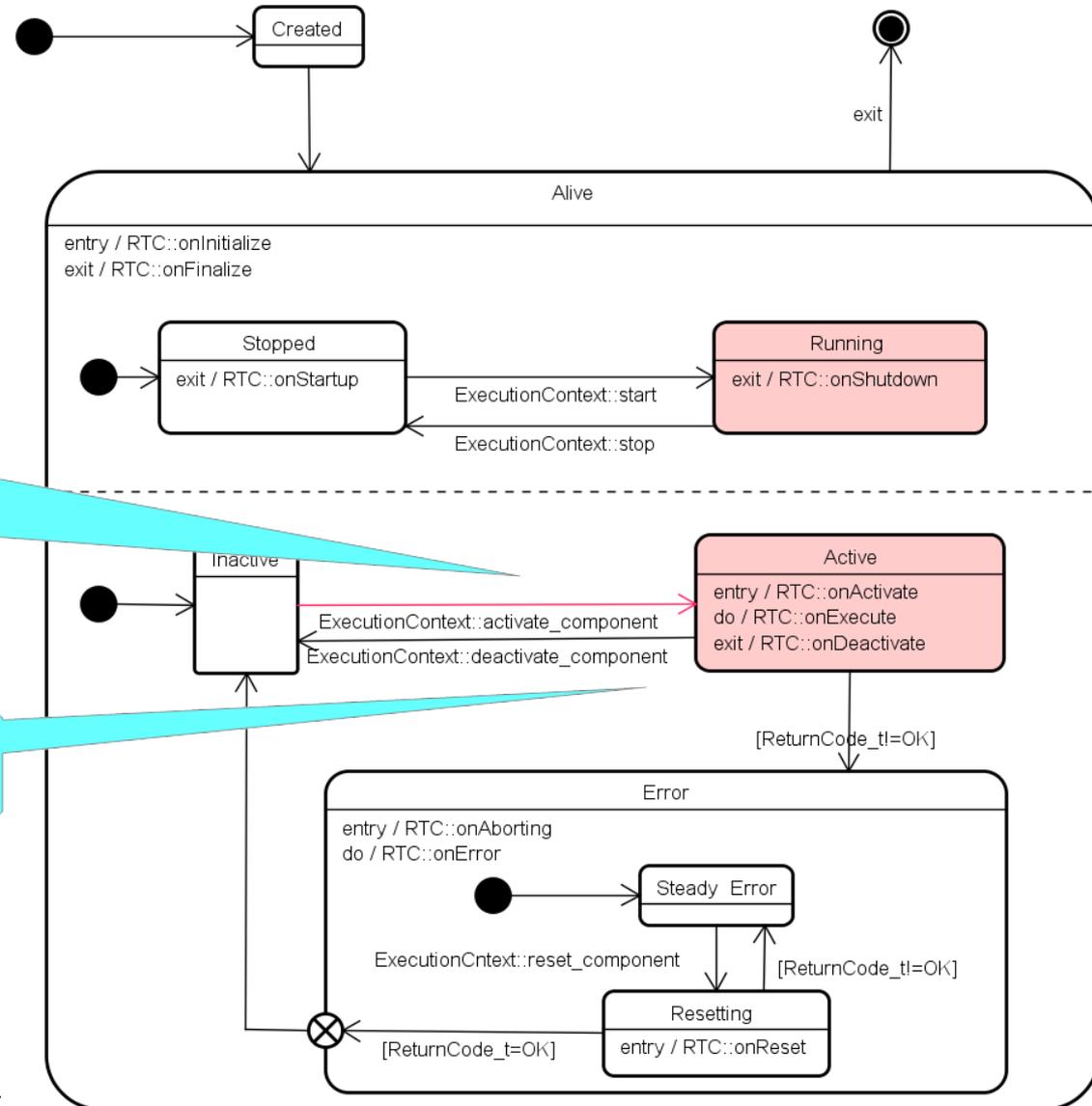
最初にCreated状態に遷移して
実行コンテキストの生成等を行う
この時にonInitializeコールバックを実行する

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

Created状態の次にInactive状態に遷移する



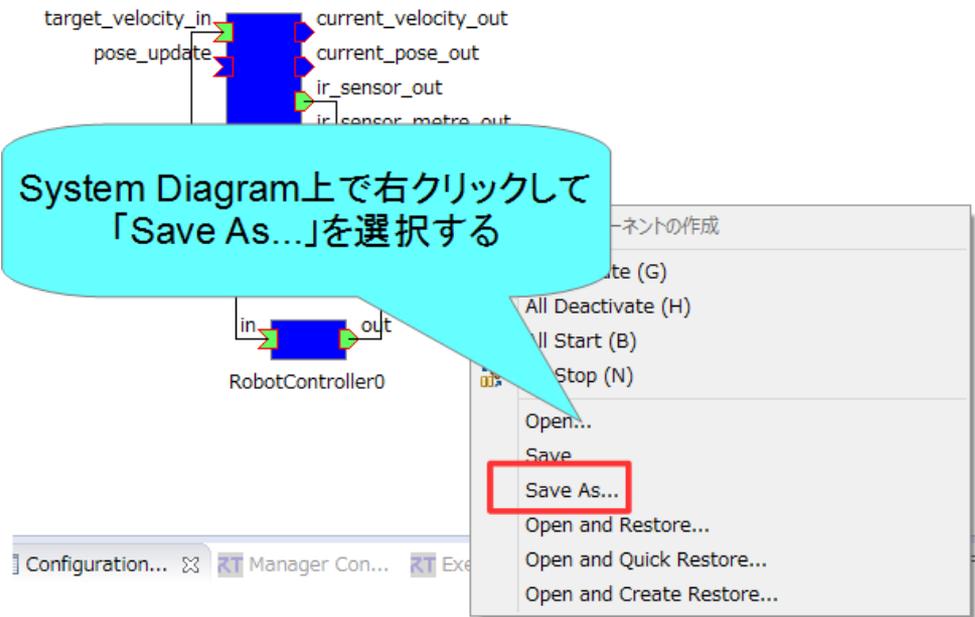
RTコンポーネントの状態遷移(アクティブ化)



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

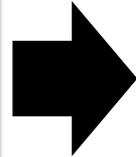
周期実行の実行コンテキストの場合、onExecuteコールバックが周期的に呼び出される。

システムの保存



System Diagram上で右クリックして「Save As...」を選択する

ベンダ名、システム名、バージョン、保存ファイル名を入力



Profile Information

Vendor: AIST

System Name: test

Version: 0

Path: C:\work\test.xml | 参照...

Update Log:

RobotController0
 RaspberryPiMouseSimulator0

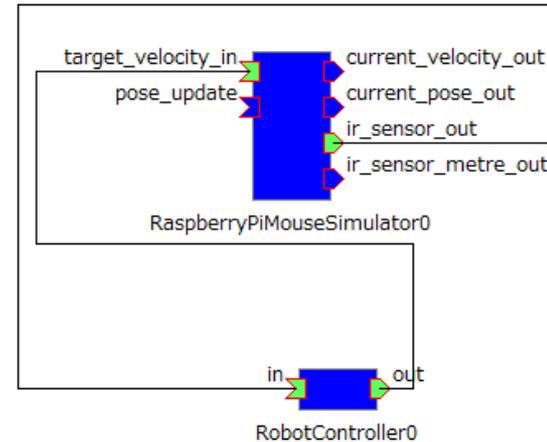
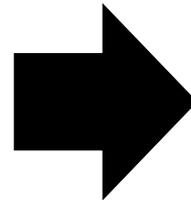
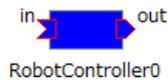
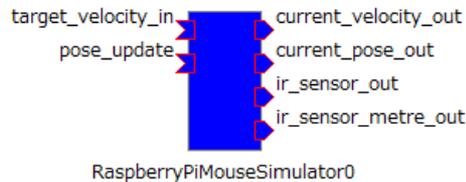
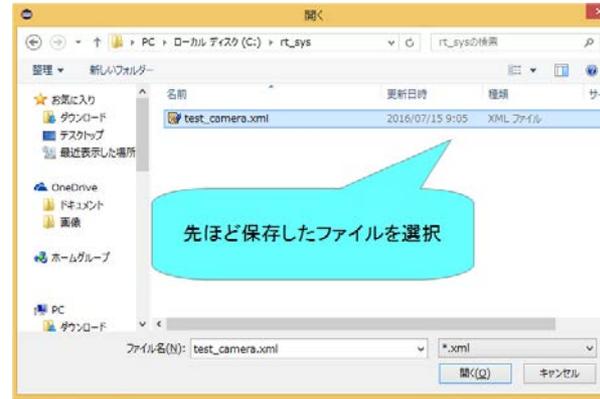
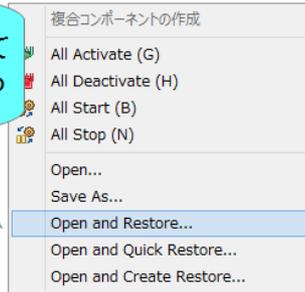
Required:

すべて選択(S) | すべて解除(D)

OK | キャンセル

システムの復元

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



- 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

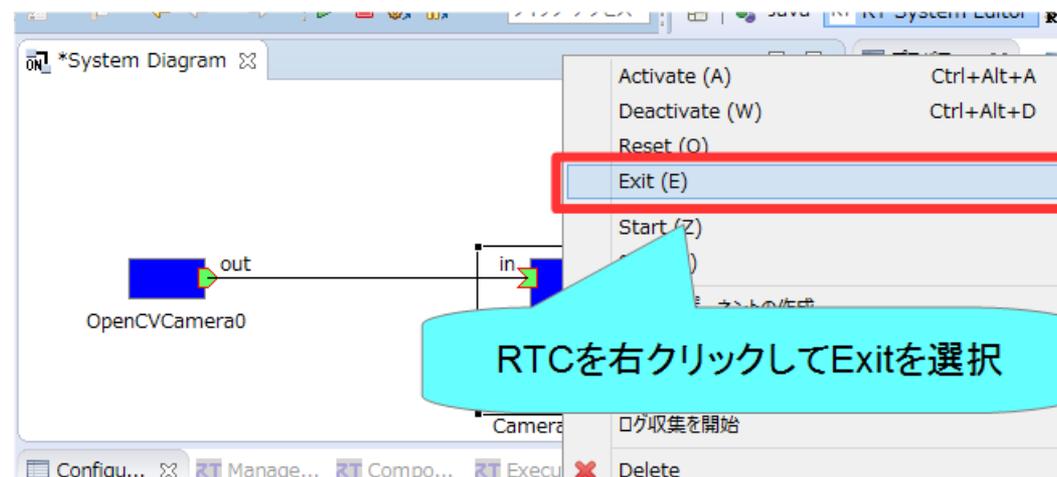
- 「Open and Create Restore」を選択した場合はマネージャからコンポーネント起動

非アクティブ化、終了

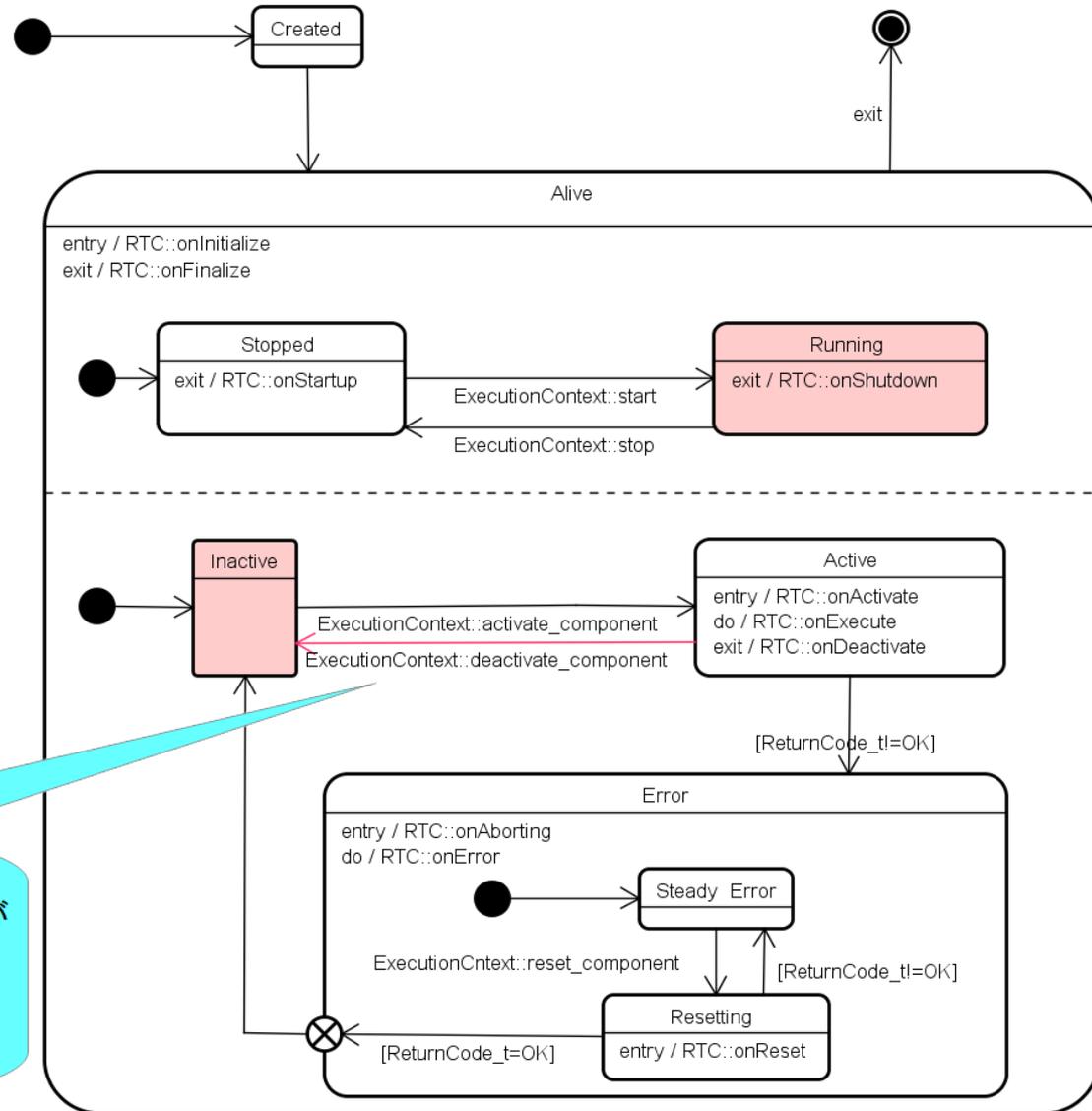
- 非アクティブ化



- 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

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

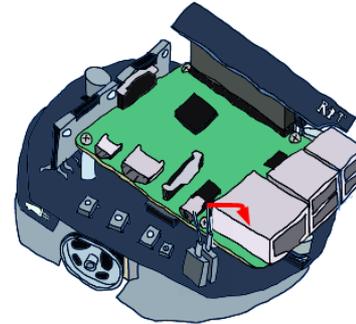


- 注意事項

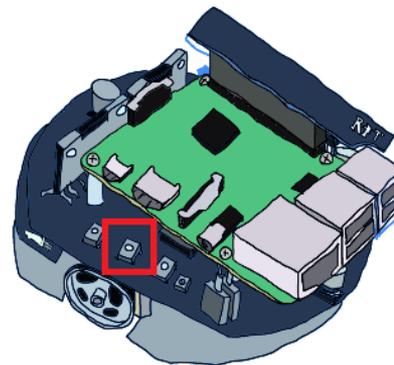
- ノートPCに複数のネットワークインターフェースが存在する場合にRTCの通信ができなくなる可能性があります。
 - 問題が発生した場合は個別に対応します。
- Raspberry Piアクセスポイント接続後はインターネットに接続できなくなります。
- Raspberry Piアクセスポイント接続後に、**起動済みのネームサーバーとRTCは再起動してください。**
- Raspberry Piはシャットダウンしてから電源スイッチをオフにするようにしてください
- モーター電源スイッチはこまめに切るようにしてください

Raspberry Piとの接続

- Raspberry Piの電源投入
 - 内側のスイッチをオンにする



- 電源を切る場合
 - 3つ並んだスイッチの中央のボタンを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

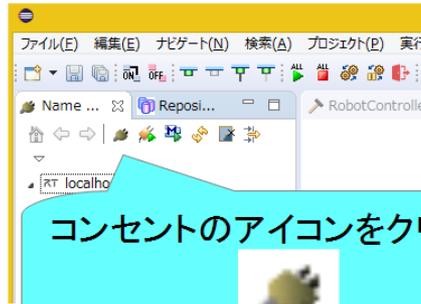
- 無線LANアクセスポイントとの接続
 - SSID、パスワードはRaspberry Piマウス上のシールに記載
 - 接続手順(Windows)
 - 画面右下のネットワークアイコンをクリック



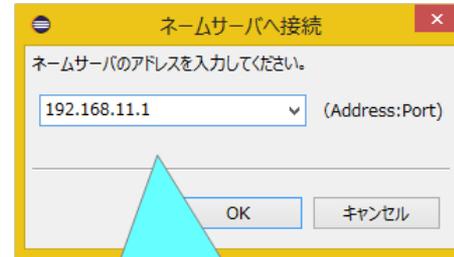
- raspberrypi_xxに接続後、パスワードを入力



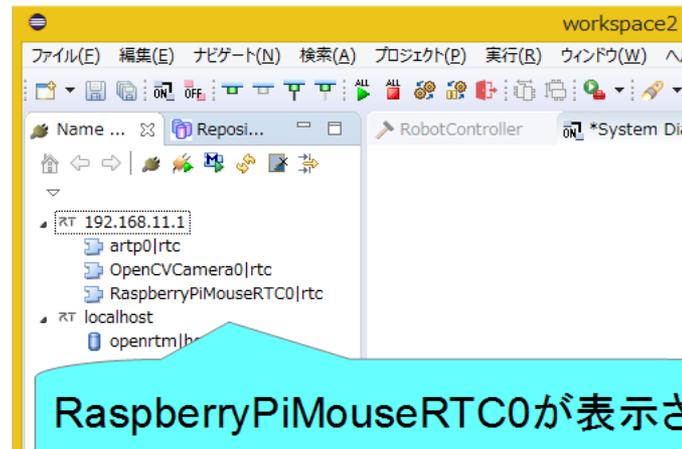
ネームサーバーとの接続



コンセントのアイコンをクリック



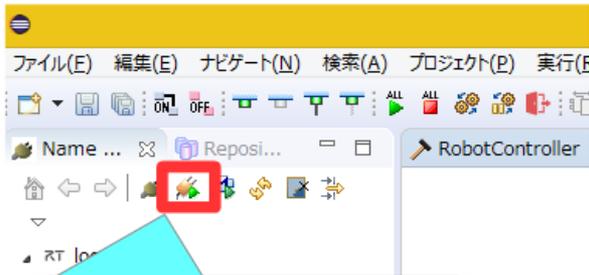
192.168.11.1を入力



RaspberryPiMouseRTC0が表示される

起動済みのRTC、ネームサーバー再起動

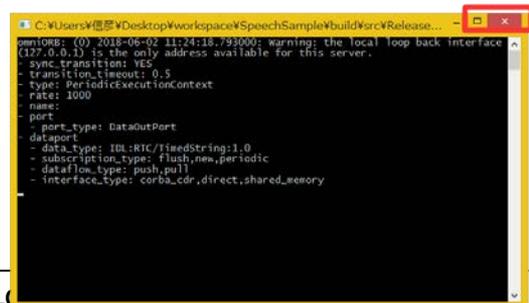
- ネームサーバーを再起動する
 - OpenRTM-aist 1.2の場合はネームサーバー起動ボタンで再起動



ネームサーバー起動ボタンを押すと
ネームサーバー再起動



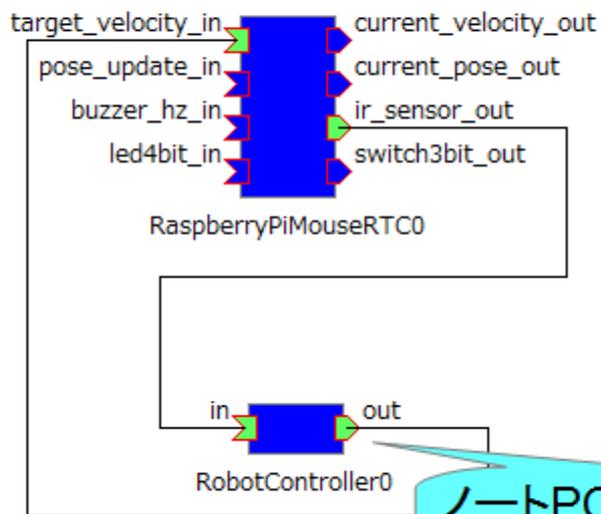
- OpenRTM-aist 1.1.2の場合はネームサーバーのプロセス終了後、「Start Naming Service」を再度実行
- RTC再起動
 - RTCをexitするか、RTC起動時に表示したウィンドウの×ボタンを押して終了する
 - 実行ファイル(RobotControllerComp.exe)を再度実行



ポートの接続

- RobotController0とRaspberryPiMouseRTC0を接続する

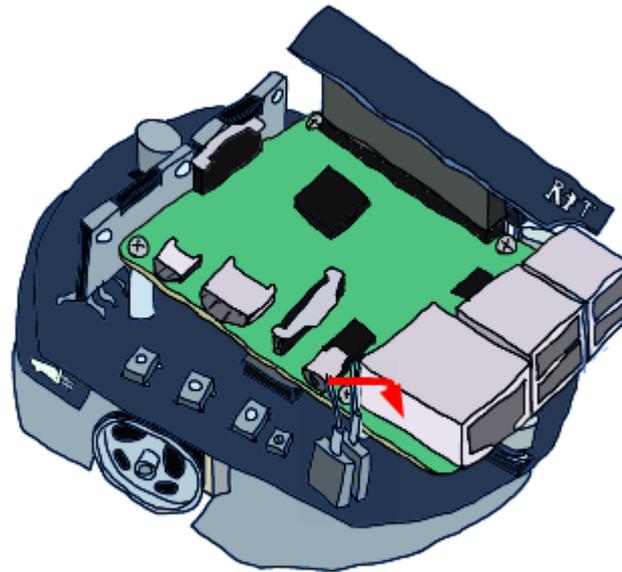
Raspberry Piで起動したRTC
(RaspberryPiMouseRTC)



ノートPCで起動したRTC
(RobotController)

動作確認

- モーターの電源投入
 - 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

リセット

- RTCがエラー状態に遷移した場合にエディタ上には赤く表示される。



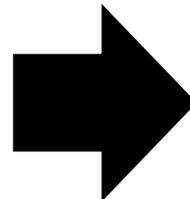
```
RTC::ReturnCode_t Test::onActivated(RTC::UniqueId ec_id)↓
{↓
HANDLE hCom = INVALID_HANDLE_VALUE;↓
hCom = CreateFile("COM5", GENERIC_READ | GENERIC_WRITE, 0, NULL, OPEN_EXISTING, 0, NULL);↓
if (hCom == INVALID_HANDLE_VALUE)↓
{↓
return RTC::RTC_ERROR;↓
}↓
}
```

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

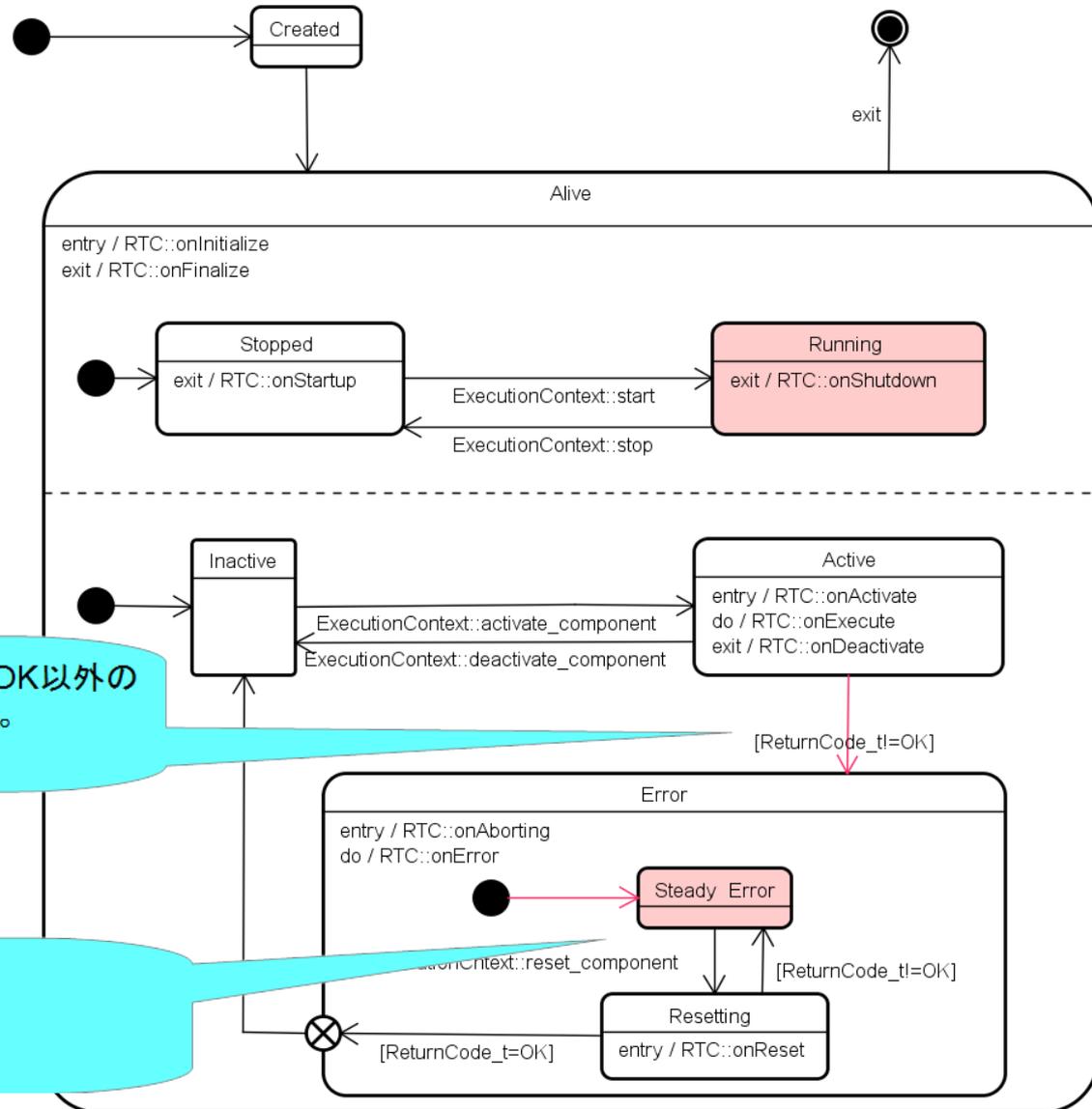
- 以下の操作で非アクティブ状態に戻す



RTCを右クリックしてResetを選択



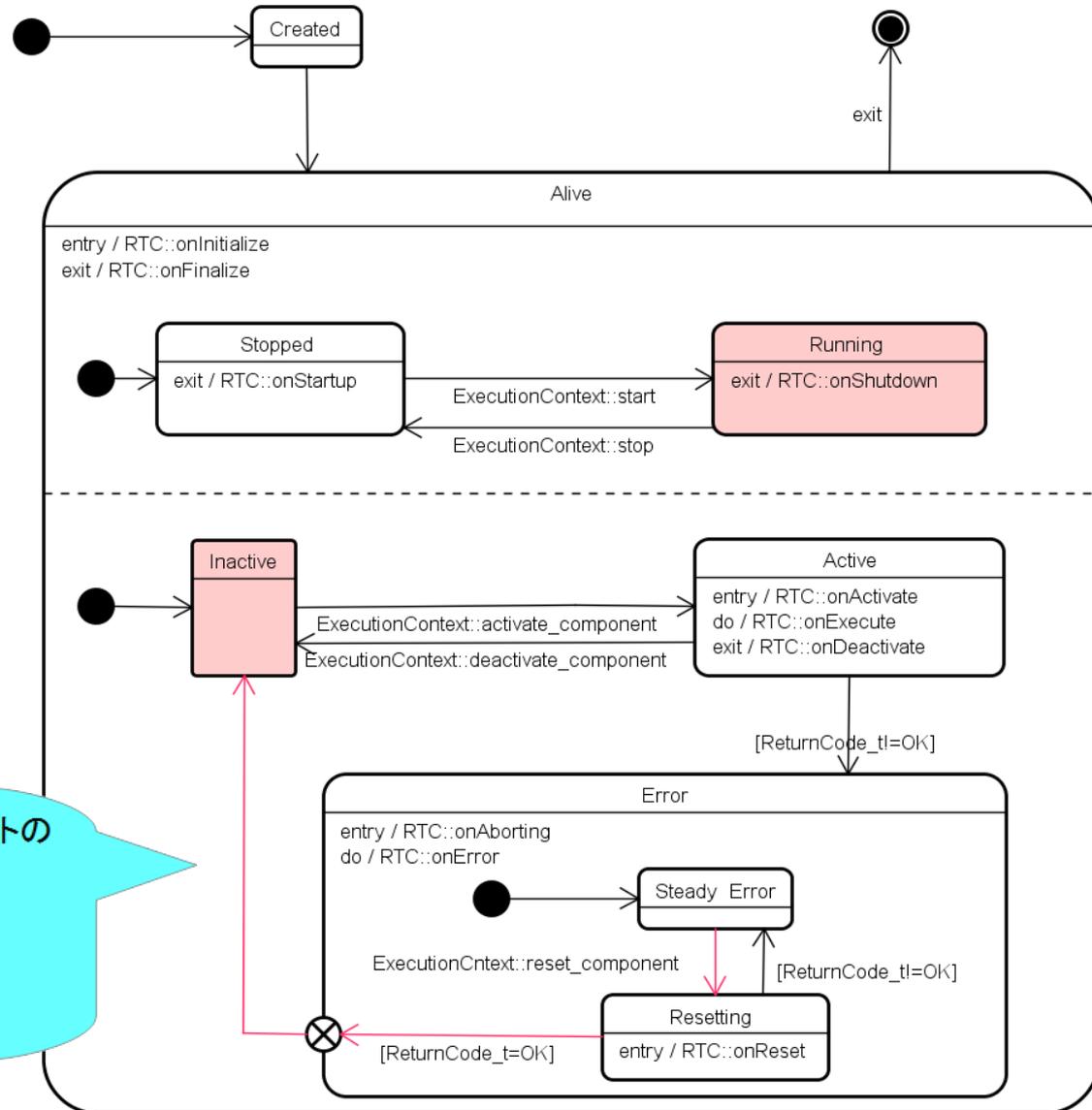
RTコンポーネントの状態遷移(エラー)



onActivated、onExecute、onDeactivatedがRTC_OK以外のリターンコードを返した場合、エラー状態に遷移する。この時、onAbortingコールバックが実行される。

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

RTコンポーネントの状態遷移(リセット)

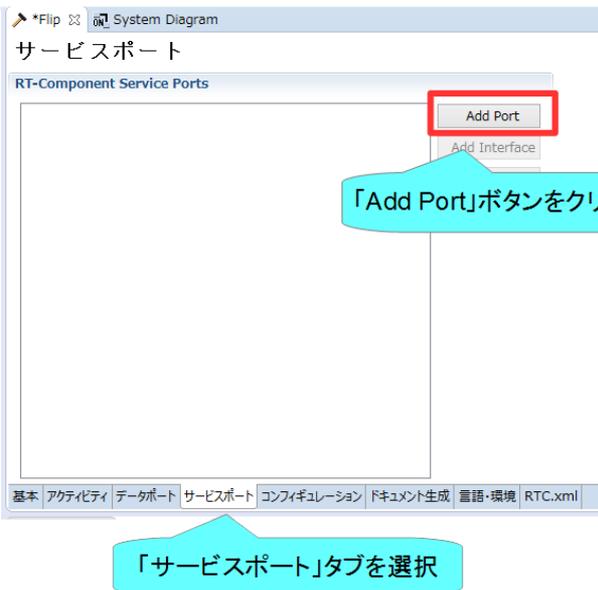


RTシステムエディタの操作によりRTコンポーネントのリセットを行うとreset_componentメソッドが呼び出される。
 reset_componentメソッドによりコンポーネントがInactive状態に遷移する。
 この時onResetコールバックが実行される

RTC Builder 補足

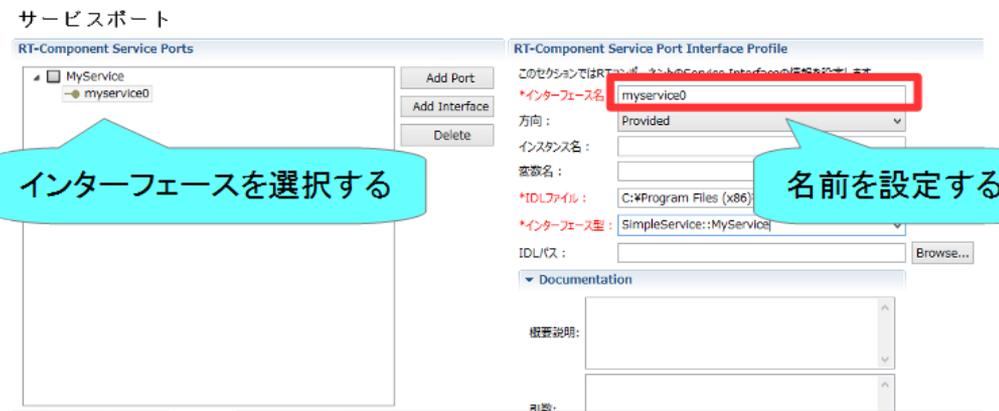
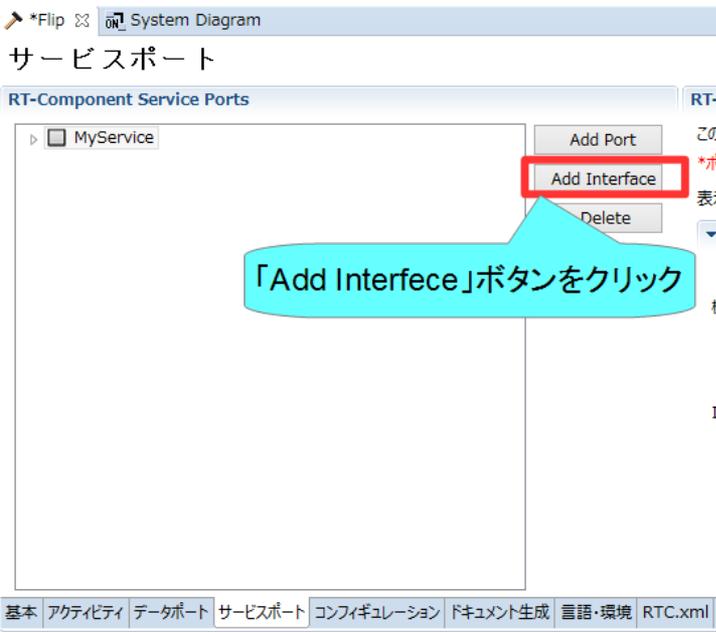
サービスポートの設定

- 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

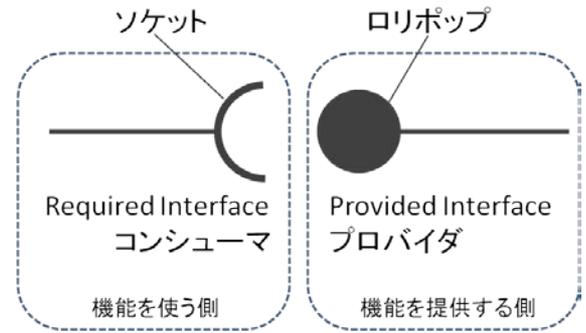
- インターフェースを追加する



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



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

*インターフェース名: myservice0

方向: Provided

インスタンス名:

変数名:

*IDLファイル: C:\Program Files (x86)\OpenRTM-aist\1.1.2\Com Browse...

*インターフェース型: SimpleService::MyService

IDLパス: Browse...

「Browse...」をクリックしてIDLファイルを選択

インターフェース型を選択

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

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



サービスポートの設定

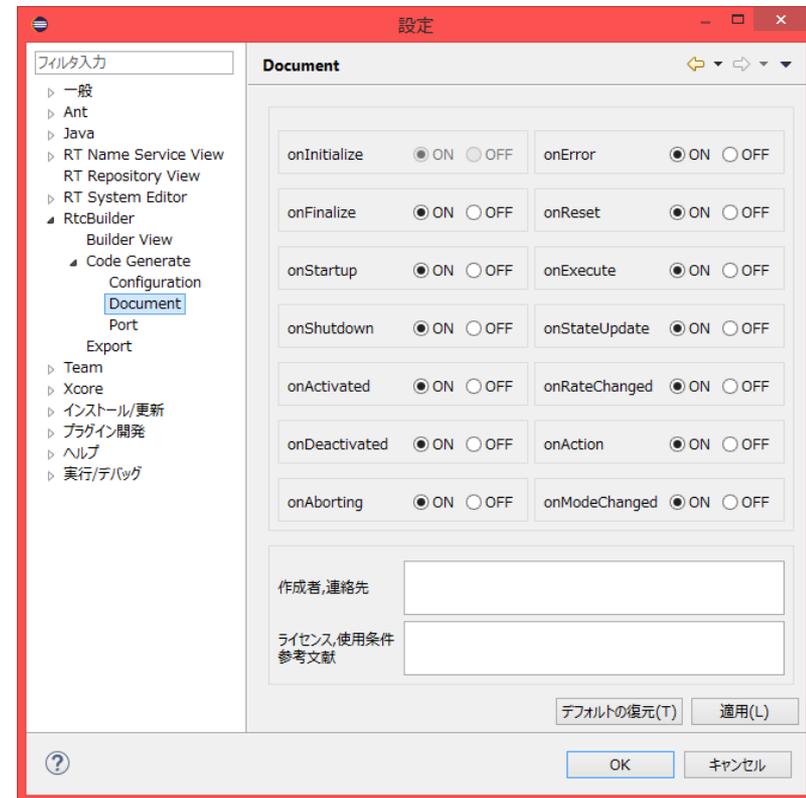
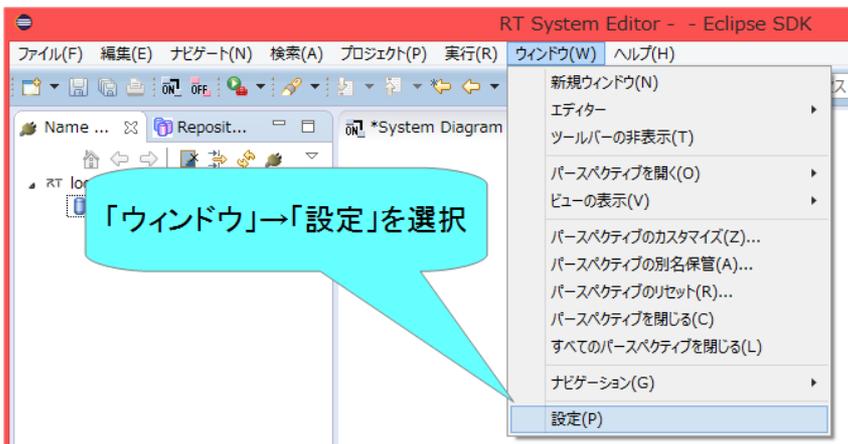
- IDLファイルについて
 - プログラミング言語に非依存のインターフェース定義言語

```

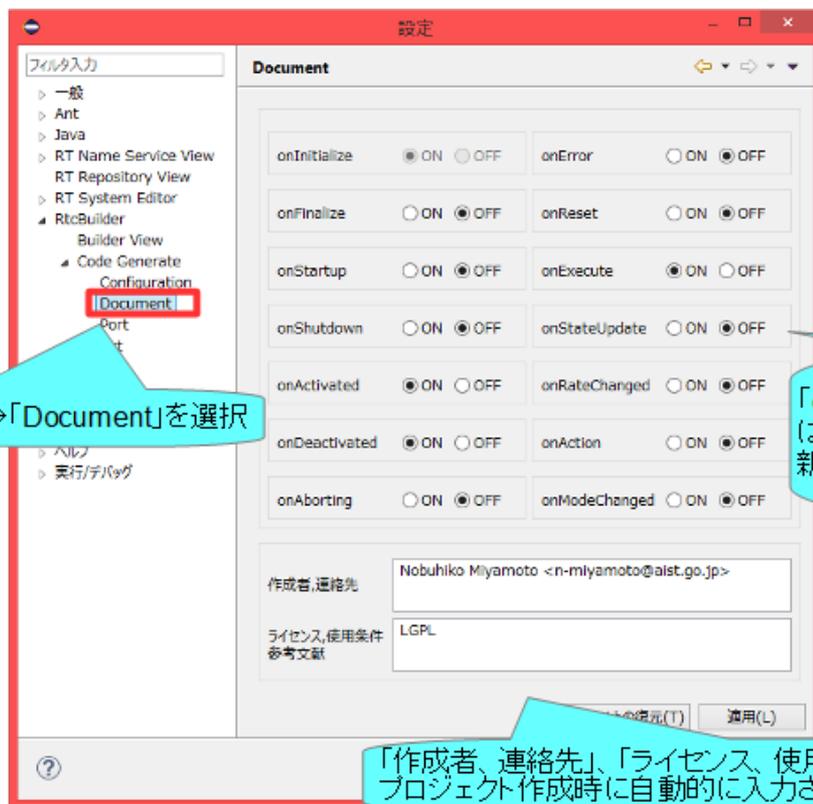
1 | module SimpleService {↓
2 |     typedef sequence<string> EchoList;↓
3 |     typedef sequence<float> ValueList;↓
4 |     interface MyService↓
5 |     {↓
6 |         string echo(in string msg);↓
7 |         EchoList get_echo_history();↓
8 |         void set_value(in float value);↓
9 |         float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

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

RTC Builderに関する設定



RTC Builderに関する設定



設定

フィルタ入力

- 一般
- Ant
- Java
- RT Name Service View
- RT Repository View
- RT System Editor
- RtcBuilder
 - Builder View
 - Code Generate
 - Configuration
 - Document**
 - Port

- ヘルプ
- 実行/デバッグ

Document

onInitialize	<input checked="" type="radio"/> ON <input type="radio"/> OFF	onError	<input type="radio"/> ON <input checked="" type="radio"/> OFF
onFinalize	<input type="radio"/> ON <input checked="" type="radio"/> OFF	onReset	<input type="radio"/> ON <input checked="" type="radio"/> OFF
onStartup	<input type="radio"/> ON <input checked="" type="radio"/> OFF	onExecute	<input checked="" type="radio"/> ON <input type="radio"/> OFF
onShutdown	<input type="radio"/> ON <input checked="" type="radio"/> OFF	onStateUpdate	<input type="radio"/> ON <input checked="" type="radio"/> OFF
onActivated	<input checked="" type="radio"/> ON <input type="radio"/> OFF	onRateChanged	<input type="radio"/> ON <input checked="" type="radio"/> OFF
onDeactivated	<input checked="" type="radio"/> ON <input type="radio"/> OFF	onAction	<input type="radio"/> ON <input checked="" type="radio"/> OFF
onAborting	<input type="radio"/> ON <input checked="" type="radio"/> OFF	onModeChanged	<input type="radio"/> ON <input checked="" type="radio"/> OFF

作成者,連絡先: Nobuhiko Miyamoto <n-miyamoto@aist.go.jp>

ライセンス,使用条件,参考文献: LGPL

適用(L)

「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」を入力しておくともプロジェクト作成時に自動的に入力されるので便利。

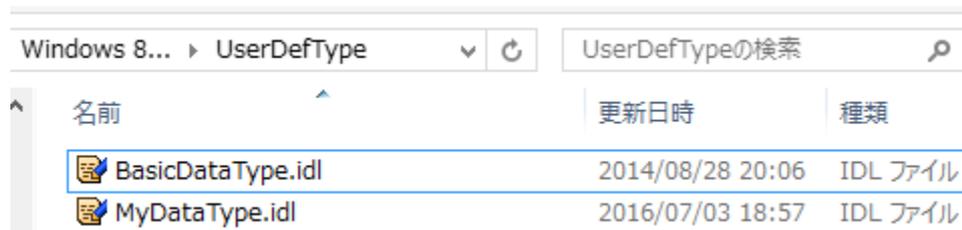
独自のデータ型の利用

- 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]
    
```

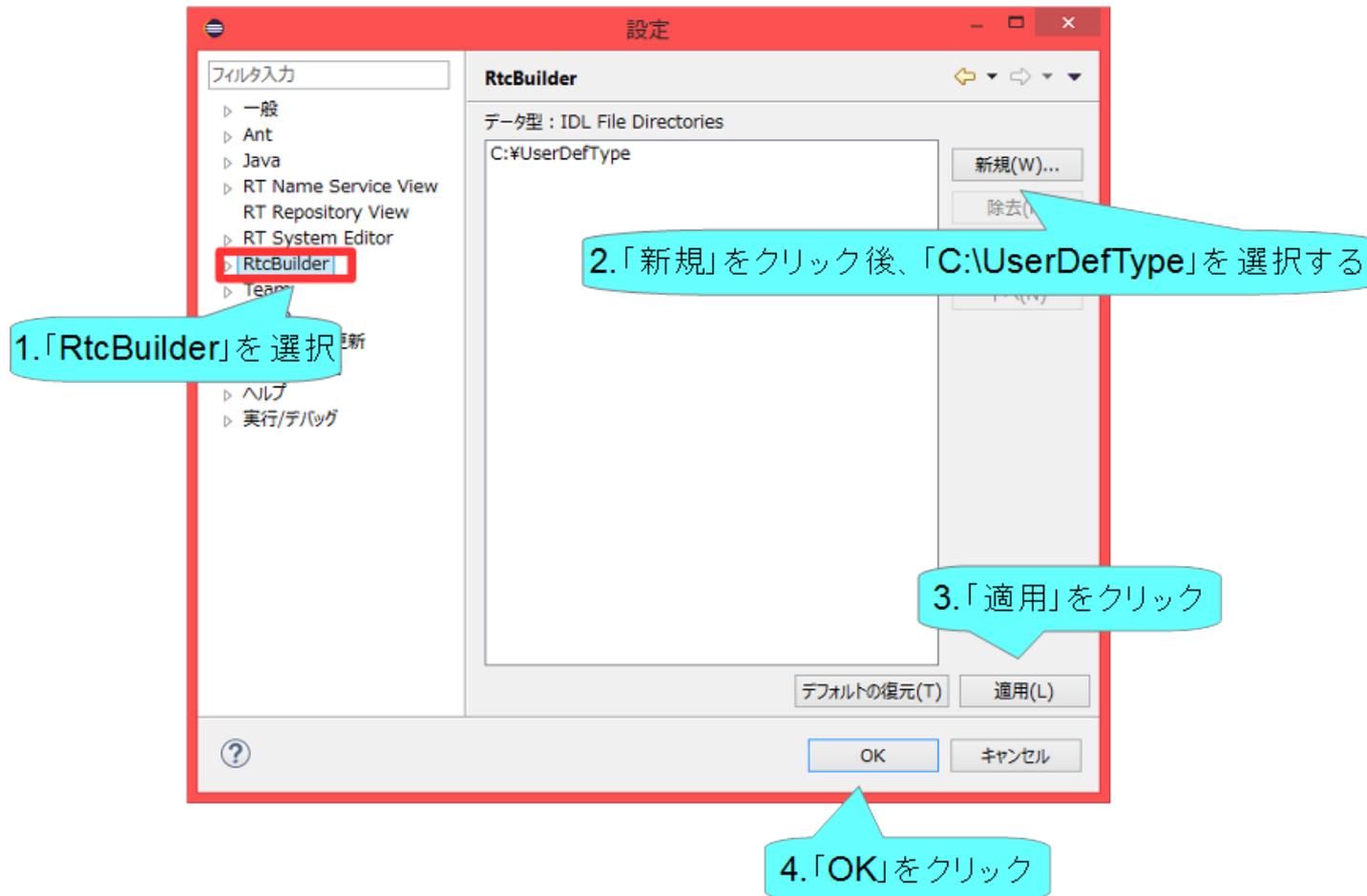
- 別のIDLファイルをインクルードしている場合は同じフォルダにコピーする



名前	更新日時	種類
BasicDataType.idl	2014/08/28 20:06	IDL ファイル
MyDataType.idl	2016/07/03 18:57	IDL ファイル

独自のデータ型の利用

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



独自のデータ型の利用

- 独自のデータ型でデータポートの通信を行う手順

▼ DataPortプロフィール

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

<p>*ポート名 (InPort)</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <tr><td style="padding: 2px;">in</td></tr> <tr><td style="padding: 2px;"> </td></tr> <tr><td style="padding: 2px;"> </td></tr> </table> <p style="text-align: right;">Add Delete</p>	in			<p>*ポート名 (OutPort)</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <tr><td style="padding: 2px;">out</td></tr> <tr><td style="padding: 2px;"> </td></tr> <tr><td style="padding: 2px;"> </td></tr> </table> <p style="text-align: right;">Add Delete</p>	out		
in							
out							

▼ Detail

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

ポート名:

*データ型	MyData
変数名	MyData
表示位置	RTC::Acceleration2D
	RTC::Acceleration3D
	RTC::ActuatorCurrent
Document	RTC::ActuatorGeometry

データ型一覧にMyDataが追加

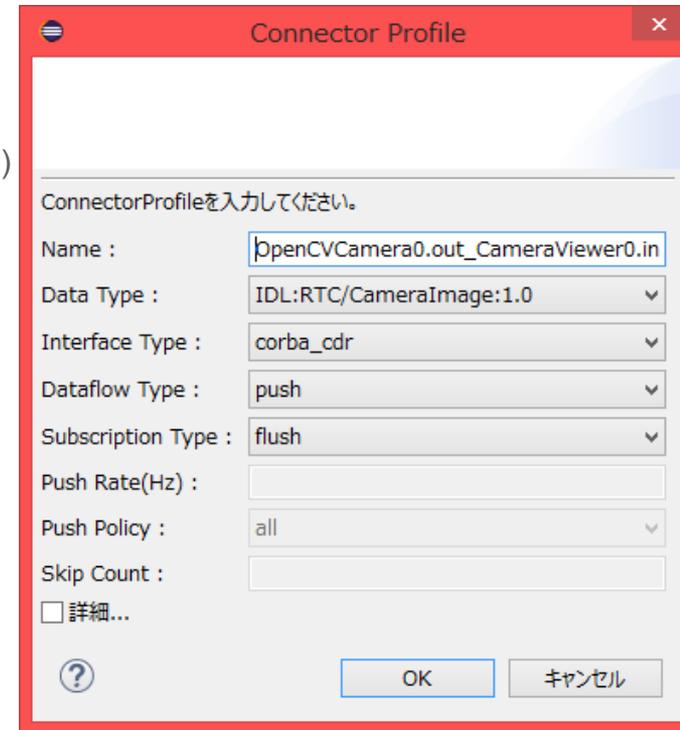
RT System Editor 補足

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
Data Type	ポート間で送受信するデータの型. ex) TimedOctet, TimedShortなど
Interface Type	データを送信方法. ex) corba_cdrなど
Data Flow Type	データの送信手順. ex) push, pullなど
Subscription Type	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). Subscription TypeがPeriodicの場合のみ有効
Push Policy	データ送信ポリシー. Subscription TypeがNew, Periodicの場合のみ有効. all, fifo, skip, newから選択
Skip Count	送信データスキップ数. Push PolicyがSkipの場合のみ有効

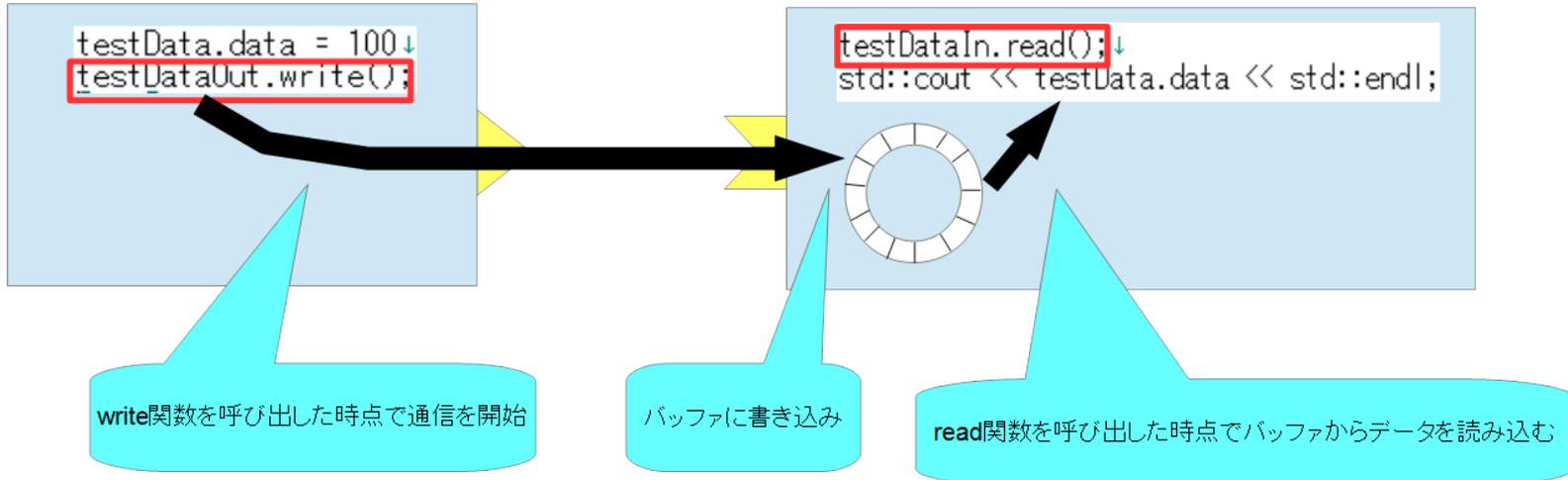
コネクタプロファイルの設定

- InterfaceType
 - データの送信方法
 - 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に問い合わせでデータを受け取る
- 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)

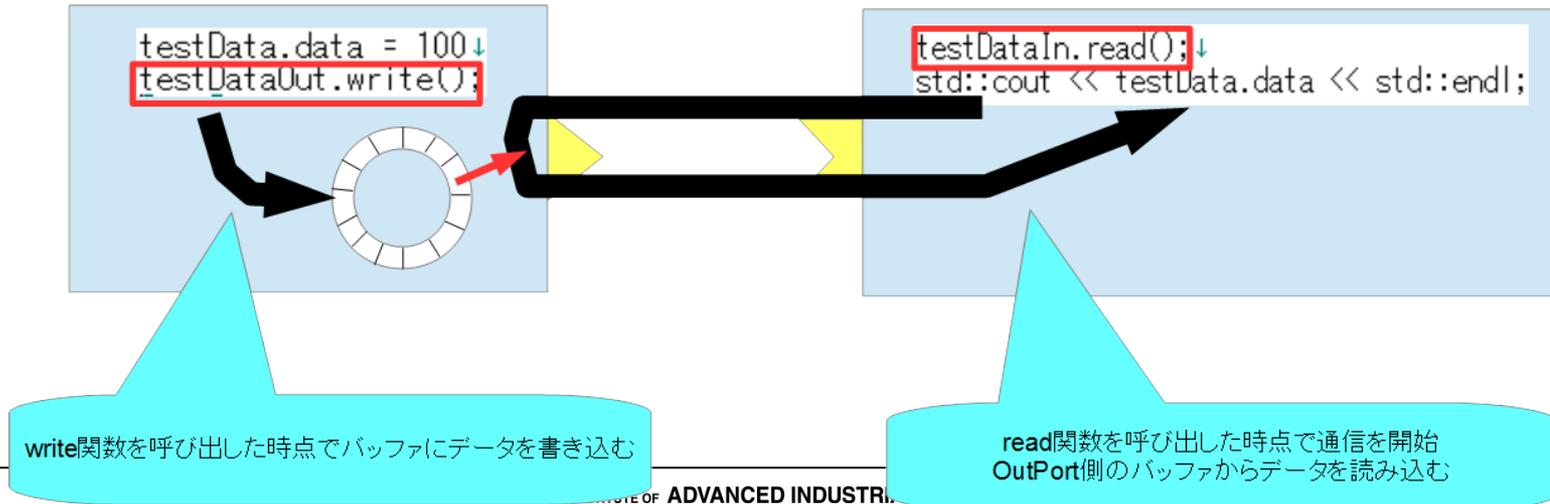


コネクタプロファイルの設定

- DataFlowType
 - Push

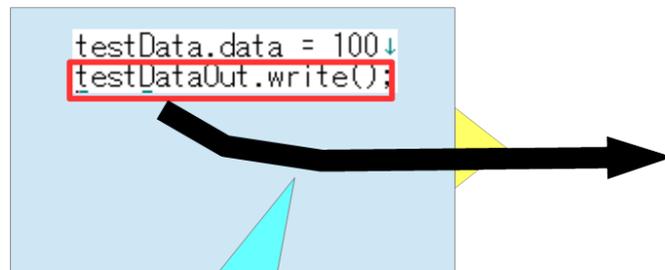


- Pull



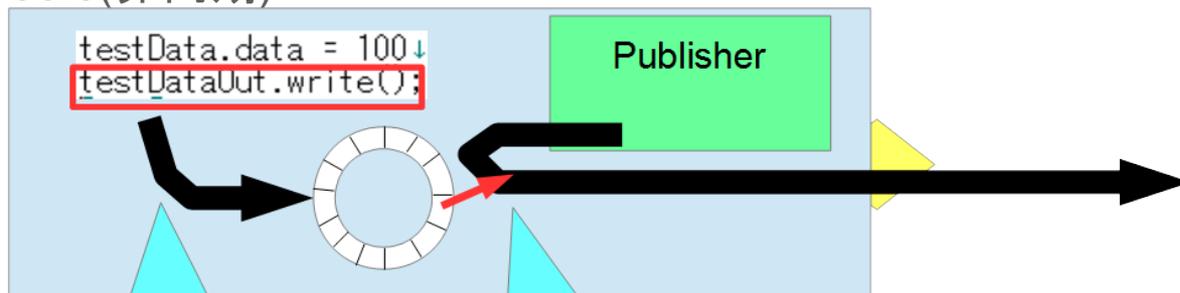
コネクタプロファイルの設定

- SubscriptionType
 - flush(同期)



write関数を呼び出した時点で通信を開始

- new、periodic(非同期)

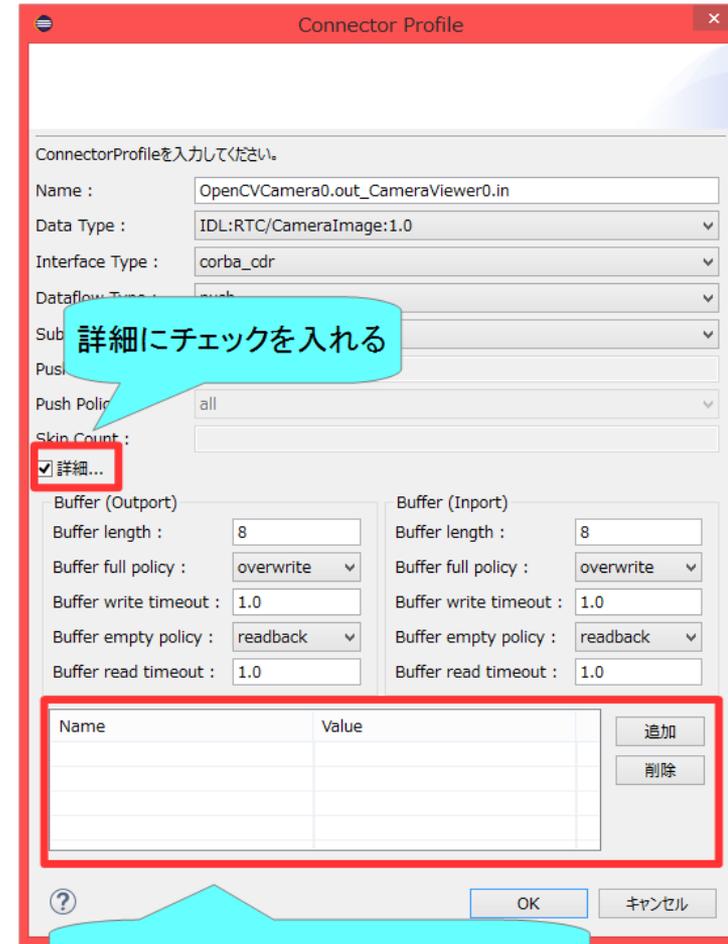


write関数を呼び出した時点でバッファにデータを書き込む

Publisherがデータ送信用のスレッドでデータの通信を行う

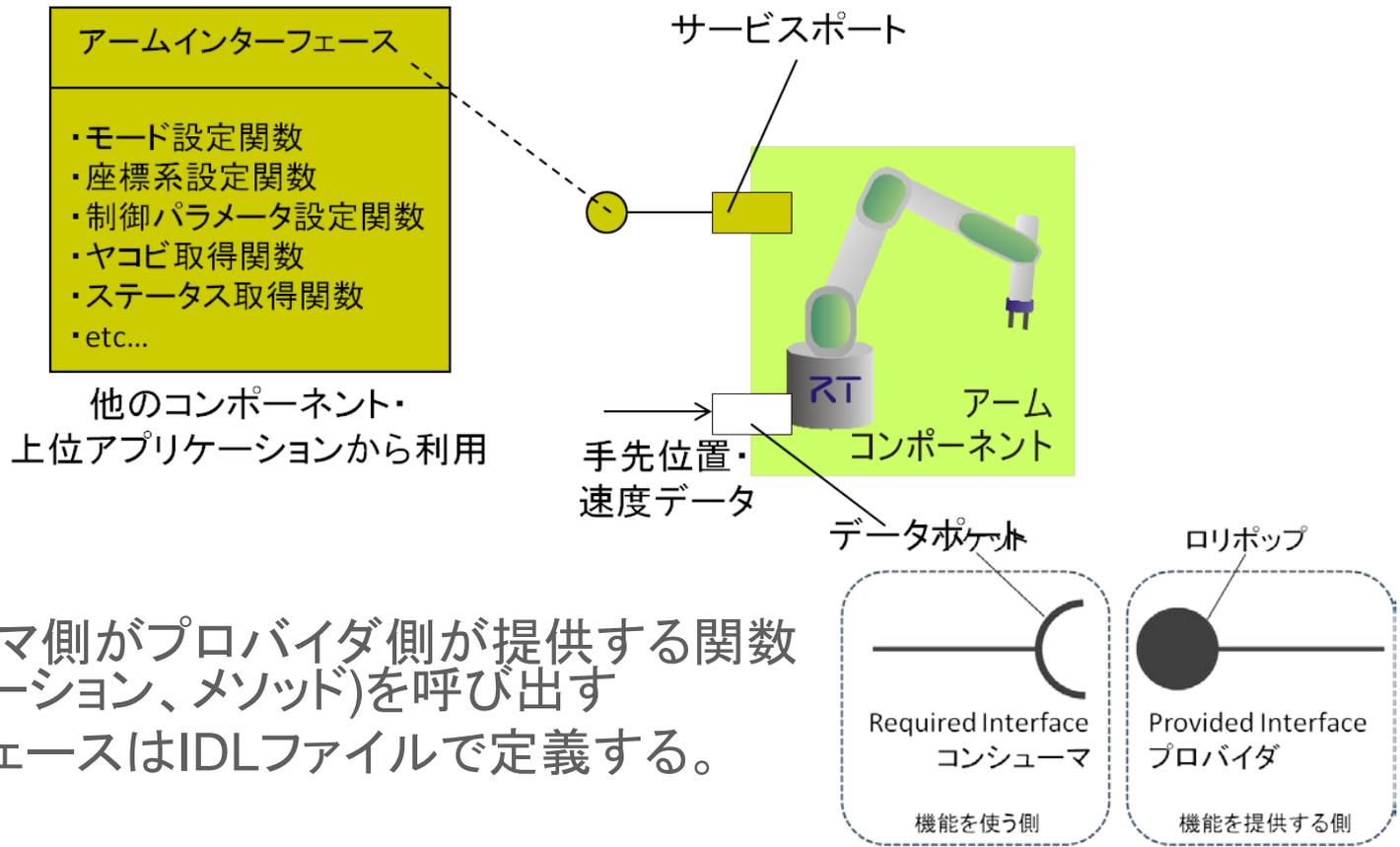
コネクタプロファイルの設定

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

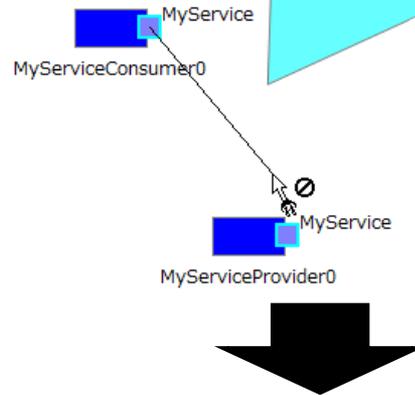
- コマンドレベルのやり取りを行うための仕組み
 - 任意のタイミングで操作を行いたい時などに使用
 - 例えばロボットアームのサーボを停止させる、ハンドを閉じる等



- 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

ポートの片方からもう片方へドラッグアンドドロップ



Port Profile

ポートプロファイルを入力してください。

Name : MyServiceProvider0.MyService_MyServiceConsumer0.MyService

詳細...

Consumer	Provider

追加 削除

Name	Value

追加 削除

OK キャンセル

名前の設定

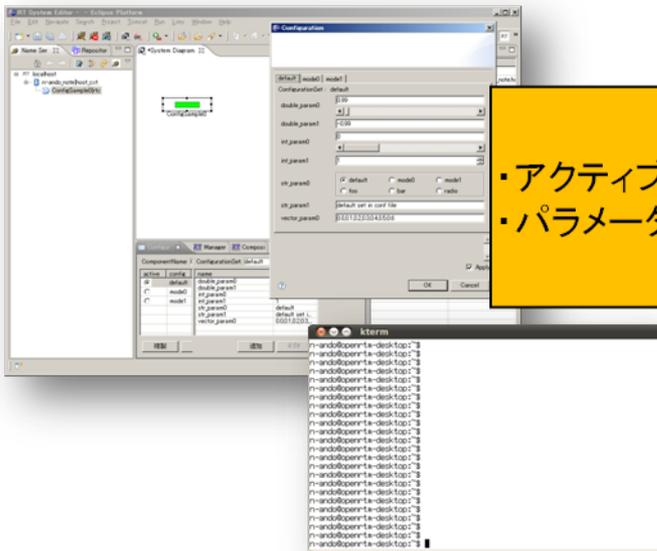
接続するインターフェースの設定
複数のインターフェースが定義されていた場合に、どのインターフェースに接続するかを設定

その他の設定を直接入力

コンフィギュレーションパラメータについて

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

ツール・アプリケーション



・アクティブセットの変更
・パラメータ値の変更

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。

コンポーネント コンフィギュレーションパラメータ

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

アクティブ
コンフィギュレーション
セット

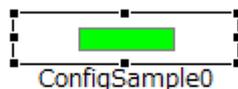
パラメータ変数: $m_Kp = 0.4$

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

コンフィギュレーションパラメータの設定

対象のRTCをクリックすると表示

「Configuration View」タブを選択



パラメーター一覧

ComponentName: ConfigSample0 ConfigurationSet: default

active	config	name	value
<input checked="" type="radio"/>	default	double_param0	0.99
<input type="radio"/>	mode0	double_param1	-0.99
<input type="radio"/>	mode1	int_param0	0
		int_param1	1
		str_param0	default
		str_param1	defau...
		vector_param0	0.0,0...

Buttons: 編集, 適用, キャンセル, 複製, 追加, 削除, 詳細

コンフィギュレーションセット一覧

コンフィギュレーションパラメータの設定

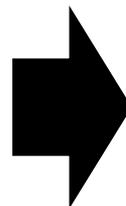
方法1

ComponentName: ConfigSample0 ConfigurationSet: default

active	config	name	value
<input checked="" type="radio"/>	default	double_param0	0.99
<input type="radio"/>	mode0	double_param1	-0.99
<input type="radio"/>	mode1	int_param0	0
		int_param1	
		str_param0	
		str_param1	
		vector_param0	

Buttons: 複製, 追加, 削除, 詳細, 適用, キャンセル

Callout: 編集ボタンを押す



ConfigurationSet: default

double_param0: 1.0

double_param1: 0.99

int_param0: 0

int_param1: 1

vector_param0: 0,0,0,1,0,2,0,3,0,4,0,5,0,6

Buttons: OK, キャンセル

Callout: パラメータを編集する

方法2

ComponentName: ConfigSample0 ConfigurationSet: default

active	config	name	value
<input checked="" type="radio"/>	default	double_param0	1.0
<input type="radio"/>	mode0	double_param1	-0.99
<input type="radio"/>	mode1	int_param0	0
		int_param1	
		str_param0	default
		str_param1	fa...
		vector_param0	0...

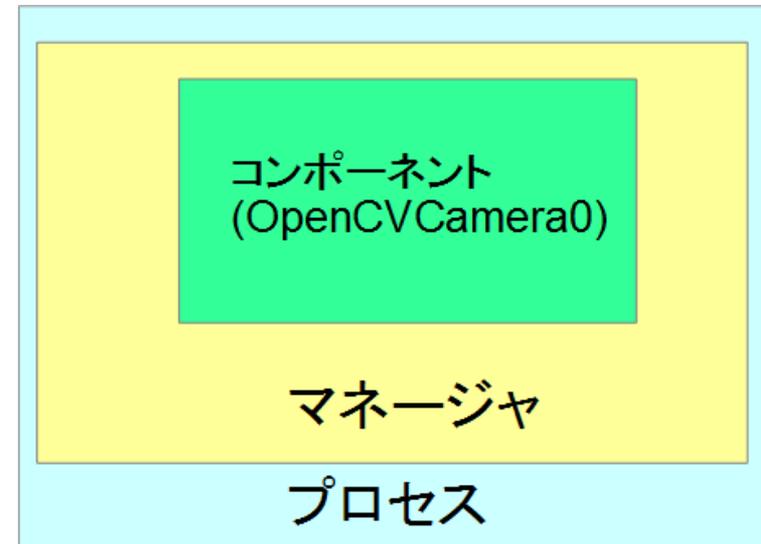
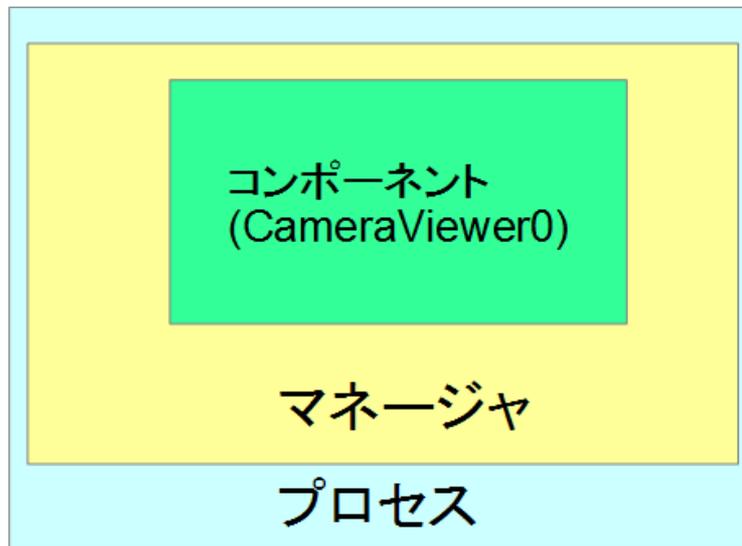
Buttons: 複製, 追加, 削除, 詳細, 適用, キャンセル

Callout: 適用ボタンを押す

パラメータを編集する

マネージャの操作

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



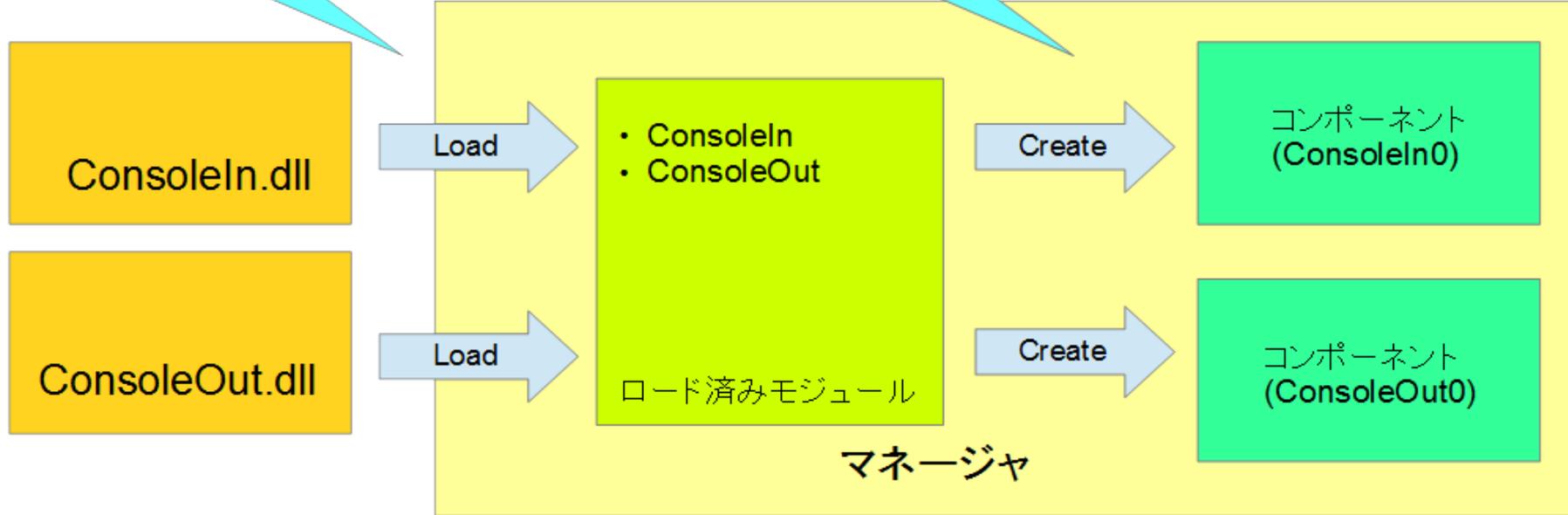
基本的にマネージャは各プロセスに1つ起動する。
マネージャがコンポーネントを起動する

マネージャの操作

1. モジュールをロードする

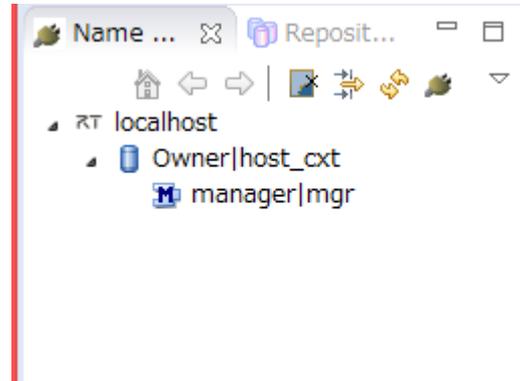
- C++: .dll, .so
- Python: .py
- Java: .jar

2. コンポーネントを起動する



マネージャの操作

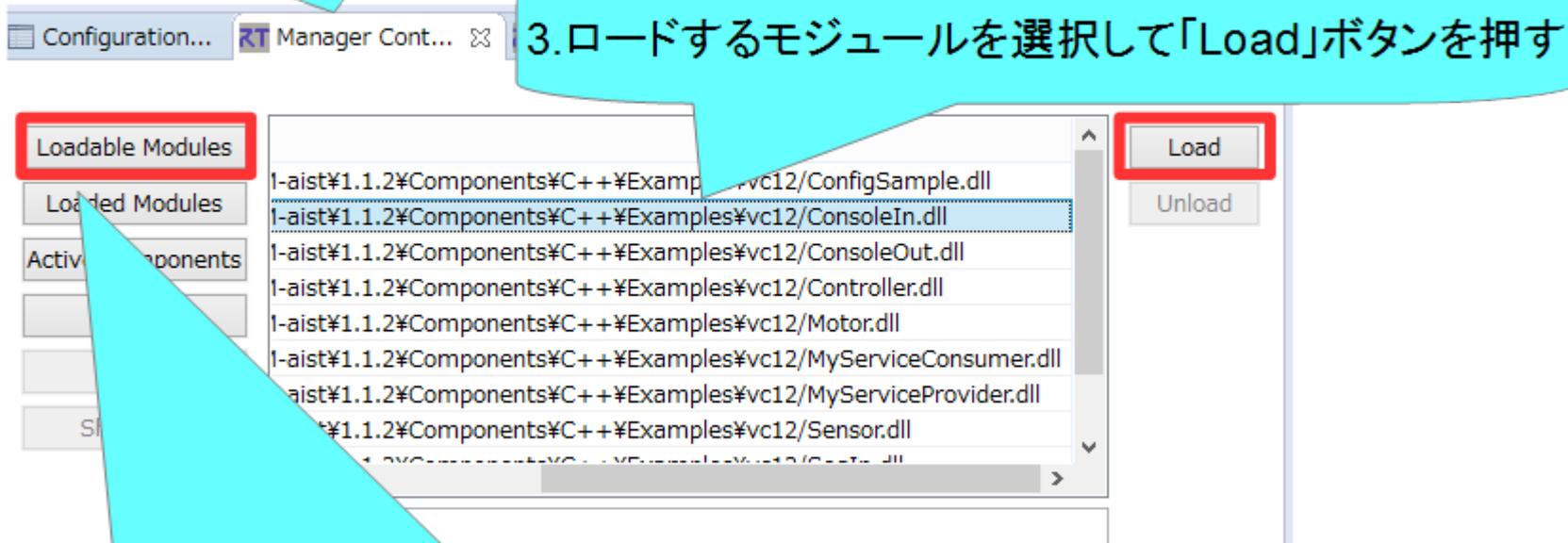
- マスターマネージャの起動、RT System Editorからの操作によるRTCの生成までの手順を説明する
 - rtc.confの設定
 - 「manager.is_master」を「YES」に設定して起動するマネージャをマスターに設定する
 - manager.is_master: YES
 - モジュール探索パスの設定
 - manager.modules.load_path: ., C:\Program Files (x86)\OpenRTM-aist\1.1.2\Components\C++\Examples\vc12
 - 作成したrtc.confを設定ファイルの指定してrtcd.exe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 System Editorのネームサービスビューにマネージャが表示される



マネージャの操作

- モジュールのロード

1.「Manager Control View」タブを選択

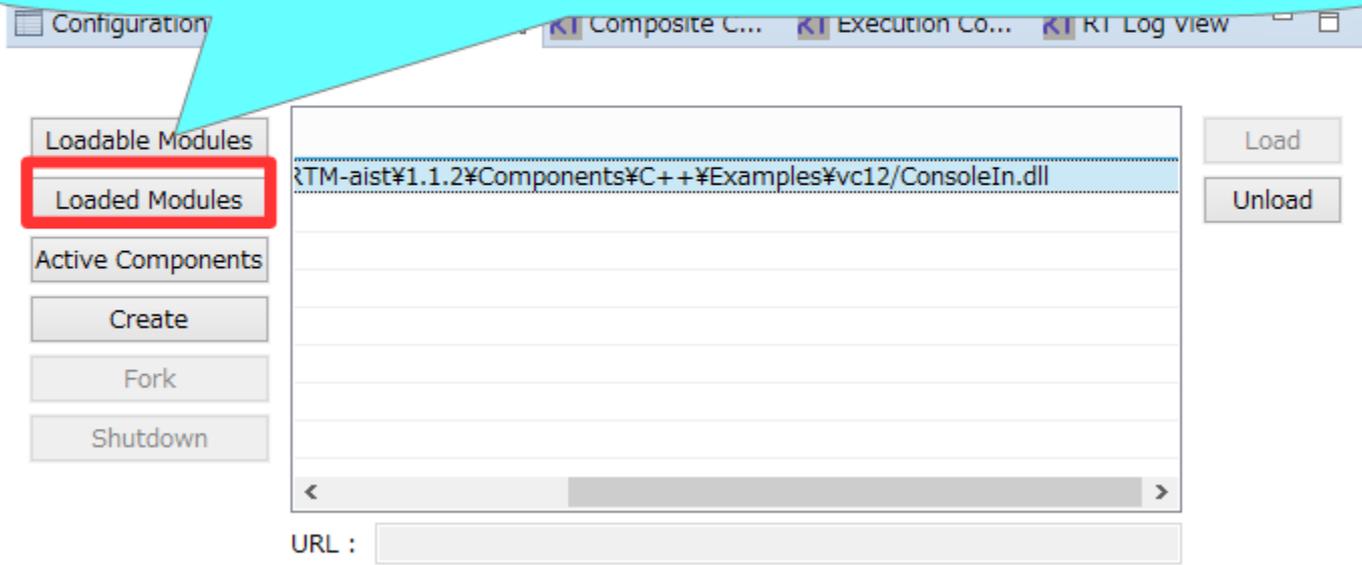


2.「Loadable Modules」ボタンを押すとロード可能なモジュール一覧表示

マネージャの操作

- モジュールのロード

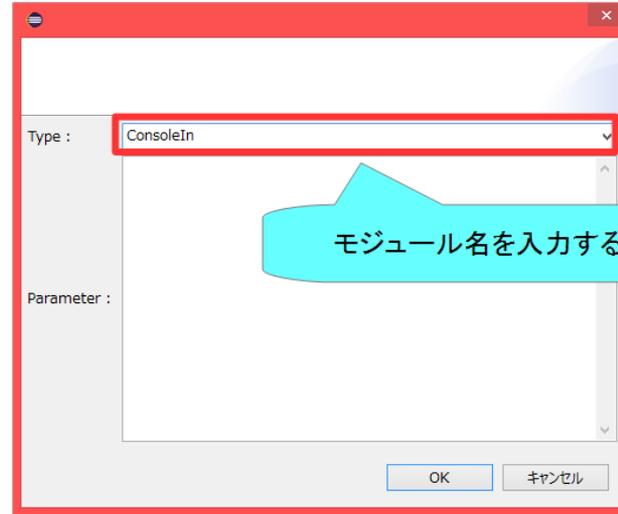
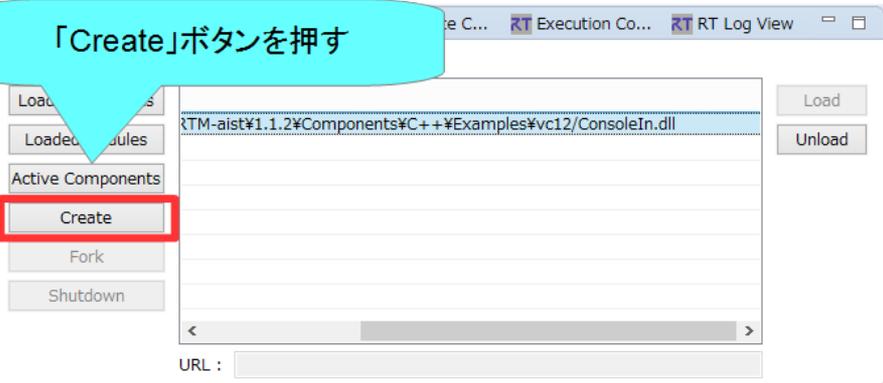
「Loaded Modules」ボタンを押すとロード済みのモジュール一覧表示



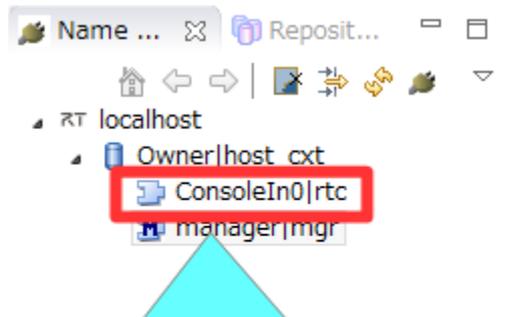
マネージャの操作

- RTCの生成

「Create」ボタンを押す



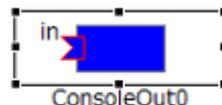
モジュール名を入力する



指定したRTCが起動する

実行コンテキストの操作

RTCをクリック



「Execution Context View」タブを選択

実行周期

component: ConsoleOut0

rate: 1000.0

Execution Context	
owned0	

Name	Value
id	0
kind	PERIODIC
state	RUNNING
component_state	INACTIVE
owner	ConsoleOut0
participants	0

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

関連付けている実行コンテキスト一覧

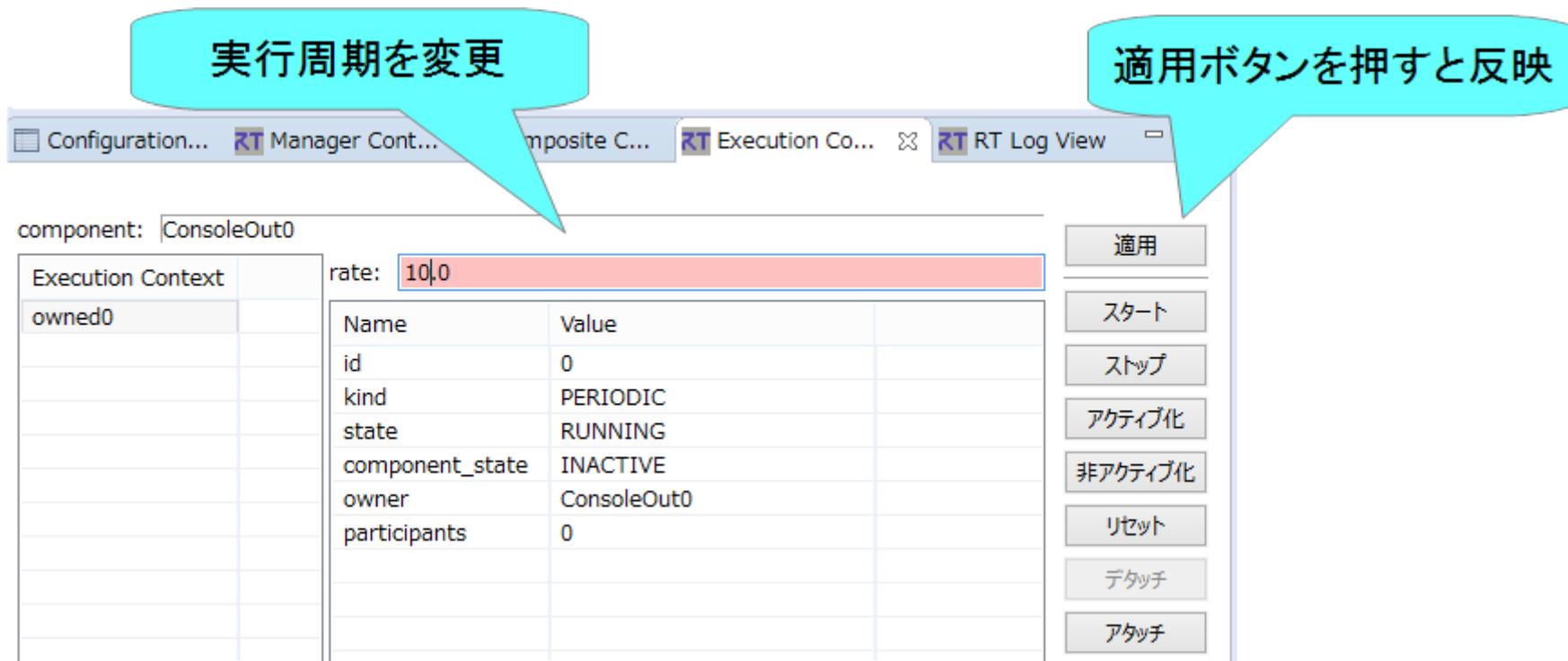
実行コンテキストの情報

実行コンテキストの操作

- 実行周期の設定

実行周期を変更

適用ボタンを押すと反映



component: ConsoleOut0

Execution Context	rate:
owned0	10.0

Name	Value
id	0
kind	PERIODIC
state	RUNNING
component_state	INACTIVE
owner	ConsoleOut0
participants	0

適用

スタート

ストップ

アクティブ化

非アクティブ化

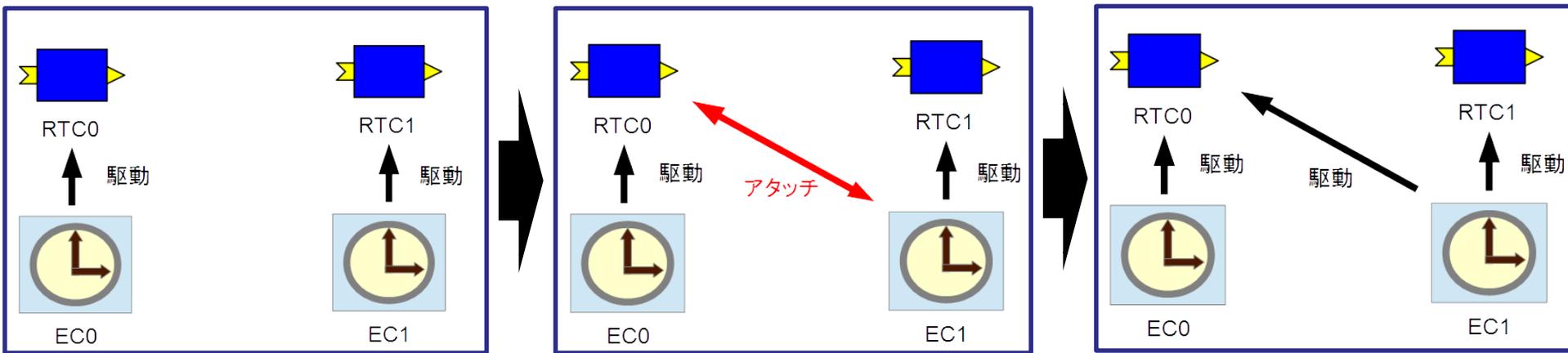
リセット

デタッチ

アタッチ

実行コンテキストの操作

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



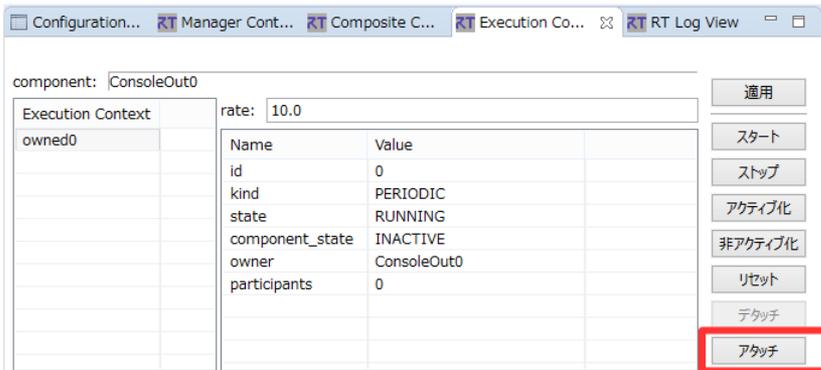
RTC0とRTC1は別々の
実行コンテキストで駆動

RTC0とEC1を関連付ける

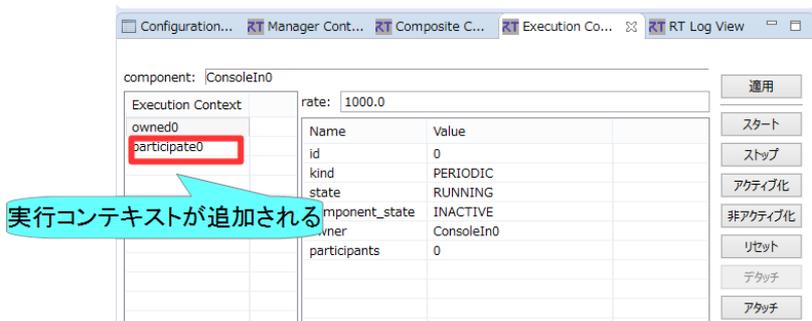
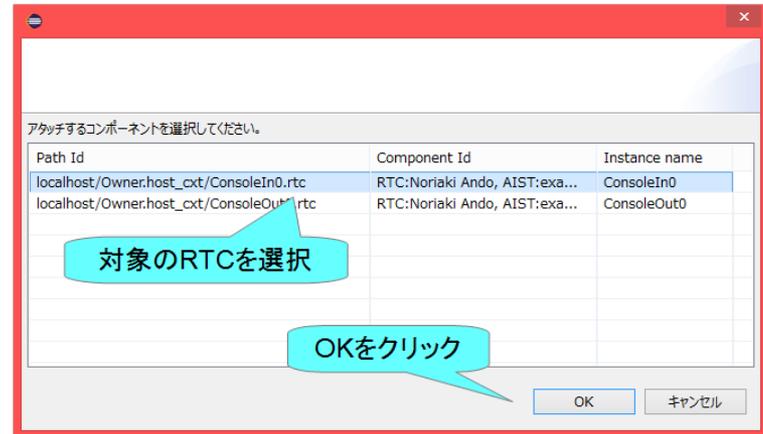
EC1がRTC0も駆動する

実行コンテキストの操作

- 実行コンテキストの関連付け

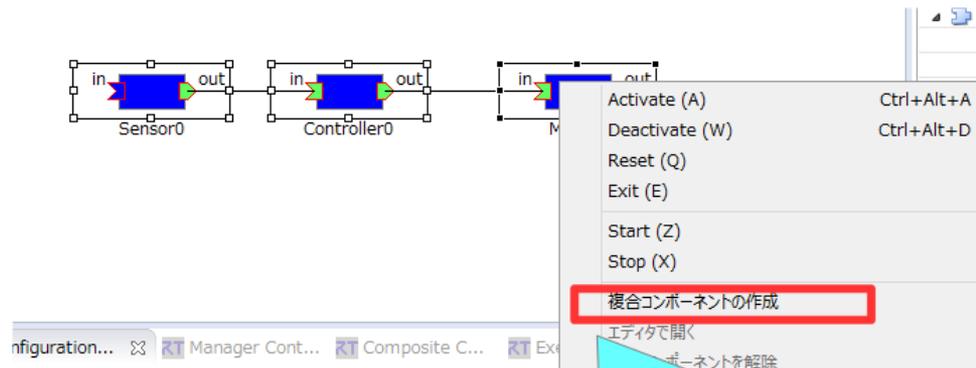
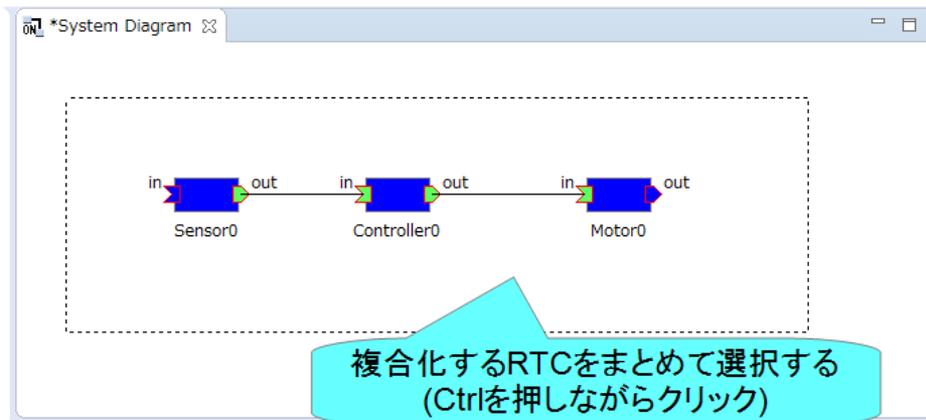


アタッチボタンを押す



複合コンポーネントの操作

- 複合コンポーネントの生成



右クリックして「複合コンポーネントの作成」を選択

複合コンポーネントの操作

- 複合コンポーネントの生成

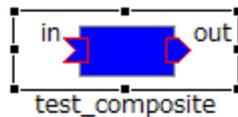
The image shows a 'New Composite Component' dialog box with several callouts explaining its fields:

- 複合コンポーネントを起動するマネージャの指定**: Points to the 'Manager' dropdown menu.
- 名前を設定**: Points to the 'Name' text field.
- タイプを設定**: Points to the 'Type' dropdown menu.
- 複合コンポーネントに表示するポートを指定**: Points to the 'Port' list of checkboxes.
- OKをクリックする**: Points to the 'OK' button.

Below the dialog box, a diagram shows a component named 'test_composite' with an 'in' port on the left and an 'out' port on the right, connected by a blue arrow. A callout below the diagram states: **複合コンポーネントが生成される**.

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

複合コンポーネントの操作



複合コンポーネントをクリック

「Composite Component View」タブを選択

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

component: test_composite type: PeriodicECShared

	component	port
<input type="checkbox"/>	Controller0	Controller0.in
<input type="checkbox"/>	Controller0	Controller0.out
<input checked="" type="checkbox"/>	Sensor0	Sensor0.in
<input type="checkbox"/>	Sensor0	Sensor0.out
<input type="checkbox"/>	Motor0	Motor0.in
<input checked="" type="checkbox"/>	Motor0	Motor0.out

適用
キャンセル

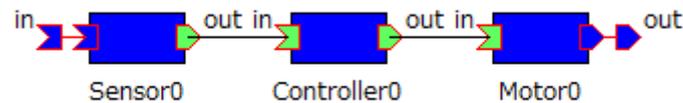
適用ボタンを押すと変更を反映

表示するポートの選択

複合コンポーネントの操作



RTCをダブルクリック

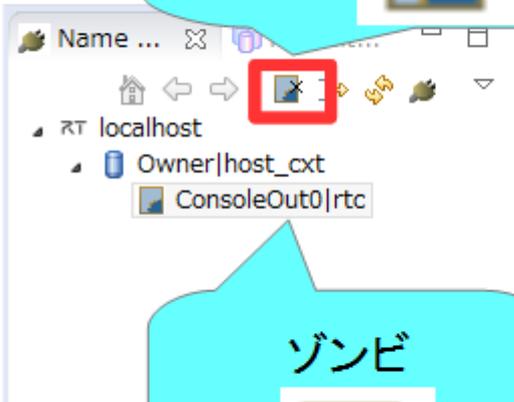


子コンポーネントを表示

ゾンビの削除

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

ゾンビクリアボタンを押す



ゾンビ



ゾンビが消える

RT System Editorに関する設定

