SUGAR SWEET ROBOTICS CO., LTD.

OpenRTM-aist入門

株式会社SUGAR SWEET ROBOTICS　代表取締役

早稲田大学基幹理工学部表現工学科 尾形哲也研究室 客員次席研究員

芝浦工業大学SIT研究所　客員研究員

菅　佑樹

インストール (Windows)

- まず，UACをOFFにしましょう.
- OpenRTM-aist バージョン1.1.1 C++言語版 (32bit版)
  - http://openrtm.org/openrtm/ja/node/5711
    - JDK8・・・Oracle Webサイトからダウンロード．JREはOpenRTMに同梱だが，Java版を開発する場合等に必要なのでインストールをオススメ(32bit版オススメ)
    - Python 2.7・・・ python-2.7.9.msi(32bit)
    - PyYAML (2.7対応)・・・PyYAML-3.11.win32-py2.7.exe
    - OpenRTM-aist・・・OpenRTM-aist-1.1.1-RELEASE_x86_vc12.msi (32bit)
    - GUIツール入りeclipse・・・ eclipse381-openrtp110rc5v20150317-ja-win32.zip (32bit)
  - Doxygen
    - Doxygen win32 ・・・ doxygen-1.8.9.1-setup.exe
  - CMake 3.2
    - CMake 3.2 ・・・ cmake-3.2.1-win32-x86.exe
- Visual Studio 2013 Community Edition
  - https://www.microsoft.com/ja-jp/dev/products/community.aspx

インストール (OSX10.9以降)

- SUGAR SWEET ROBOTICSで管理しているOSX版パッケージの使い方
- OSXは10.9.3+Xcodeは5.1 (これ以外の環境では，自分でビルドする必要があります)
- C++版インストーラ
  - http://sugarsweetrobotics.com/pub/Darwin/OpenRTM-aist/cxx/1.1/OpenRTM-aist-1.1-cxx-osx-10.9.dmg
  - $HOME/.bash_profileに以下の記述を追加する必要がある.
    - RTM_ROOT=/usr/local/include/openrtm-1.1
    - PKG_CONFIG_PATH=/usr/local/lib/pkgconfig
- これ以外に，以下のツールが必要です（自分でビルドする場合も必要）
  - Xcodeコマンドラインツール
    - https://daw.apple.com/cgi-bin/WebObjects/DSAuthWeb.woa/wa/login?&appIdKey=891bd3417a7776362562d2197f89480a8547b108fd934911bcbea0110d07f757&path=%2F%2Fdownloads%2Findex.action （Apple Developerにサインイン必要）
  - cmake http://www.cmake.org/files/v2.8/cmake-2.8.10.2-Darwin64-universal.dmg
  - doxygen http://sugarsweetrobotics.com/pub/Darwin/doxygen/Doxygen-1.8.3.1.dmg
  - pyyaml http://sugarsweetrobotics.com/pub/Darwin/libs/PyYAML-3.10.tar.gz
  - pkg-config http://sugarsweetrobotics.com/pub/Darwin/pkgconfig/PkgConfig.dmg
  - Sun JDK （ターミナルでjavaと打つと，インストールされていなければ公式サイトに飛びます）
- また，ツールとして，RTシステム全部入りのEclipse (Mac版) をOpenRTM-aistのページからダウンロードする必要があります.

動作確認 (Windows)

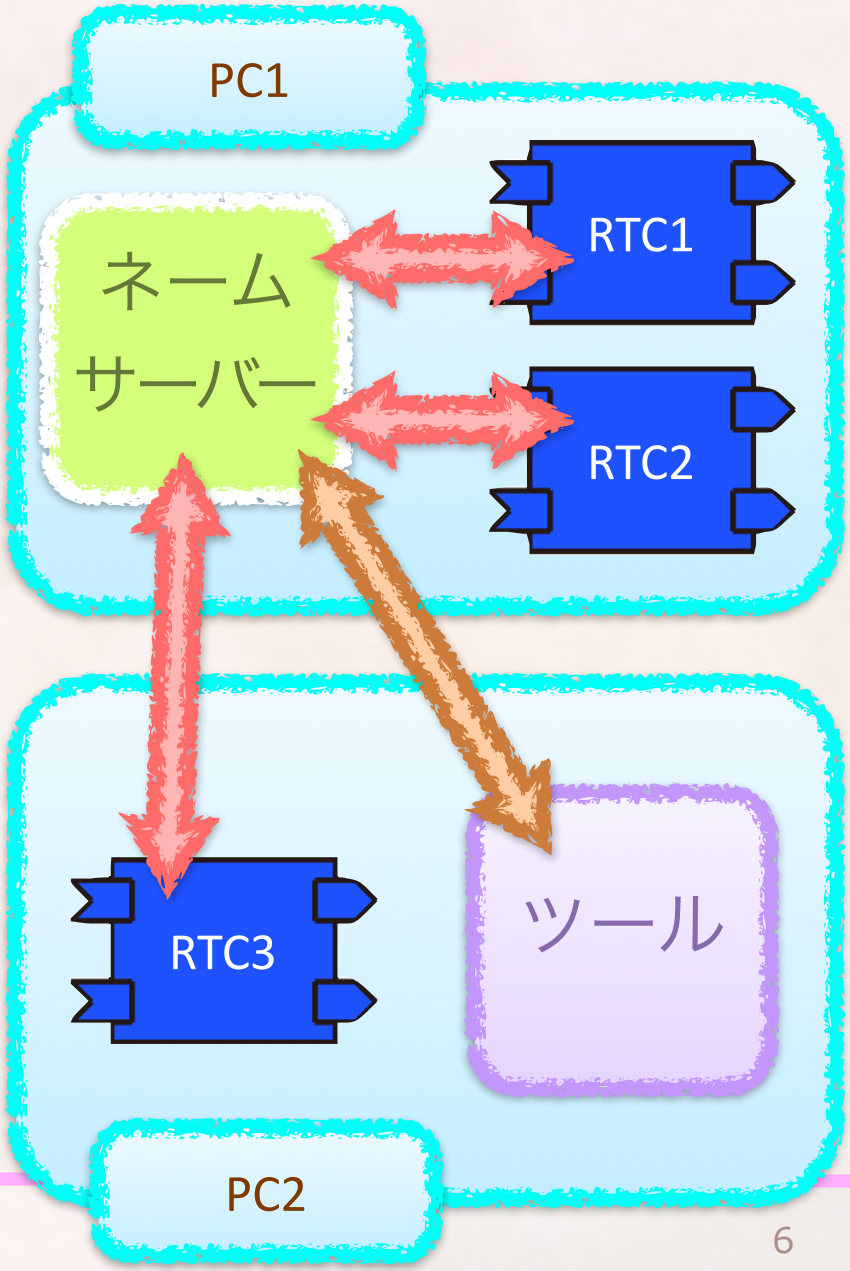
- ネームサービスの起動
  - 「スタートメニュー」＞「OpenRTM-aist 1.1」＞「tools」＞「Start C++ Naming Service」
- RTCの起動（ConsoleIn, ConsoleOut）
  - 「スタートメニュー」＞「OpenRTM-aist 1.1」＞「C++」＞「components」＞「examples」＞「ConsleInComp.exe」と同じく「ConsoleOutComp.exe」
- GUIツールの起動
  - eclipse***.zipを展開
  - Eclipseフォルダ内のeclipse.exeを起動
  - ワークスペースはデフォルトでOK
    - 通常は，/Users/$USER_NAME/workspace

動作確認 (OSX / Ubuntu)

- 新規ターミナルからNameServer起動
  - $ rtm-naming (Ubuntuだと，現在のプロセスをkillして再起動するか聞かれるので，必ずyesして再起動する．デフォルトで立ち上がっているサービスはRTMに不向き)
- ターミナルからConsoleIn起動
  - $ cd /usr/local/share/openrtm-1.1/examples/ (Ubuntuなら/usr/share/openrtm-1.1/examples)
  - $ ./ConsoleInComp
- ターミナルからConsoleOut起動
  - $ cd /usr/local/share/openrtm-1.1/examples/ (Ubuntuなら/usr/share/openrtm-1.1/examples)
  - $ ./ConsoleOutComp
- ターミナルからEclipseを起動
  - Downloadsにダウンロードして，そのまま展開してeclipseというディレクトリが出来た，と仮定します．
    - $ cd $HOME/Downloads/eclipse
    - $ cd Eclipse.app/Contents/MacOS
    - $ ./eclipse
  - Ubuntuなら解答したディレクトリに移動してeclipseというバイナリを実行するのみ

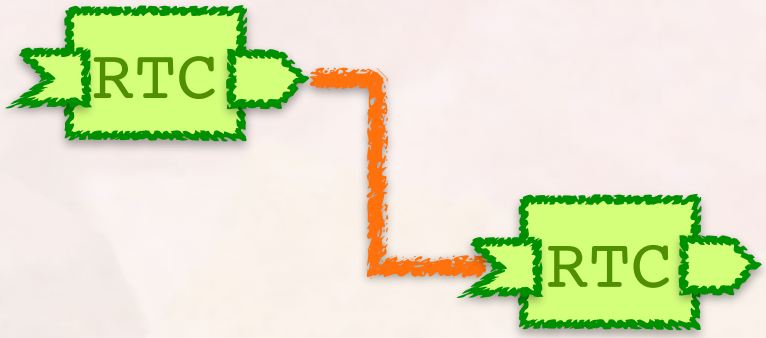
ネームサービスとは

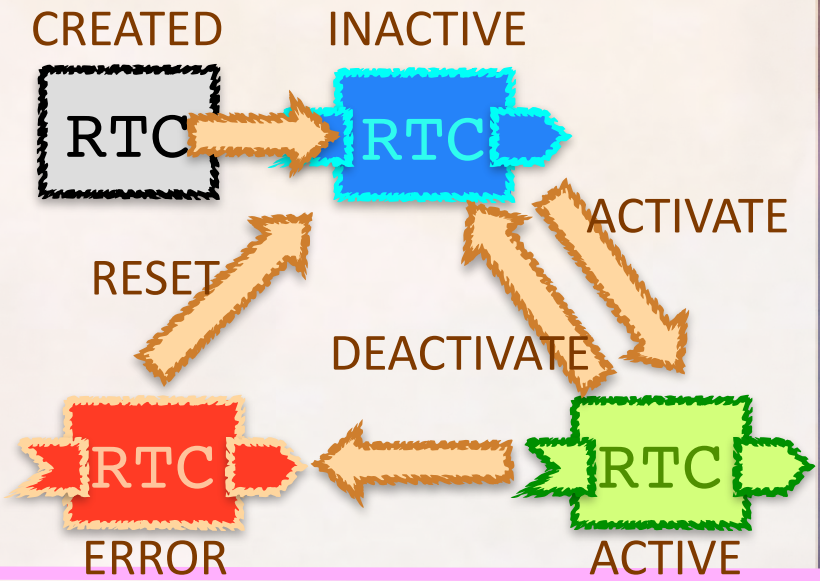
- 実行中のRTCの管理
  - OpenRTM-aistを使う場合，RTCを使ったシステムでは最低１つ必要
  - 異なるホストのRTCも登録可能
  - 複数のネームサービスを併用可能



ツールの役割

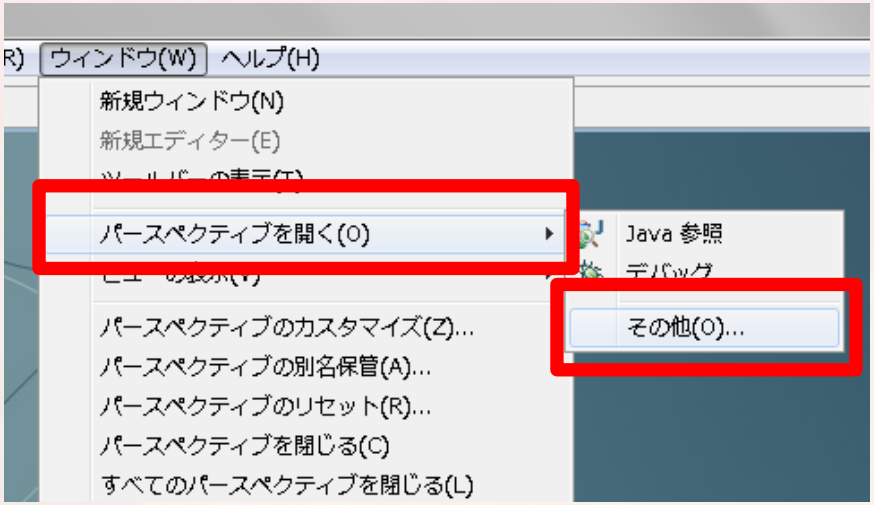
- ツールの種類
  - RT System Editor
    - Eclipseのプラグイン
    - GUI
  - rtshell
    - コマンドからRTCを制御
    - CUI．スクリプト化が可能＝自動化
  - 自作ツール
    - ツール作成自体も容易
- ツールの役割
  - RTC間の接続
  - RTCのコンフィグレーションの変更
  - RTCの状態の変更
  - 実行コンテキストの制御

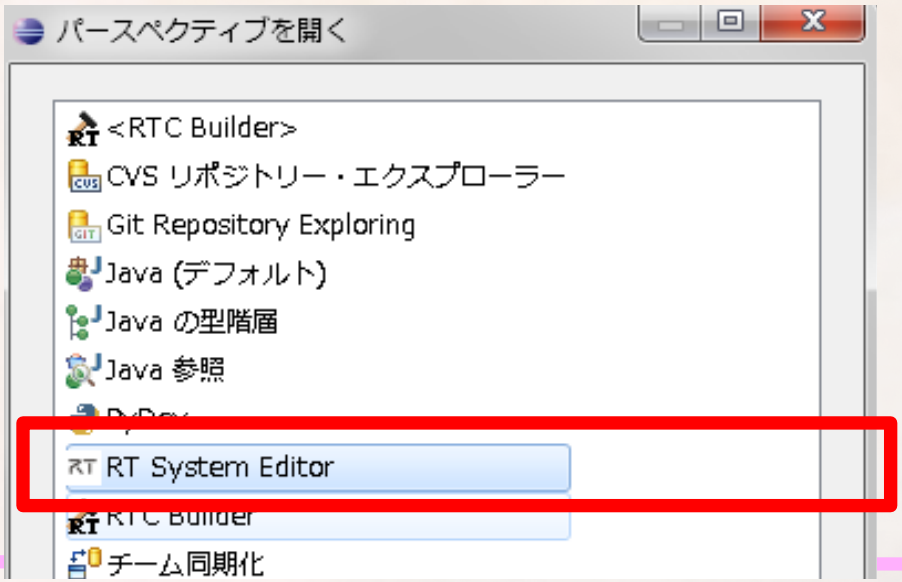




RT System Editorの使い方

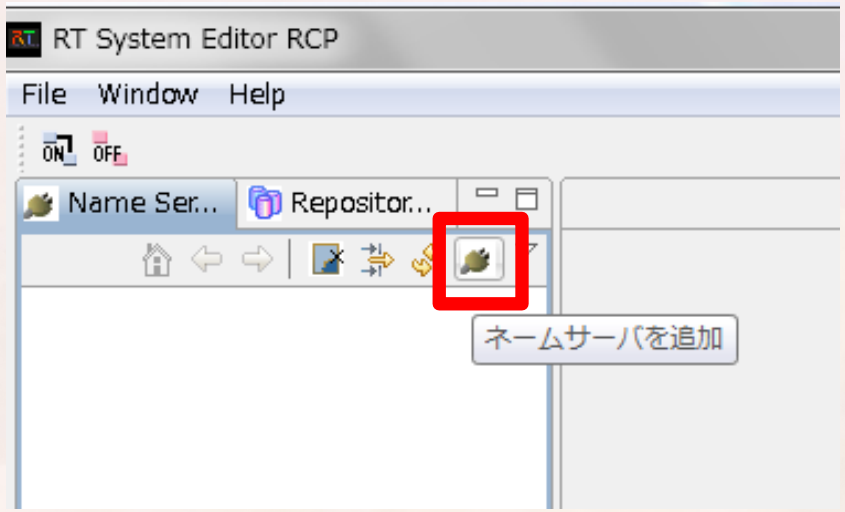
- パースペクティブを「RT System Editor」に変更
  - パースペクティブとはEclipseの作業画面のレイアウトタイプ
  - メニュー＞「ウィンドウ」＞「パースペクティブを開く」＞「その他」
  - 「RT System Editor」を選択

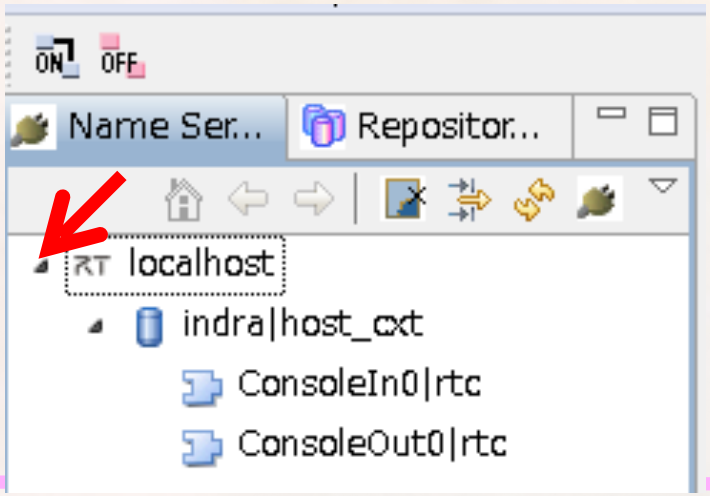




RT System Editorの使い方

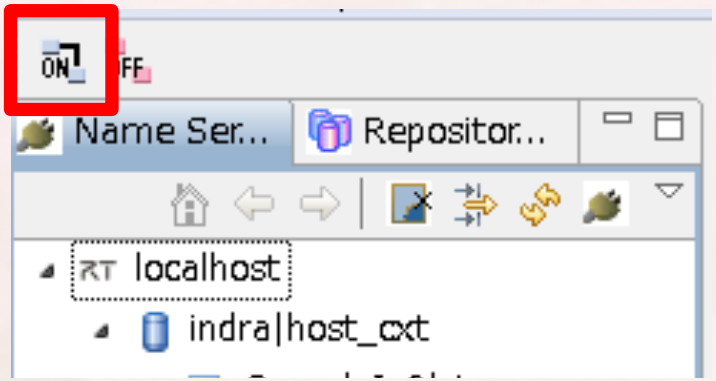
- 所望のネームサーバが見つからない場合は追加処理
  - ネームサーバとは実行中のRTCの管理を行うサーバ
    - localhost (自分自身)
    - デフォルトでlocalhostを検索し，ネームサーバーが発見されれば追加される
    - Macでは不具合があるので，ネームサーバーを起動する前にRTSEを起動した方がいい
- ネームサーバに起動したRTCが登録されていれば成功
  - ConsoleInは
    - 「/localhost/お使いのPC名.host_cxt/ConsoleIn0.rtc」という名前でネームサーバーに登録されているはず
  - 名前の登録ルールを変更することも簡単
    - rtc.conf

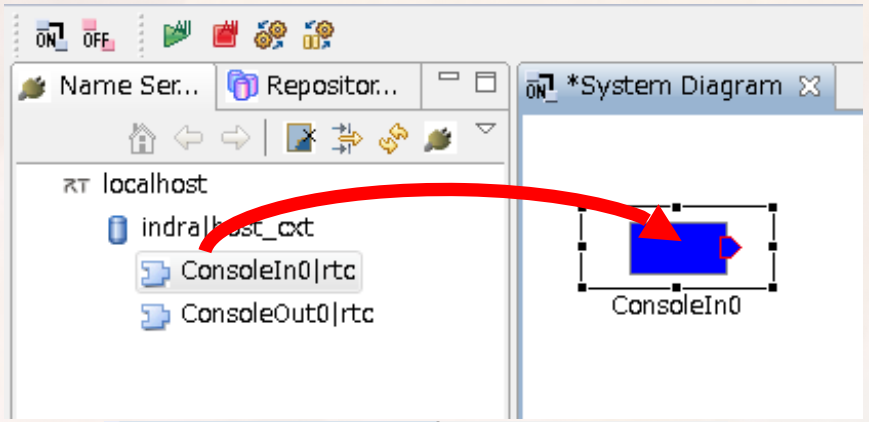


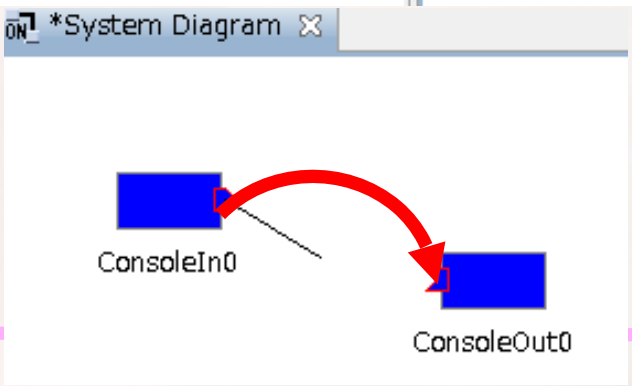


RT System Editorの使い方

- System Editorを開いてRTシステムを編集する
  - File > Open On-Line System Editorもしくはツールバー上のボタン (右図)
  - メインビューに空の「System Diagram」が表示される
- ネームサービスビューからドラッグ＆ドロップ
  - 利用するRTCをすべて表示
- ポートとポートをドラッグ＆ドロップで接続
  - 表示されるダイアログはOK

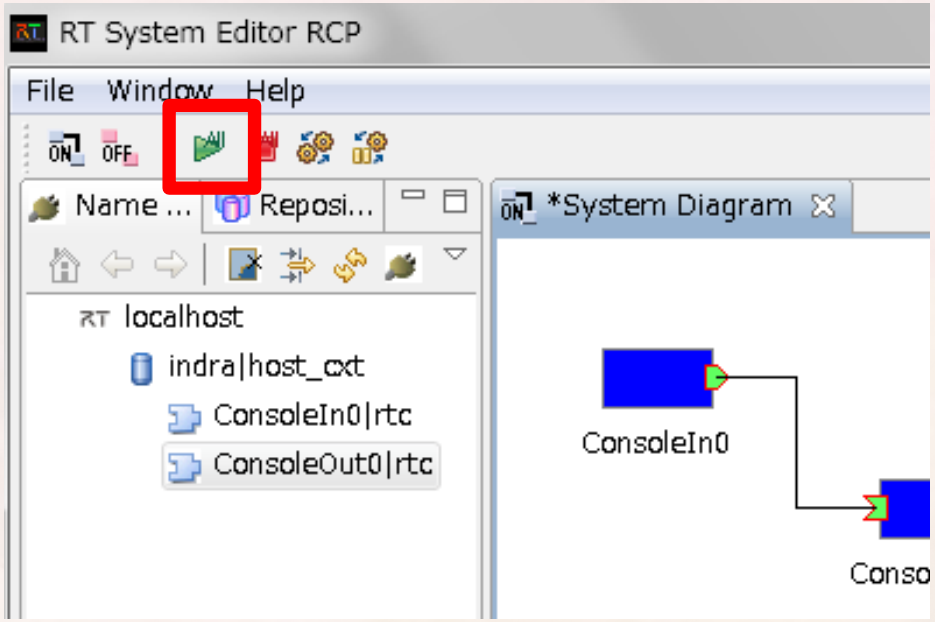






RT System Editorの使い方

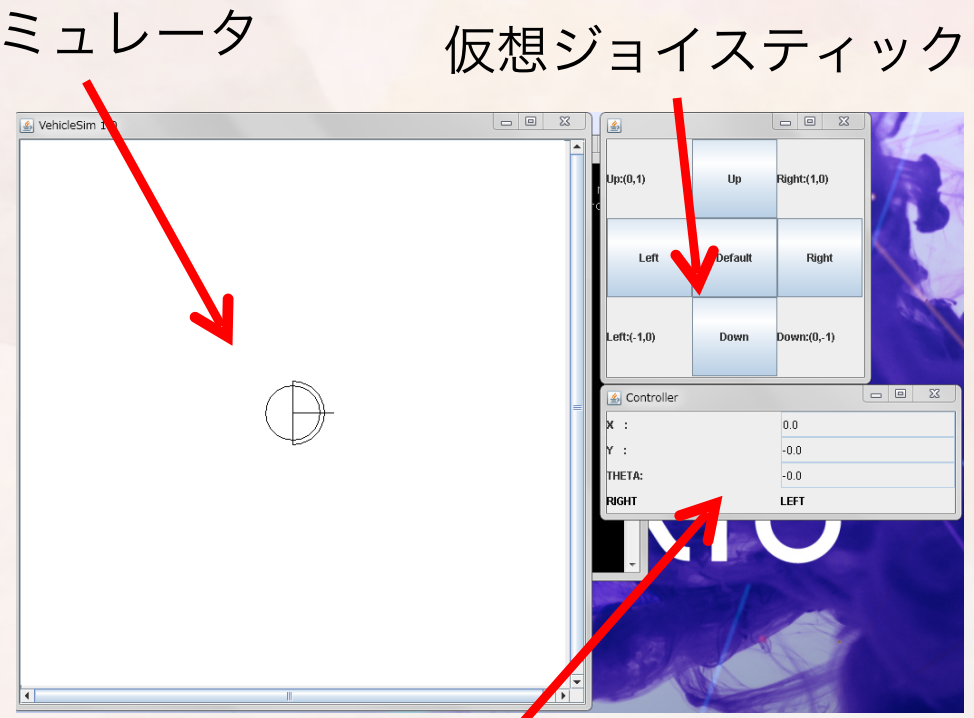
- すべてのRTCをACTIVATE
- ConsoleInのウィンドウに「Please Input number」と表示
  - 適当な数字を入れるとConsoleOut側に表示
  - RTC間の通信が行われたことが分かる
- DEACTIVATEする場合は，指令を送ってからConsoleInに数字を送る必要がある
  - （scanf入力待ちになっている）



OpenRTM-aist学習用台車シミュレータ

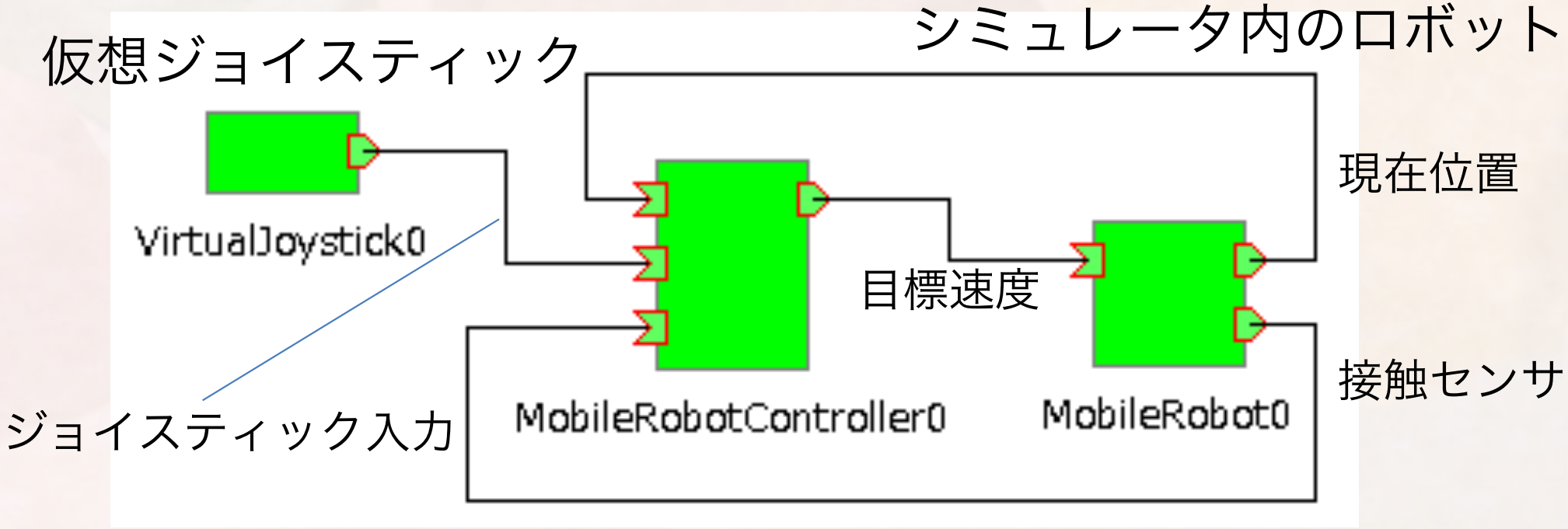
http://ysuga.net/?p=133

- Loader.batを実行
  - シミュレータ
  - 仮想ジョイスティック
  - コントローラ
- すべて接続してACTIVATE
- ジョイスティックで操作
- コントローラGUIで位置や接触センサを確認



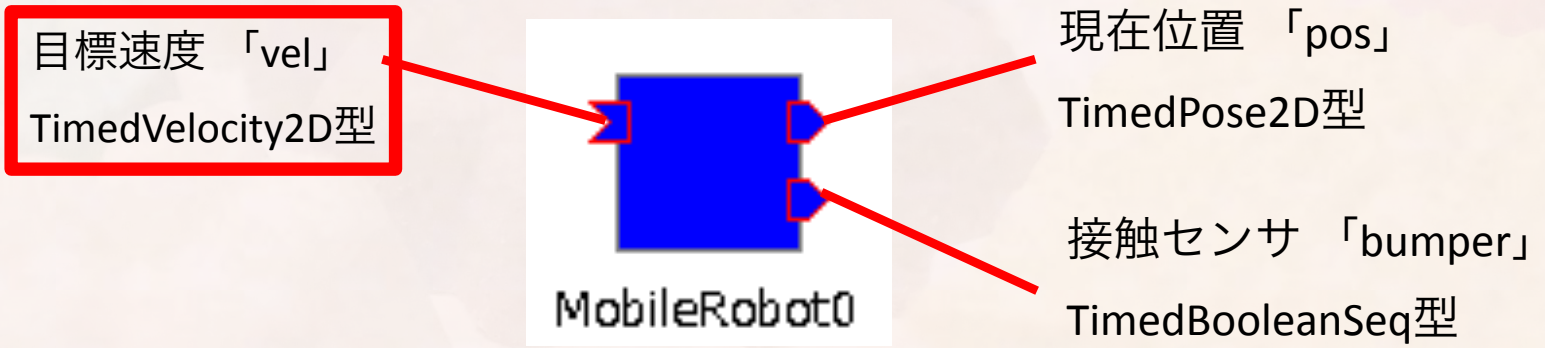
台車シミュレータの中身

- RT System Editorで表示

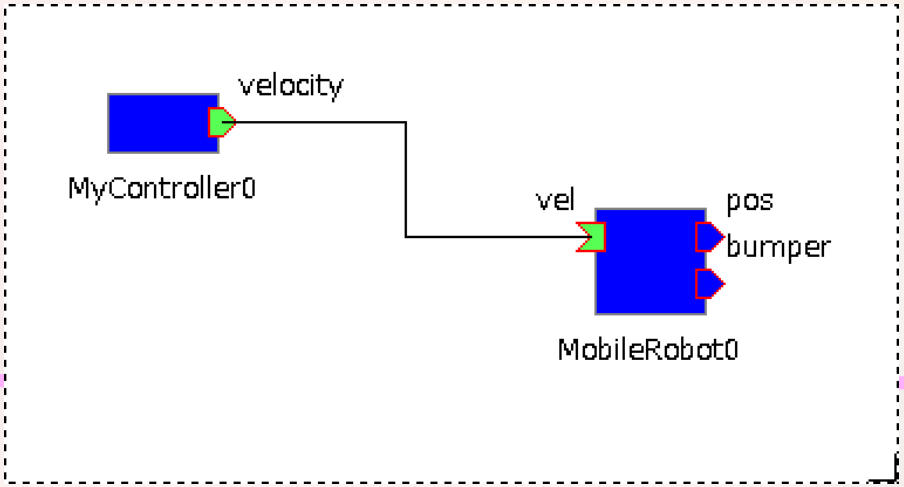


実習1. 台車に指令を送ってみよう

- 下記のMobileRobotのRTCは移動ロボットシミュレータ内の台車を表している



- 台車に直進と回転の指令を同時に送り，その場でぐるぐる回る動作をさせてみよう



RTCプログラミングの流れ

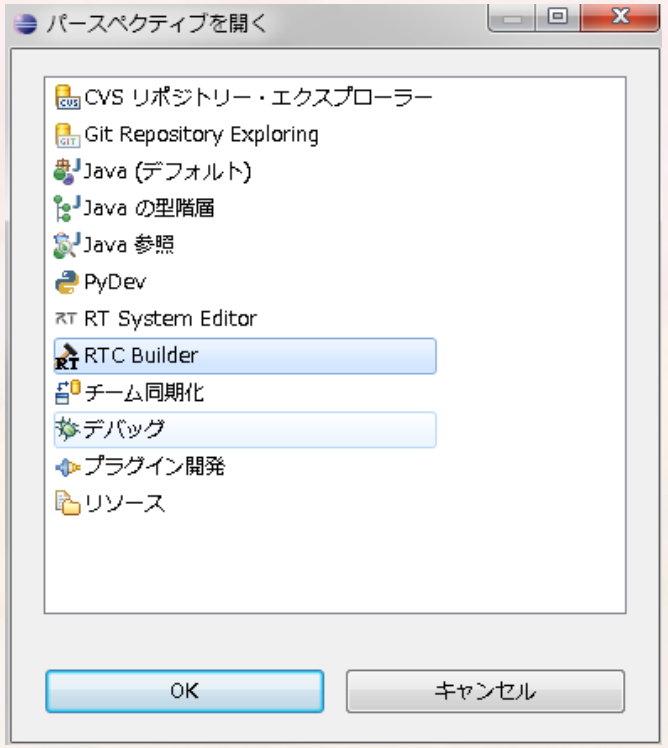
- RTC Builderによるスケルトンコードの生成

- スケルトンコードの特定のイベントハンドラに独自のコードを追加
  - on_initialized・・・CREATED->INACTIVE
  - on_activated・・・INACTIVE->ACTIVE
  - on_deactivated・・・ACTIVE->INACTIVE
  - on_execute・・・ACTIVE状態で周期的に呼ばれる

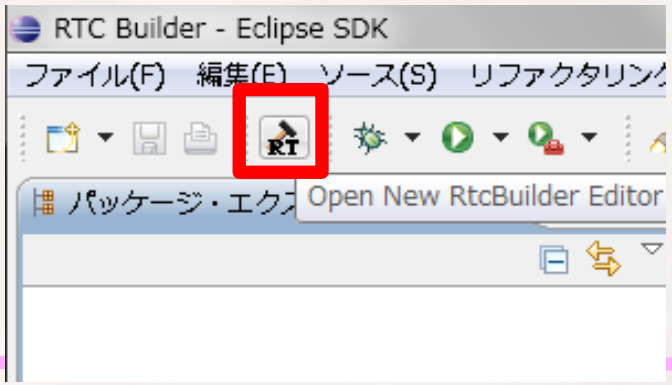
- コンパイルし実行

RTC Builder

- パースペクティブをRTC Builderに変更
  - Builder Editorで，RTCの骨格（スケルトン）コードを作成できる

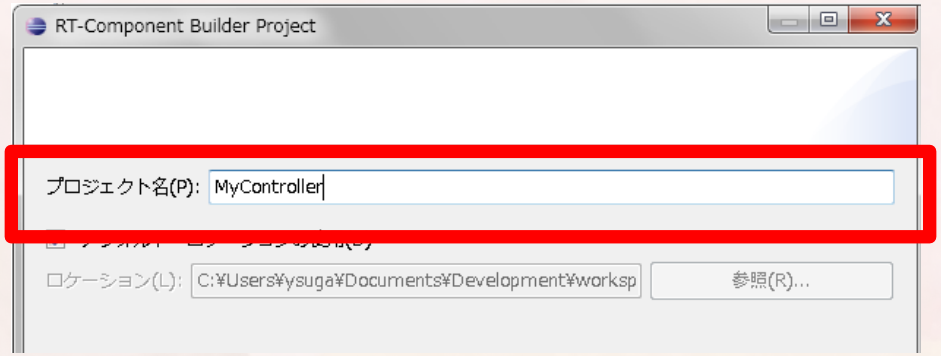


- Builder Editorを開く
  - RTCが持つべき「イベントハンドラ」が実装されているので，それを編集して自分のコードを差し込む



RTC Builder

- プロジェクト名の入力
  - MyController
- 画面下部のタブを切り替えながら必要な情報を入力

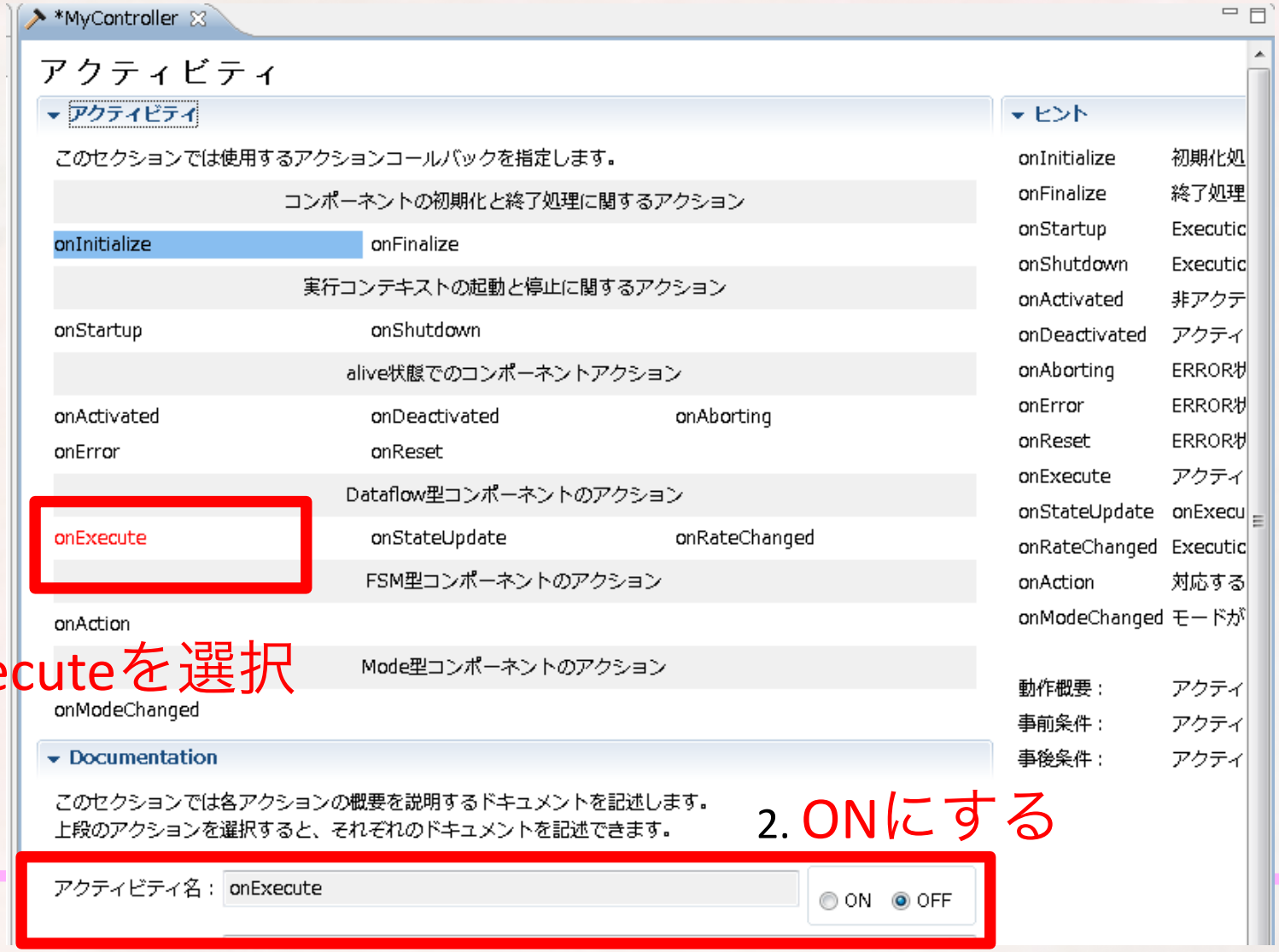


RTC Builder

- 基本タブ

RTC Builder

- アクティビティタブ

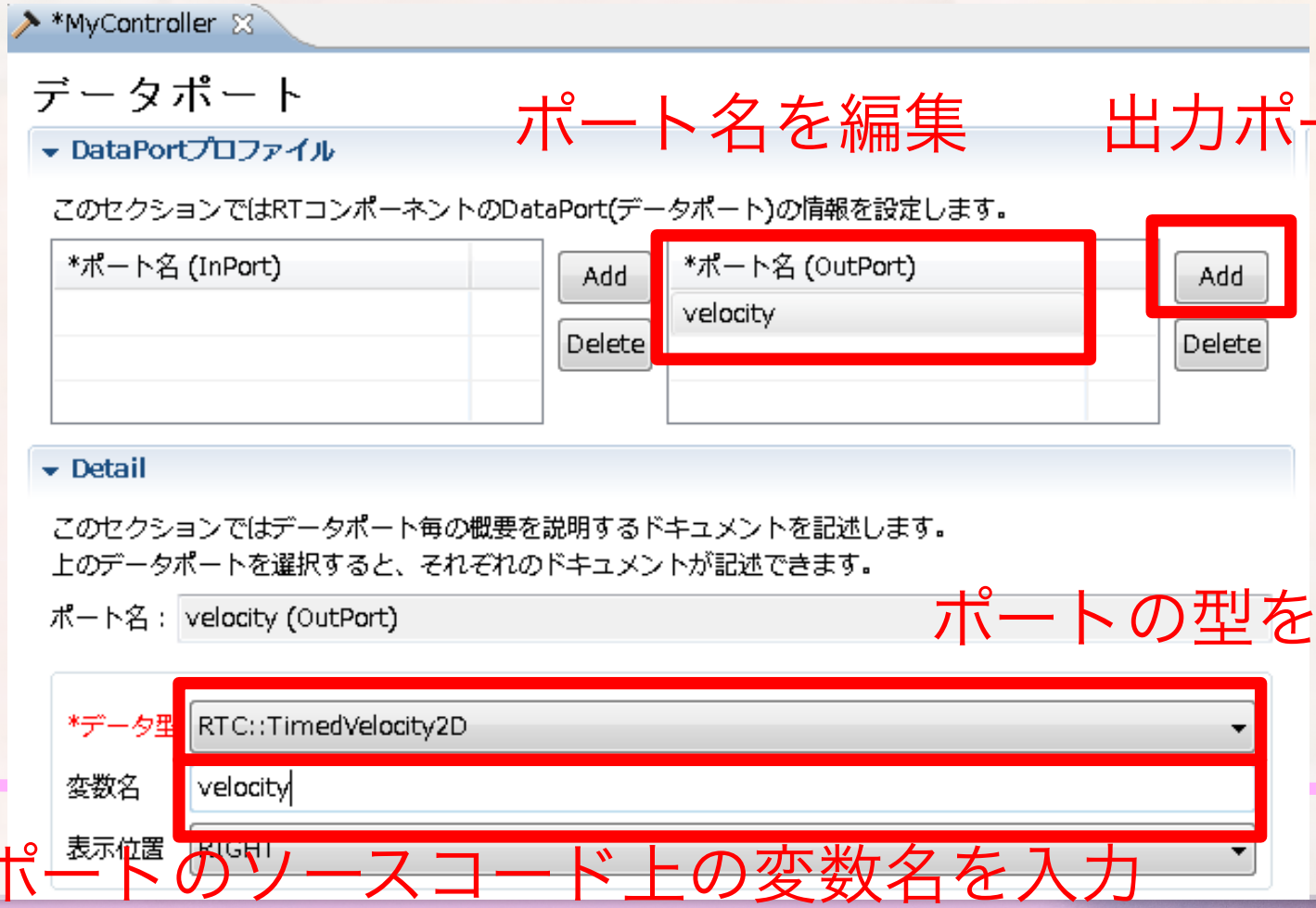


1. onExecuteを選択

2. ONにする

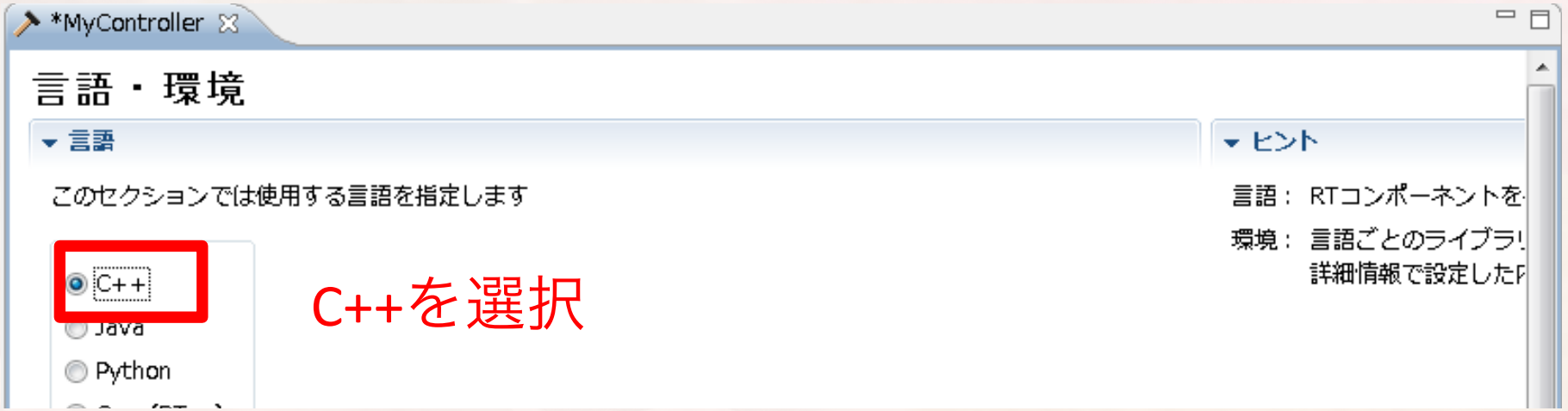
RTC Builder

- データポートタブ
  - データ型が見つから無い場合はRTM_ROOT環境変数が設定されて無い
  - システムの詳細設定や，~/.bashrcなどを確認．必要によっては再起動.
  - TimedVelocity2DはTimedVector2Dと間違いやすい！！

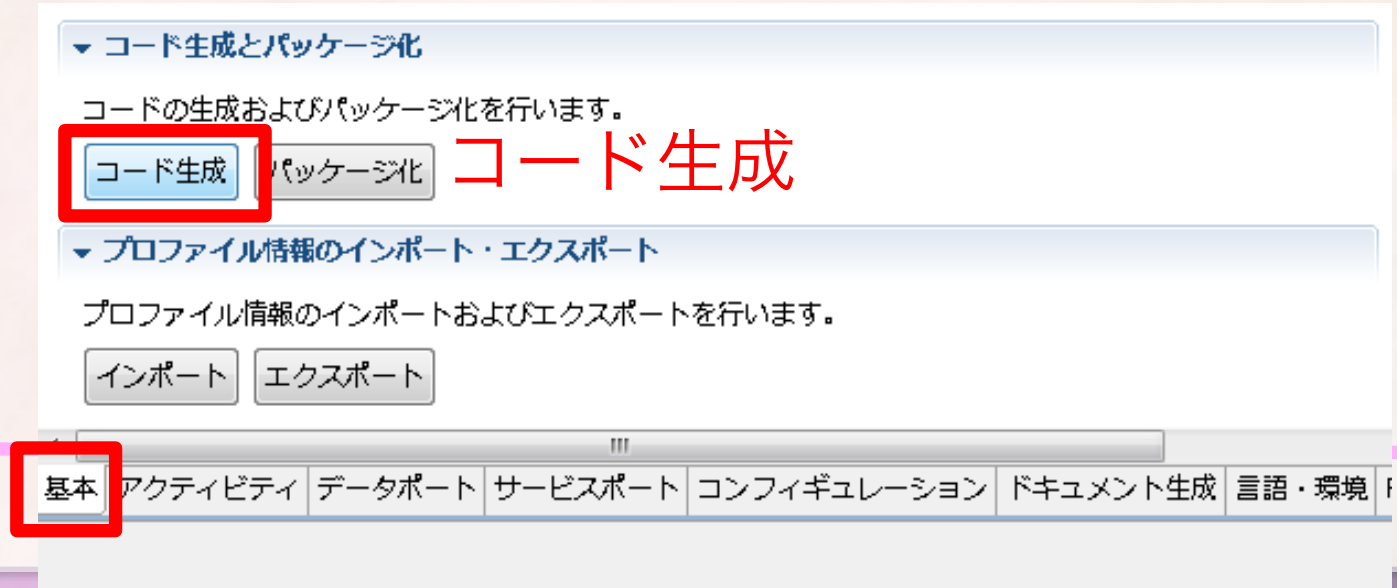


RTC Builder

- 言語タブ

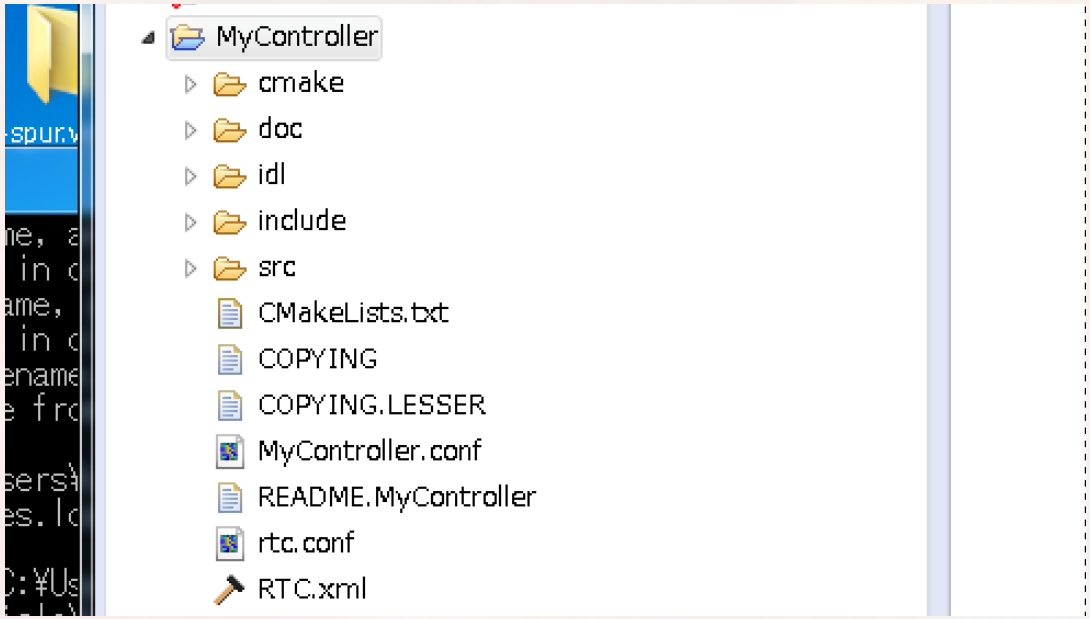


- 基本タブに戻る (下にスクロール)



RTC Builder

- MyControllerプロジェクトにソースコードが生成される
  - Builder Editor（真ん中のウィンドウ）は閉じてよい

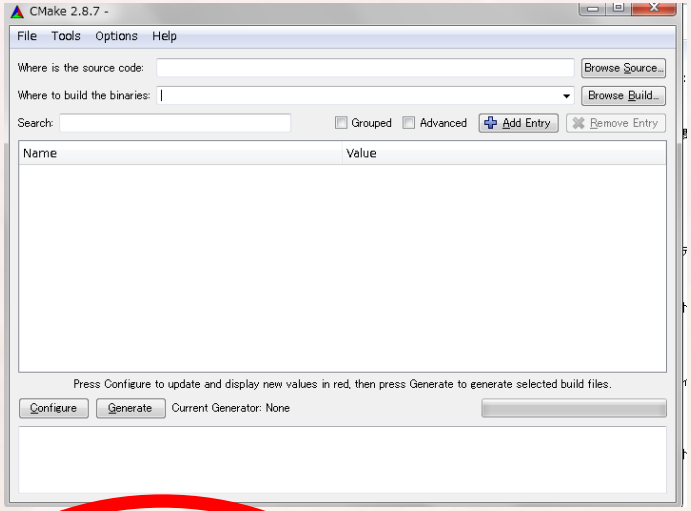


RTCのソースコードを見つける

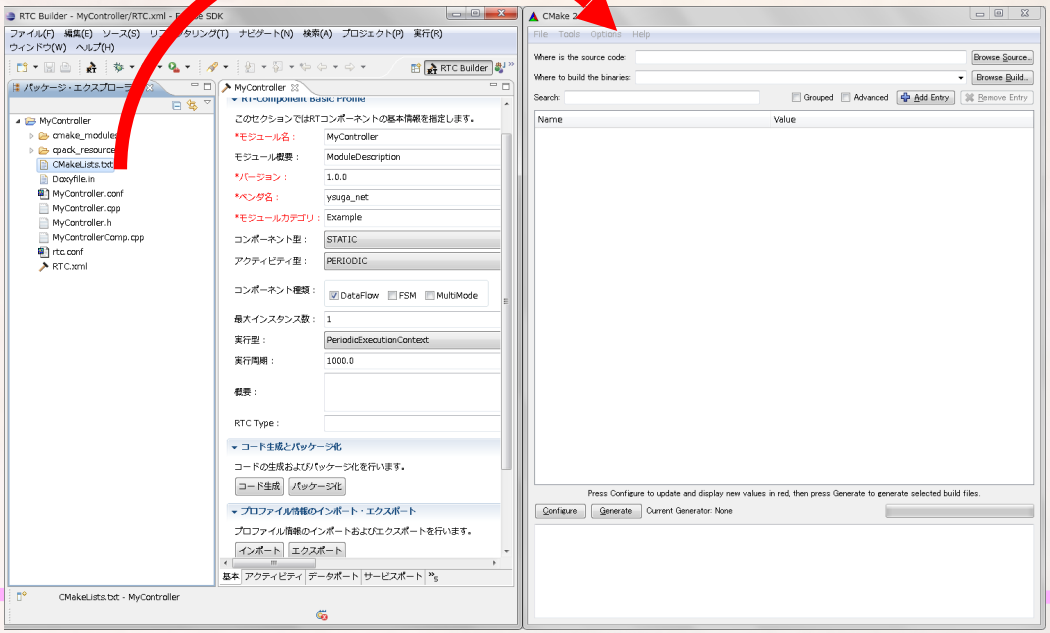
- Eclipseは通常，自分のホームディレクトリ直下のworkspaceフォルダにワークスペースを作る
  - Winなら C:\Users\ユーザ名\workspace
  - Linuxなどなら/home/ユーザ名/workspace
- ワークスペース内に，作成したRTCのソースコードを見つけることができる

CMakeによるプロジェクト生成

- CMakeを起動

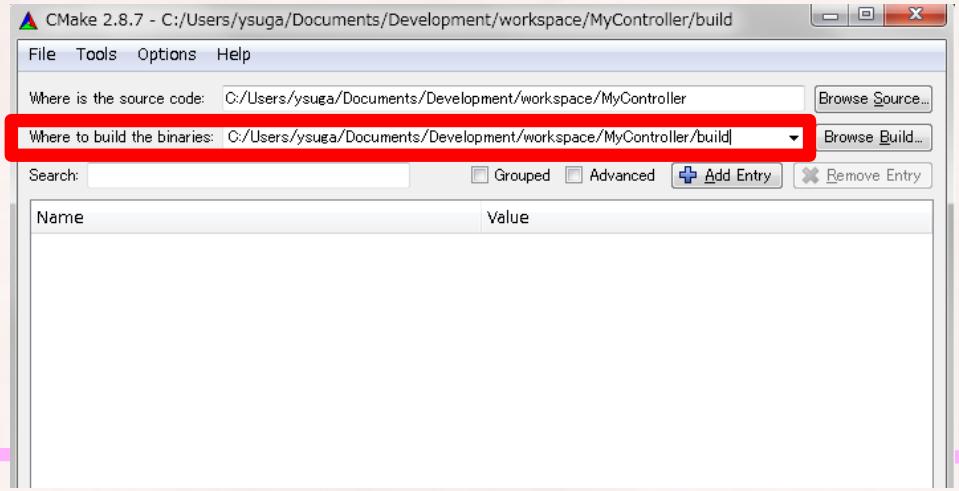


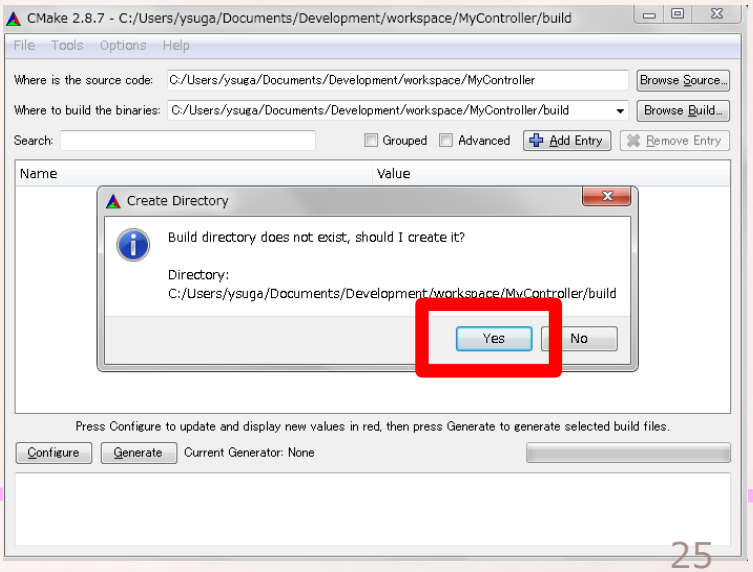
- CMakeLists.txtを
  CMakeにドラッグ
  ＆ドロップ



CMake

- 出力フォルダパスにbuildを追加
  - RTCのフォルダが”…/workspace/MyController”の場合
    - ”…/workspace/MyController/build”というフォルダを作る
- Configureを押すとフォルダ作成確認のダイアログが出るのでOK

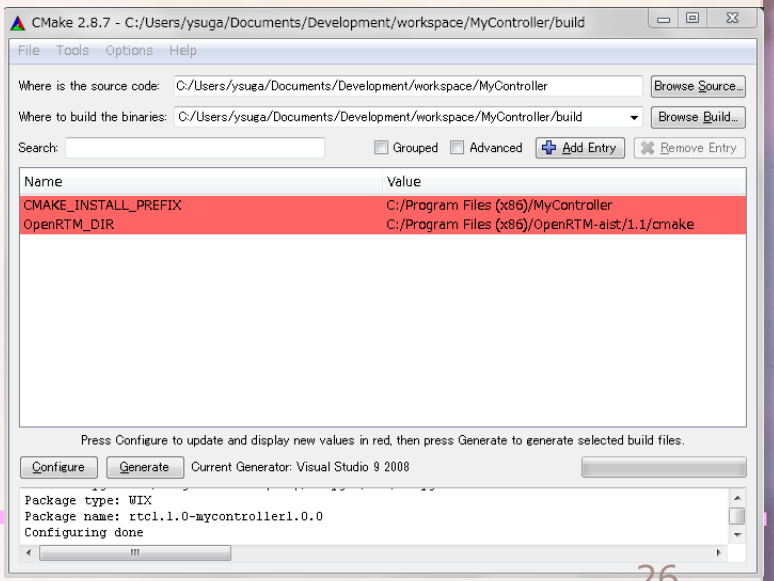




CMake2.8

- ビルド環境を選択
  - Visual Studio 12 2013を選択
  - Use default native compilersを選択

- 赤い表示が出るが恐れず「Generate」を押す
  - Visual Studio用プロジェクトファイルが生成される



MacやLinuxなら

- 以下のコマンドを打つ
  - cd $HOME/workspace/MyController # RTCのフォルダに移動
  - mkdir build　# buildディレクトリ作成
  - cd build # buildディレクトリに移動
  - cmake ../ # cmakeする
  - make #ここでコンパイル
- 古いRTCBを使っている場合，Macの場合は，RTCのヘッダーファイルを書き換える必要がある
  - RTCの名前を$ModuleNameとすると・・・
    - RTCのディレクトリ/include/$ModuleName/$ModuleName.hを開く

```
#include <rtm/Manager.h>
#include <rtm/DataFlowComponentBase.h>
#include <rtm/CorbaPort.h>
#include <rtm/DataInPort.h>
#include <rtm/DataOutPort.h>
#include <rtm/idl/BasicDataTypeSkel.h>
#include <rtm/idl/ExtendedDataTypesSkel.h>
#include <rtm/idl/InterfaceDataTypesSkel.h>
```

```
#include <rtm/idl/BasicDataTypeSkel.h>
#include <rtm/idl/ExtendedDataTypesSkel.h>
#include <rtm/idl/InterfaceDataTypesSkel.h>
#include <rtm/Manager.h>
#include <rtm/DataFlowComponentBase.h>
#include <rtm/CorbaPort.h>
#include <rtm/DataInPort.h>
#include <rtm/DataOutPort.h>
```

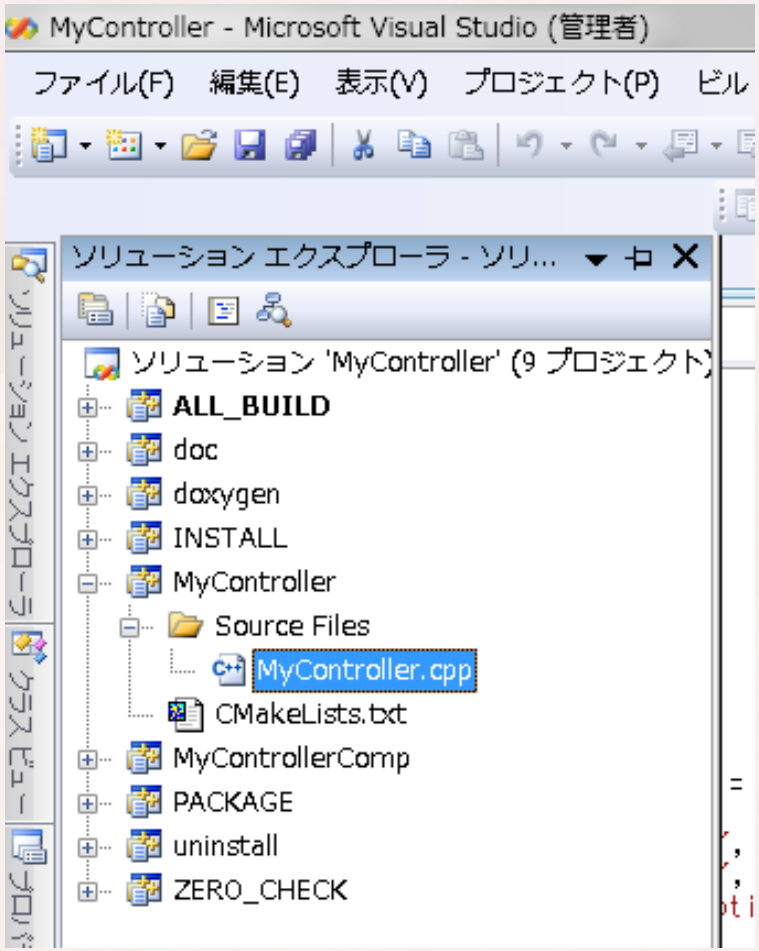
Visual Studio

- build/src/MyController.slnファイルをダブルクリック
  - Visual Studioが起動する

– 複数のプロジェクトが存在

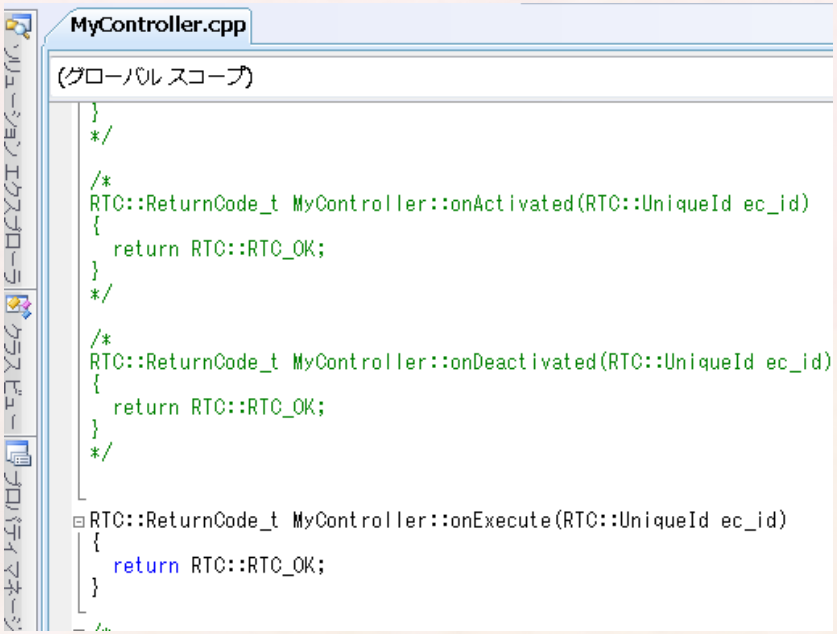
  - MyController・・RTC本体
  - MyControllerComp・・・RTCを単体のアプリケーションとして実行するためのプロジェクト

- MyControllerプロジェクト
  - MyController.cppを開く



Visual Studio

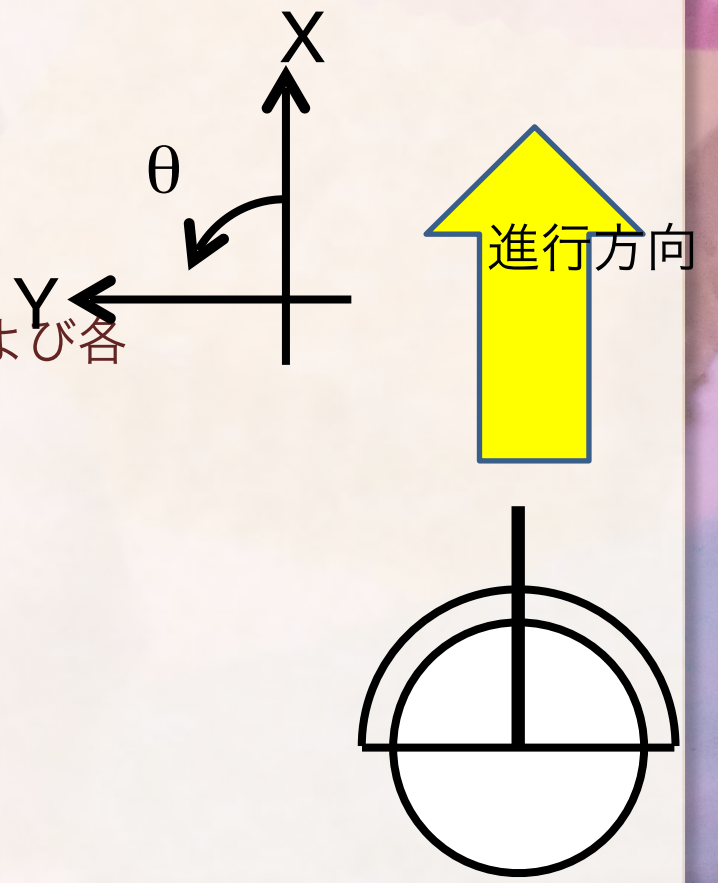
- MyController::onExecute関数
  - RTCがACTIVE状態の場合に周期的に呼ばれる
  - 実行周期は後で設定
  - ここに周期的に呼ばれるべき制御アルゴリズム等を記述




```
RTC::ReturnCode_t MyController::onExecute(RTC::UniqueId ec_id)
{
        m_velocity.data.vx = 0.05; // バッファに書込む
        m_velocity.data.vy = 0;
        m_velocity.data.va = 1.0;
        m_velocityOut.write();    // データを送信
        return RTC::RTC_OK;
}
```

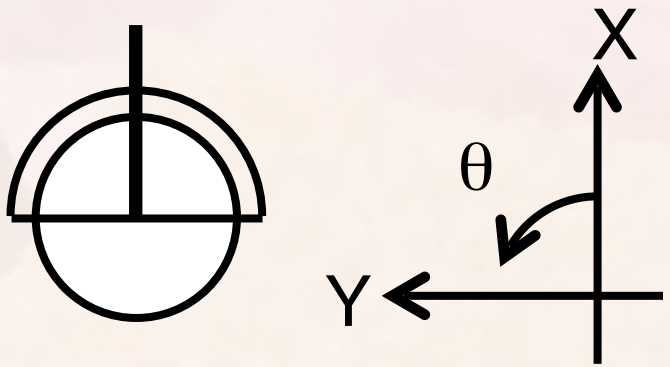
2次元速度のデータ型

- TimedVelocity2D・・・タイムスタンプ付き2次元平面での速度および角速度
  - tm : Time型・・・タイムスタンプ
    - sec : unsigned long型・・・秒
    - nsec : unsigned long型・・・ナノ秒
  - data : Velocity2D型・・・２次元平面での速度および各速度
    - vx : double型・・・X軸方向速度 (単位m/s)
    - vy : double型・・・Y軸方向速度 (単位m/s)
    - va : double型・・・各速度 (単位rad/s)



移動ロボットに指令を送る

- 従って，TimedVelocity2D型の変数「vel」の各メンバにアクセスする際は・・・
  - vel.data.vx・・・X軸方向速度 [m/s]
  - vel.data.vy・・・Y軸方向速度 [m/s]
  - vel.data.va・・・Z軸方向速度 [rad/s]
- データ型がどんな構造体なのか知るには，IDLファイルを読む
  - 通常は，C:\Program Files (x86)\OpenRTM-aist\1.1\rtm\idlにある
  - TimedVelocity2Dは，ExtendedDatatype.idlで定義されている．



LinuxやMacなら，

/usr/include/openrtm-1.1/rtm/idl

もしくは，

/usr/local/include/openrtm-1.1/rtm/idl

IDLファイル

- オブジェクト指向言語に変換可能なインターフェース定義言語
- C++, Java, Pythonに変換
- 独自のデータ型を定義するにはIDLを書いてRTC Builderに読み込ませる
- CORBAの入門書やウェブサイトを参考にしてください.

```
/*!
 * @struct TimedVelocity2D
 * @brief Time-stamped version of Velocity2D.
 */
struct TimedVelocity2D
{
  Time tm;
  Velocity2D data;
};
```

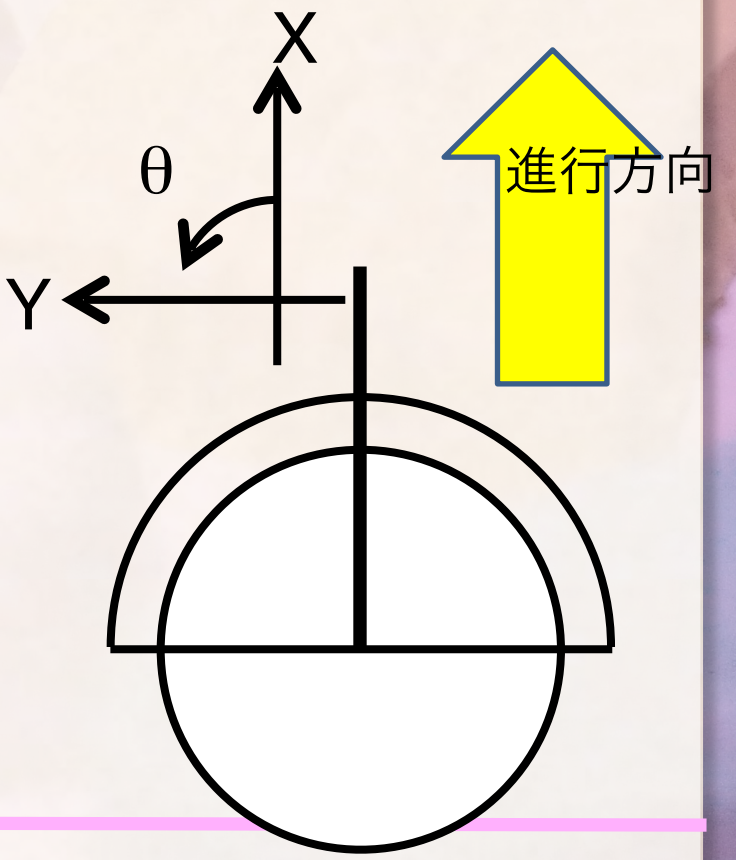
```
/*!
 * @struct Velocity2D
 * @brief Velocities in 2D cartesian space.
 */
struct Velocity2D
{
  /// Velocity along the x axis in metres per second.
  double vx;
  /// Velocity along the y axis in metres per second.
  double vy;
  /// Yaw velocity in radians per second.
  double va;
};
```

実行

- ネームサービスの起動確認
- RTCの実行
  - MyControllerCompを右クリック
  - 「デバッグ→インスタンス作成」で実行
- MobileRobotSim.batを実行するとシミュレータのみ起動
- RT System Editorでの接続，ACTIVE化

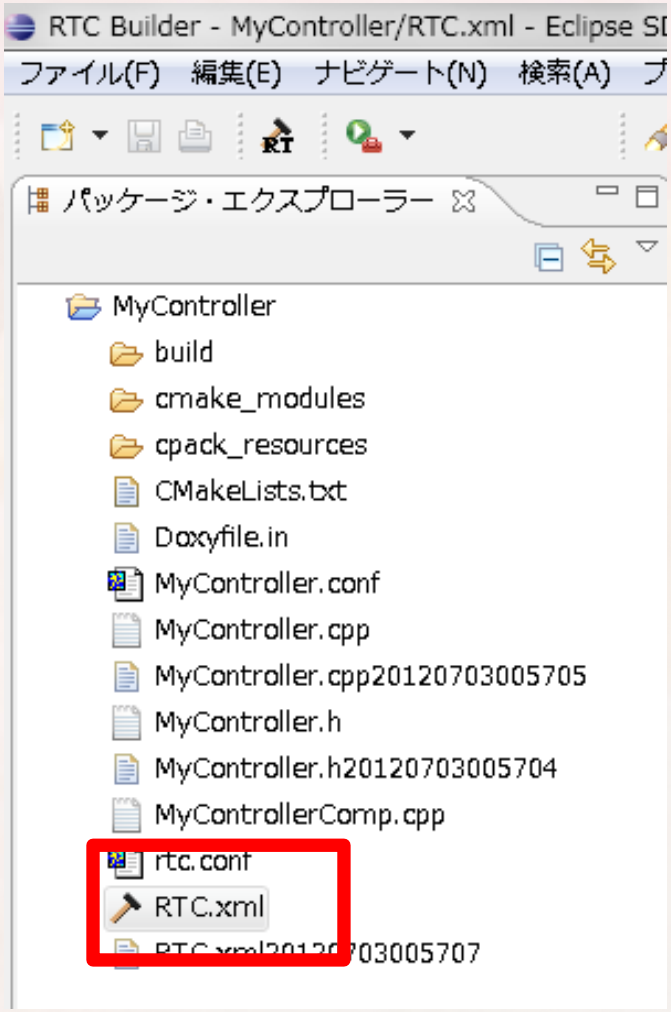
実習2. 台車の動きを調整

- コンフィグレーション機能を試す
  - コンフィグレーションとは，実行中のRTCを調整するための機能
    - 例：制御ゲインの調整
- コンフィグレーション機能を加えて実行中にMyControllerの機能を調整する



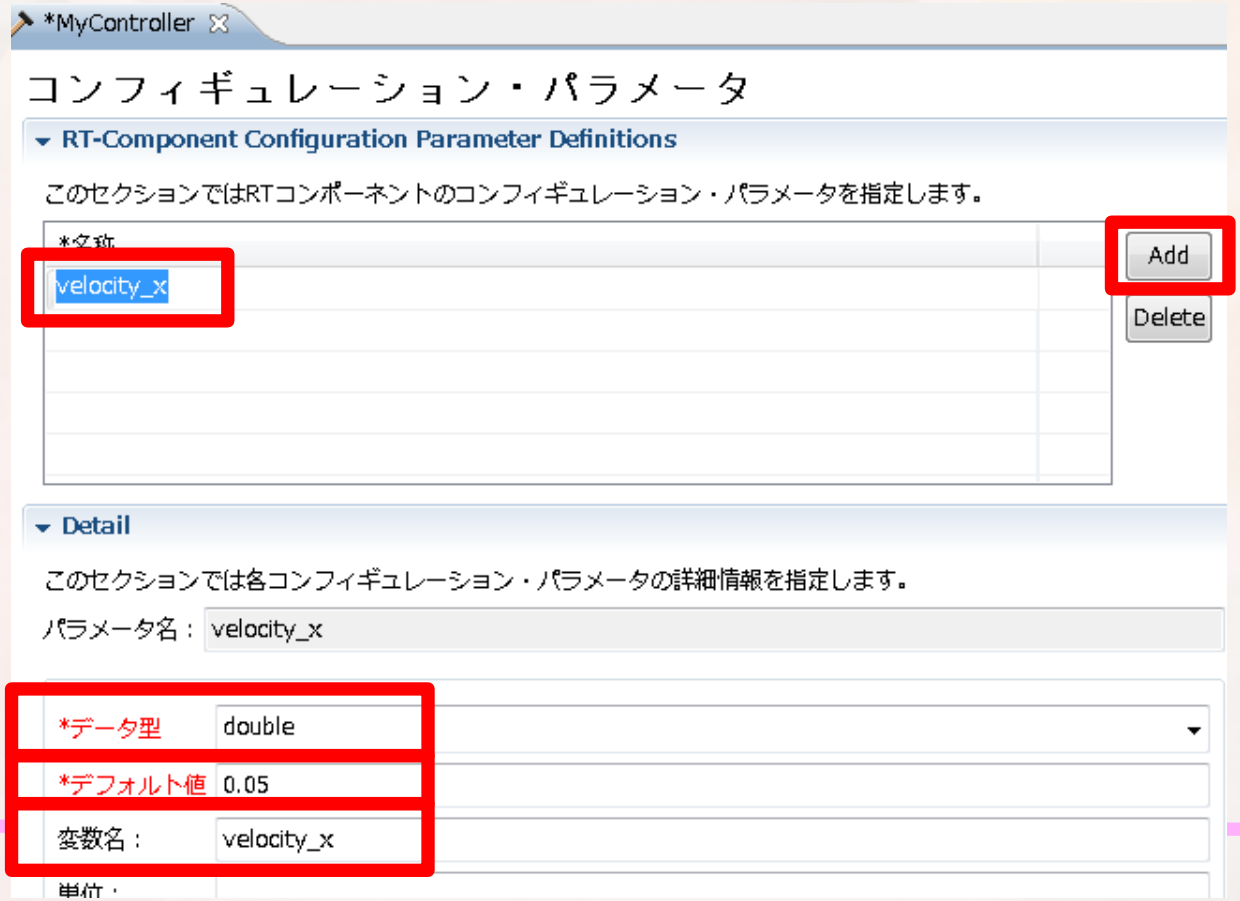
RTC Builder

- Eclipseのパースペクティブを RTC Builderに再変更
- パッケージエクスプローラの RTC.xmlファイルをダブルクリックする
  - 先ほど作成した情報が記録されている



RTC Builder

- コンフィグレーションタブ
  - velocity_x というコンフィグレーションを追加

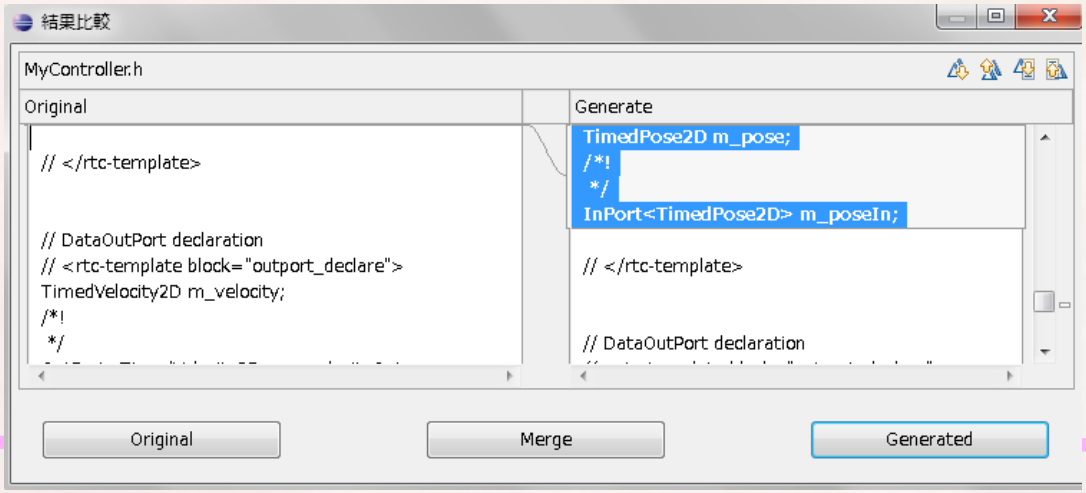


追加したコンフィグレーション

- velocity_x : double型
  - 変数名: velocity_x
  - デフォルト値: 0.05
- velocity_theta : double型
  - 変数名: velocity_theta
  - デフォルト値: 1.0

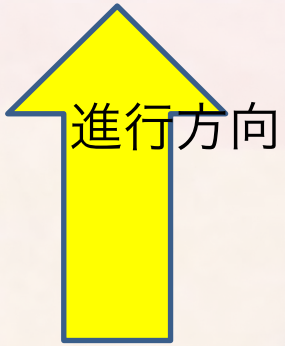
RTC Builder

- 比較ダイアログ
  - ここでは「Generated」を選択
  - 基本的に「Merge」を選択
    - 新しいコードの変更点のみ反映
  - Generatedは新しいコード側で上書きされるので，自分の記入したコードが消える
    - バックアップファイルがある (ファイル名末尾に日付が入る) ので心配要りません

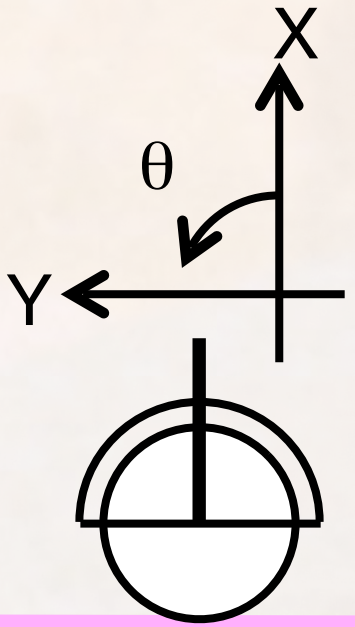


実習2. 台車の動きを調整

- コンフィグレーション機能を試す
  - コンフィグレーション機能を加えて実行中にMyControllerの機能を調整する

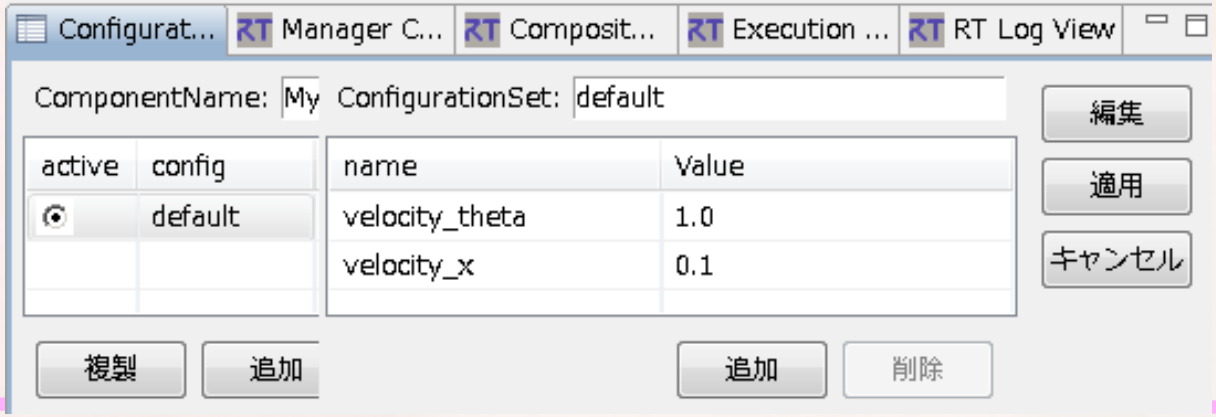



```
RTC::ReturnCode_t MyController::onExecute(RTC::UniqueId ec_id)
{
        std::cout << "Vx=" << m_velocity_x  << std::endl;
        std::cout << "Vtheta=" << m_velocity_theta << std::endl;
        m_velocity.data.vx = m_velocity_x;
        m_velocity.data.vy = 0;
        m_velocity.data.va = m_velocity_theta;
        m_velocityOut.write();
  return RTC::RTC_OK;
}
```



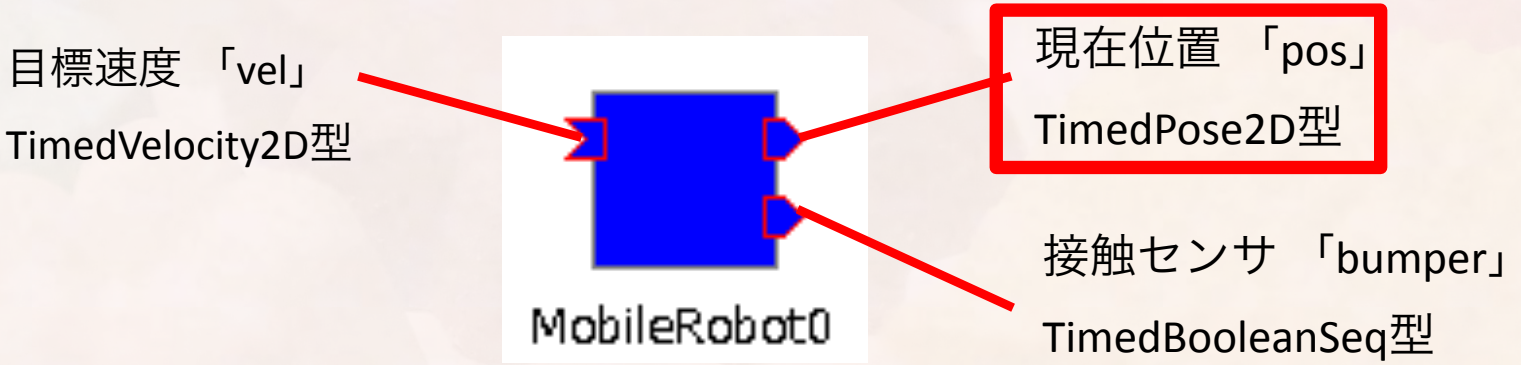
実行

- ネームサービスの起動確認
- RTCの実行
  - MobileRobotSim.batを実行するとシミュレータのみ起動
- RT System Editorでの接続，ACTIVE化
- RT System Editor下部にConfiguration Viewに選択中のRTCのコンフィグが表示されるので変更して「適用」を選択する（適用しないと反映されない）

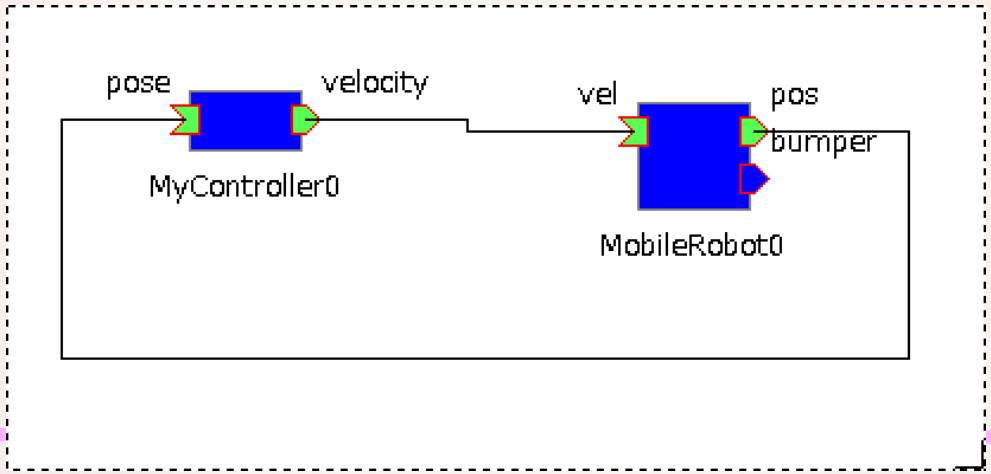


実習3. 位置の取得

- 現在位置をデータポートから受け取ってコンソールに表示する

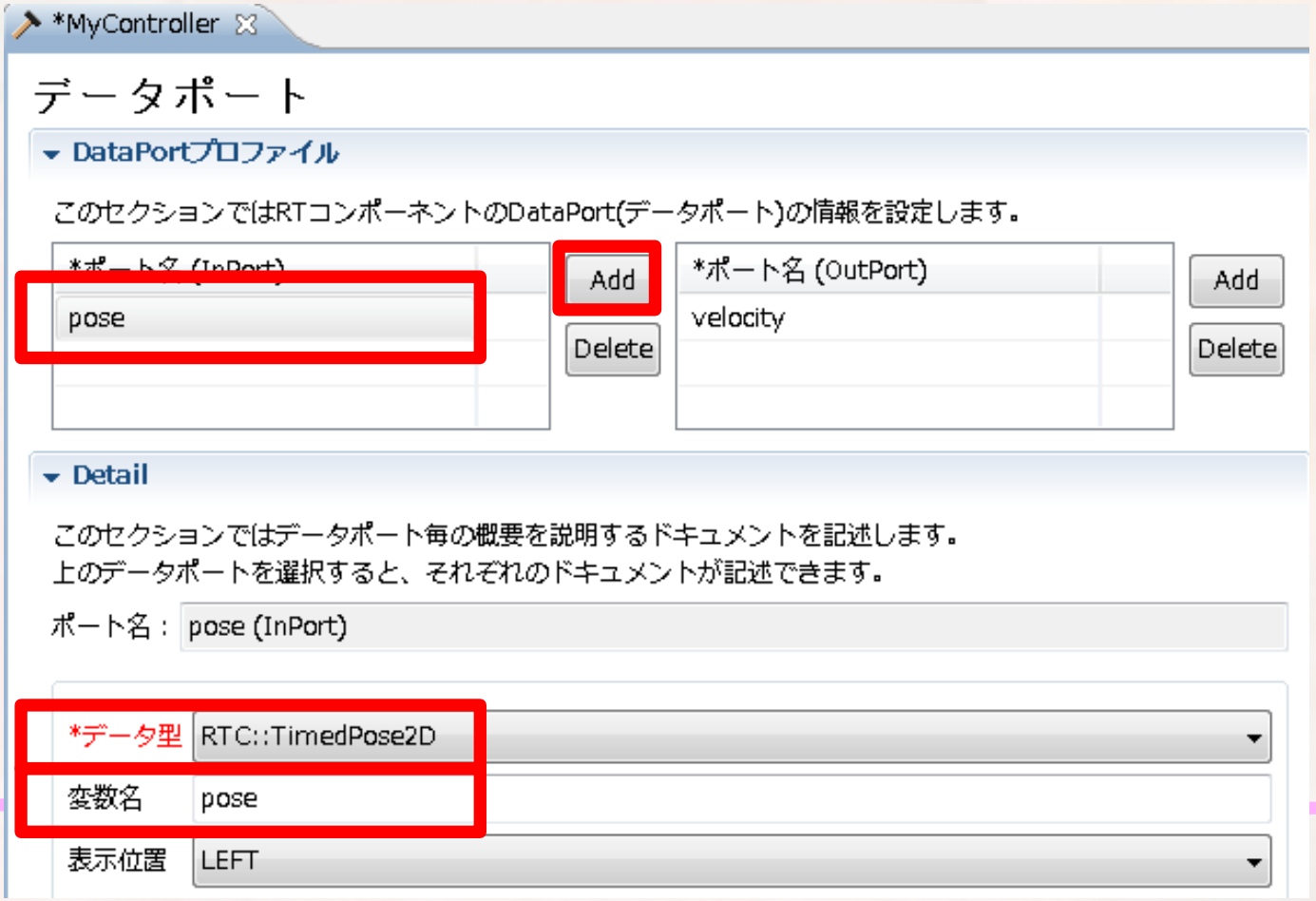


- TimedPose2D型の入力ポートをMyControllerに追加する



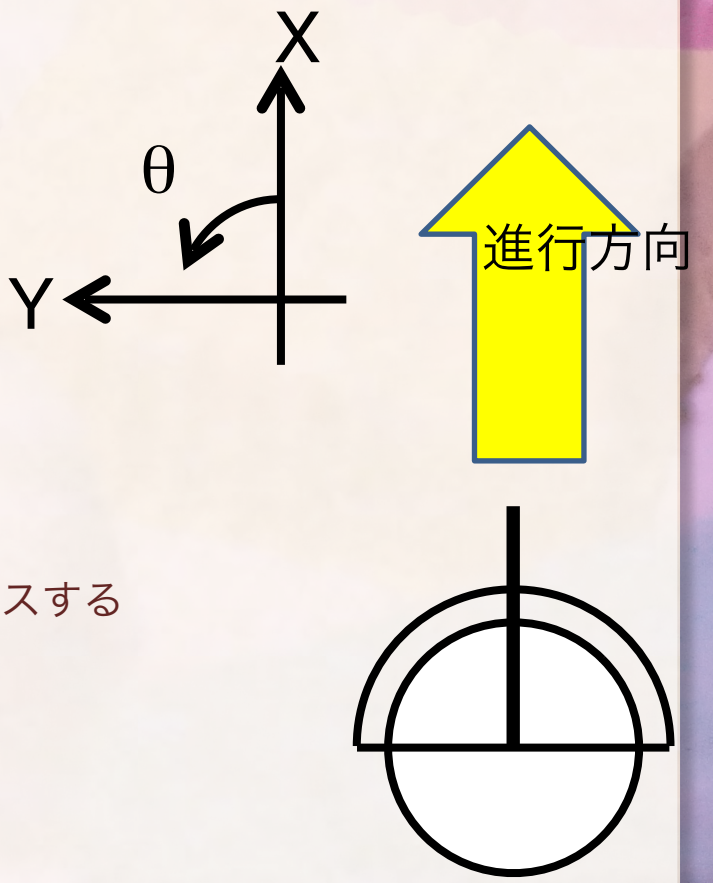
RTC Builder

- 再度，RTCBuilderを開く
  - MyController / RTC.xmlを開く (たぶん，開きっぱなし？)
  - データポートタブに移動 → TimedPose2D型の入力ポート「pose」を追加



2次元位置のデータ型

- TimedPose2D・・・2次元平面での位置および姿勢
  - tm : Time型・・・タイムスタンプ
    - sec : unsigned long型・・・秒
    - nsec : unsigned long型・・・ナノ秒
  - data : Pose2D型・・・２次元平面での位置および姿勢
    - position : Point2D型・・・２次元平面内での位置
      - x : double型・・・X軸方向変異 (単位m)
      - y : double型・・・Y軸方向変異 (単位m)
    - heading : double型・・・姿勢 (単位rad)
- 従って，C++でTimedPose2D型の変数「pose」の各メンバにアクセスする際は・・・
  - pose.data.position.x・・・X軸方向変位
  - pose.data.position.y・・・Y軸方向変位
  - pose.data.heading・・・Z軸方向回転



Visual Studio

- 再度，onExecute関数を編集
  - 入力ポートはデータが来ているか確認する処理が入る

```
RTC::ReturnCode_t MyController::onExecute(RTC::UniqueId ec_id)
{
        m_velocity.data.vx = m_velocity_x;
        m_velocity.data.vy = 0;
        m_velocity.data.va = m_velocity_theta;
        m_velocityOut.write();

        if(m_poseIn.isNew()) {      // 入力ポートに入力があるか確認
                m_poseIn.read();            // 入力があるならば読み込む
                std::cout << "X = " << m_pose.data.position.x << std::endl;
                std::cout << "Y = " << m_pose.data.position.y << std::endl;
                std::cout << "Z = " << m_pose.data.heading << std::endl;
        }
        return RTC::RTC_OK;
}
```

実行

- ネームサービスの起動確認
- RTCの実行
  - MobileRobotSim.batを実行するとシミュレータのみ起動
- RT System Editorでの接続，ACTIVE化
- 実行時の周期が速すぎる？

rtc.conf

- RTCの実行時の設定ファイル
  - ネームサーバのIPアドレス，ポート番号
  - 実行周期，実行コンテキストの種類
  - RTCの名前付け規則
  - ログの有無，ログレベル
  - etc…
- rtc.confで実行周期を変更（単位Hz）
  - 右クリック→アプリケーション→テキストエディタ

```
exec_cxt.periodic.rate: 1.0
```

- rtc.confを実行ファイルと同じ場所に置いて，実行ファイルを実行
  - Visual C++でデバッグする場合は，プロジェクトファイルと同じディレクトリに置く（デフォルトでそこがカレントディレクトリになる）

データポート関連変数の命名法則

- 出力ポートの場合
  - 「変数名」＝「example」
    - m_example
      - データポートのデータを入れるバッファ
    - m_exampleOut
      - データポート本体
- 利用方法
  - 1. m_example　にデータを入力
  - 2. m_exampleOut.write()

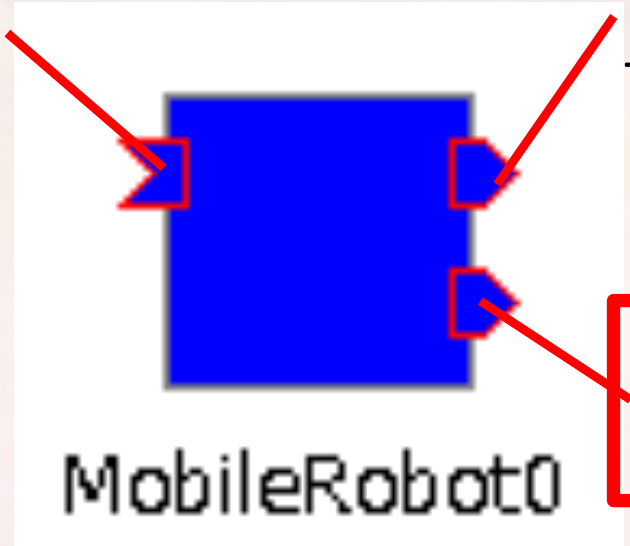
- 入力ポートの場合
  - 「変数名」＝「example」
    - m_example
      - データポートのデータを入れるバッファ
    - m_exampleIn
      - データポート本体
- 利用方法
  - 1. m_exampleIn.isNew()で受信確認
  - 2. m_exampleIn.read()でデータ取得
  - 2. m_exampleのデータを読み取る

実習4. 台車の接触スイッチ

- TimedBooleanSeq　真偽型 (True/False)の配列

目標速度 「vel」

TimedVelocity2D型



現在位置 「pos」

TimedPose2D型

接触センサ 「bumper」

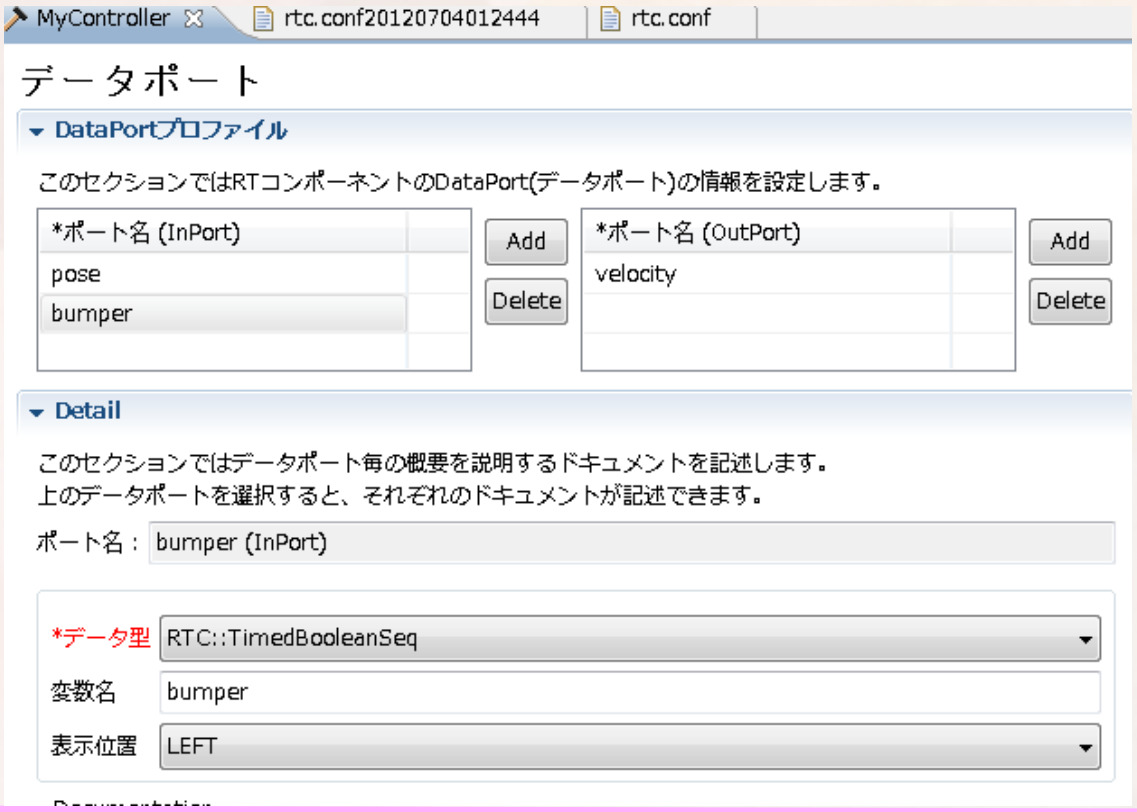
TimedBooleanSeq型

配列の0番目要素＝右側スイッチ

配列の1番目要素＝左側スイッチ

RTC Builderによるポートの追加

- InPort : TimedBooleanSeq型
  - ポート名: bumper
  - 変数名： bumper



Visual Studio

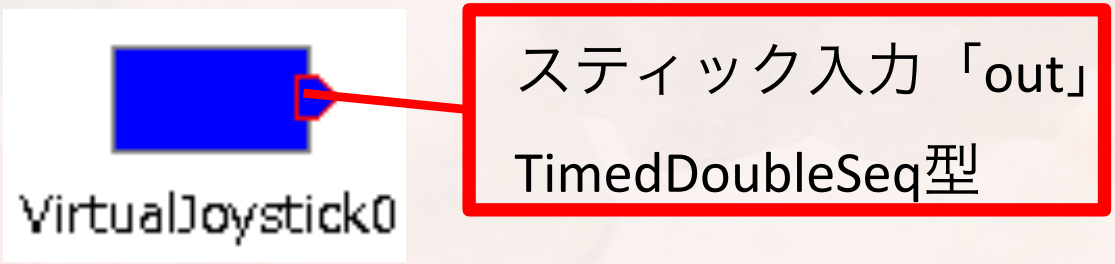
- **Seq型はdataメンバを配列のように使える

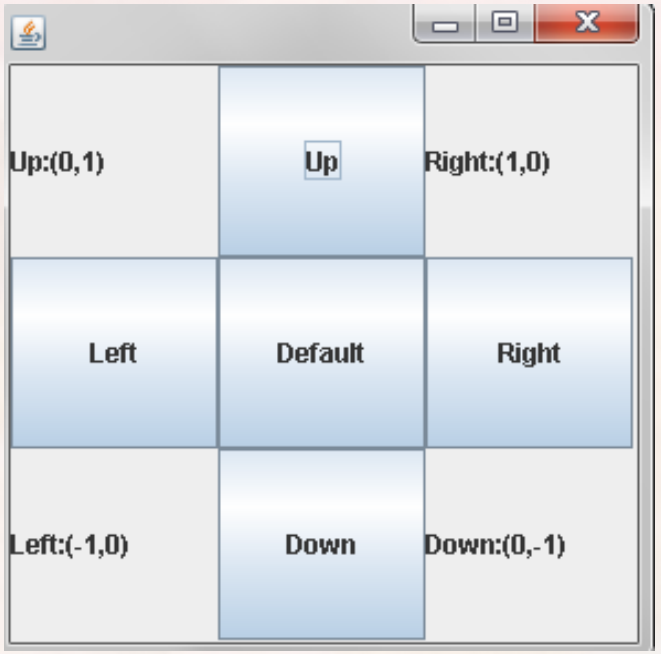
```
RTC::ReturnCode_t MyController::onExecute(RTC::UniqueId ec_id)
{
        ・・・省略・・・


        if(m_bumperIn.isNew()) {
                m_bumperIn.read();
                if(m_bumper.data[0] == true) {
                        std::cout << "Right Bumper Hit!!" << std::endl;
                }
                if(m_bumper.data[1]  == true) {
                        std::cout << "Left  Bumper Hit!!" << std::endl;
                }
        }
        return RTC::RTC_OK;
}
```

みなさんへの課題

- 接触スイッチの反応でロボットの動作を変える
  - 左右どちらかに旋回する？
- 位置の値を使って図形を描く
  - 正方形，星型などなど
- 仮想ジョイスティックを使った操作





各ボタンの出力値はGUIに記載されている

例： Up(0,1) → 0番目要素が0, 1番目要素が1

最初はコンソールにどんなデータが来るか表示して試してみよう

- お使いのロボット関連製品をRTC化する
  - onExecute以外の使えるイベントハンドラ例(RTC Builderで使えるように設定)
    - onActivated・・・Activate時に一回呼ばれる（初期化用）
    - onDeactivated・・・Deactivate時もしくはエラー状態遷移時に呼ばれる（終了処理）

まとめ

- ツールの使い方
  - RT System Editor (RTCの接続, Activate/Deactivate)
  - RTC Builder (RTCのスケルトンコード生成)
  - CMake (Visual C++用プロジェクト生成)
- コーディング方法
  - データポート入出力
    - TimedVelocity2D, TimedPose2D, TimedBooleanSeq
  - コンフィグレーション
  - rtc.confの設定

さらなる発展

- RTCを探してみよう
  - http://openrtm.org >プロジェクト > RTコンポーネント
- 独自のデータ型を使ってみよう
  - データポートに独自のデータ型を使うことができる
- サービスポートを使ってみよう
  - RTCに関数型のインターフェースを追加でき，より柔軟なシステム設計ができる
- rtshellを使ってみよう
  - コマンドラインから使えるツール．スクリプトを書くことで開発を効率化できる
- RTCを接続・コンフィグ・アクティブ化するプログラムを作ってみよう
  - システム構築を自動化できる
- 参考ページ： http://ysuga.net

ご清聴ありがとうございました

株式会社SUGAR SWEET ROBOTICS

菅　佑樹

@ysuga (RTMのこととかも呟きます)

ysuga@sugarsweetrobotics.com

http://sugarsweetrobotics.com

http://ysuga.net