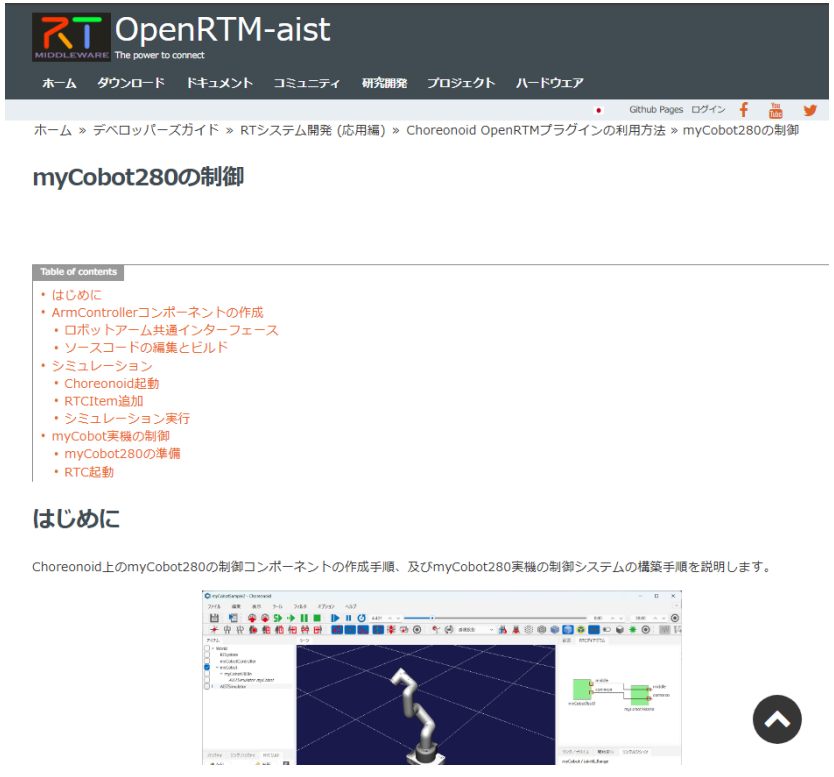
産総研
ともに挑む。つぎを創る。

サービスポート入門

国立研究開発法人産業技術総合研究所
インテリジェントシステム研究部門
宮本信彦

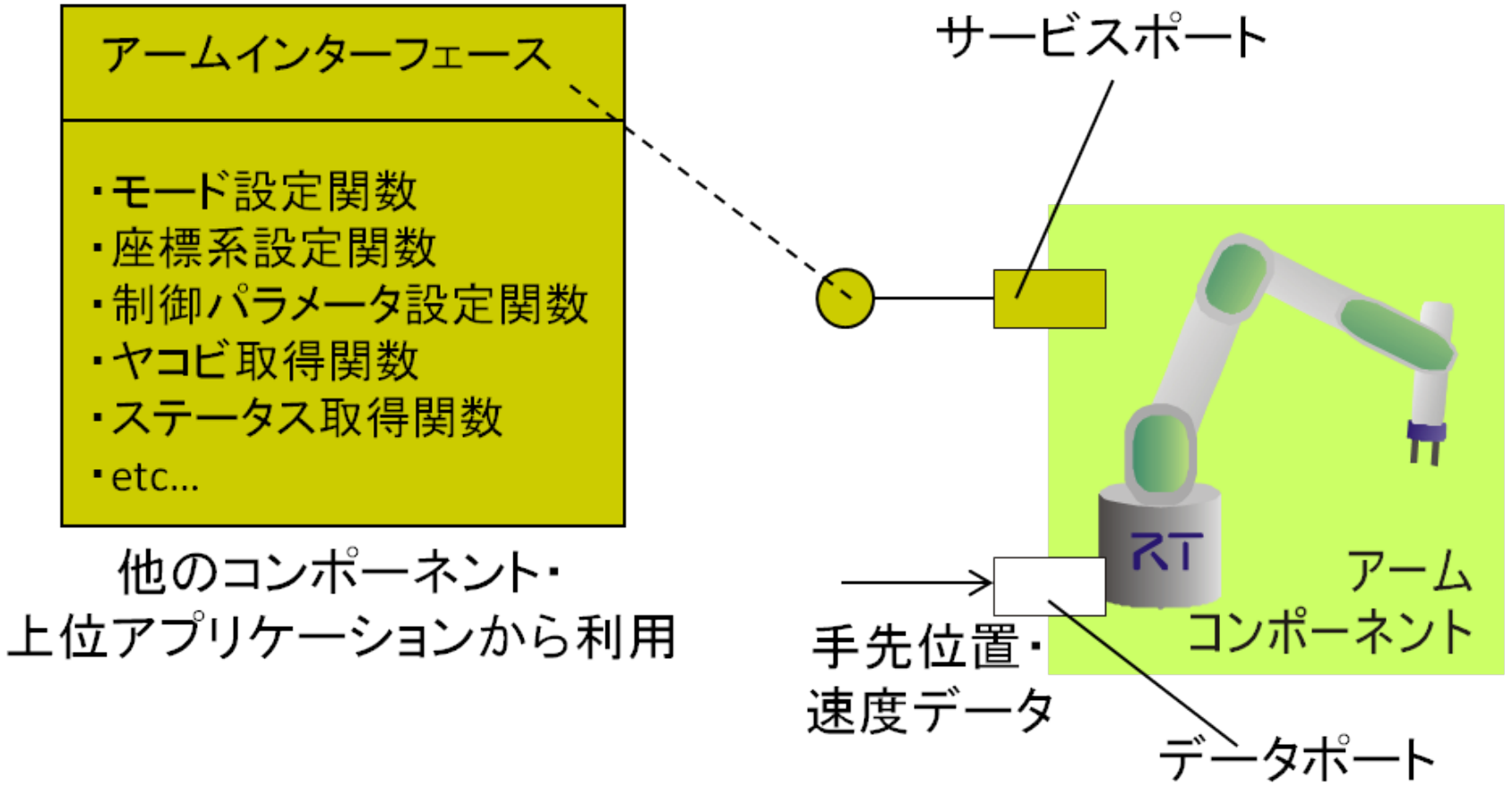
- 配布資料の「 WEBpage 」のHTMLファイルを開く
  - myCobot280の制御 _ OpenRTM-aist.html

- もしくは以下のWebページ
  - https://openrtm.org/openrtm/ja/node/7339



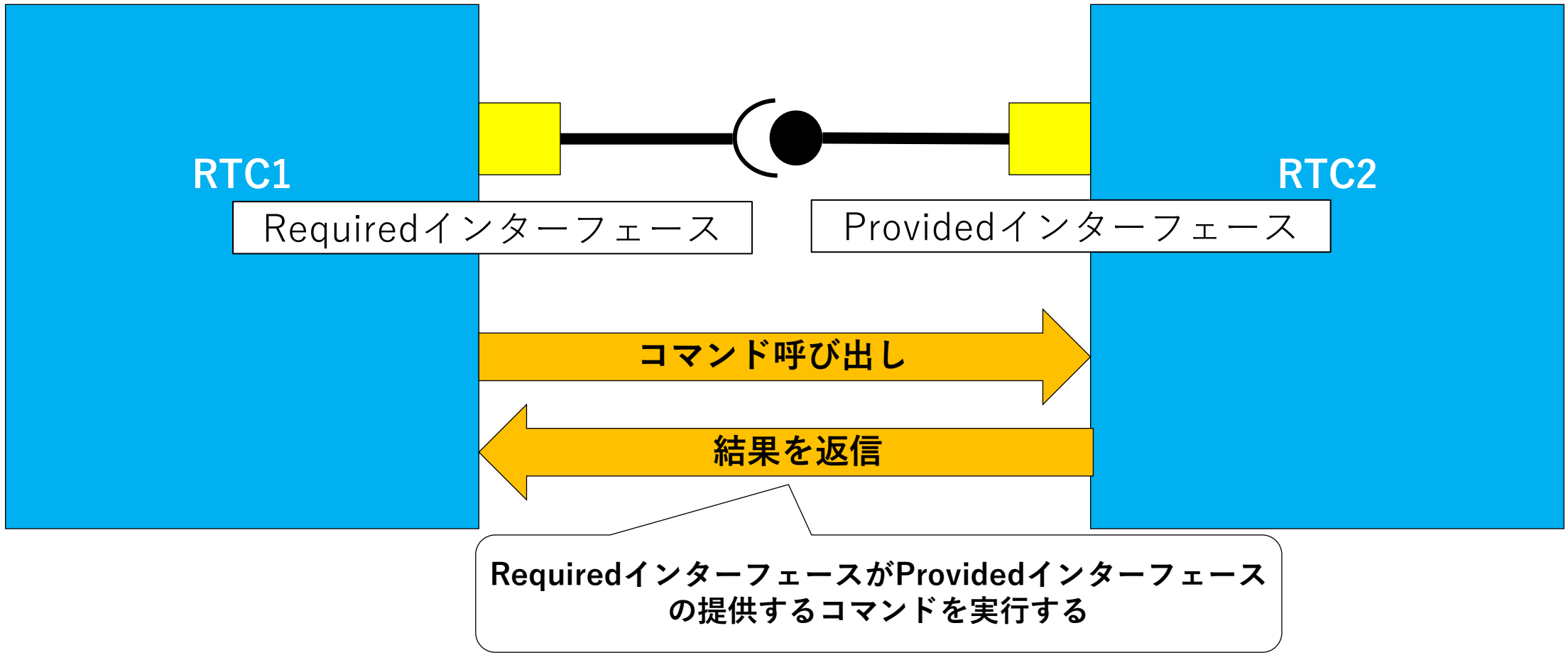
サービスポート

- サービスポートは、コマンドレベルのコンポーネント間通信の仕組みを提供
  - 複数のパラメータを送信、処理結果の取得、処理完了まで待機等



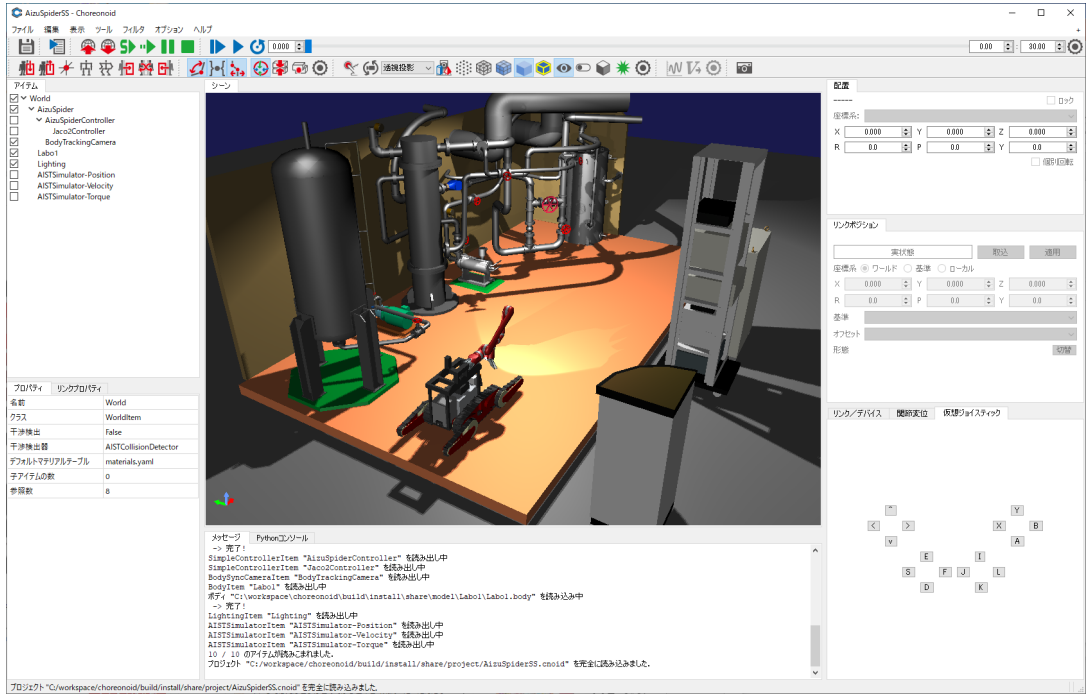
サービスポート

- サービスポートは、サービスインターフェースを持つ
  - 機能を提供するProvidedインターフェースと機能を利用するRequiredインターフェースを接続する

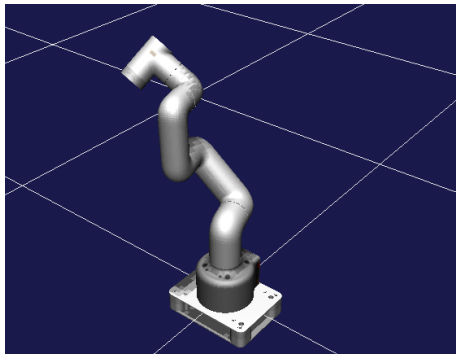


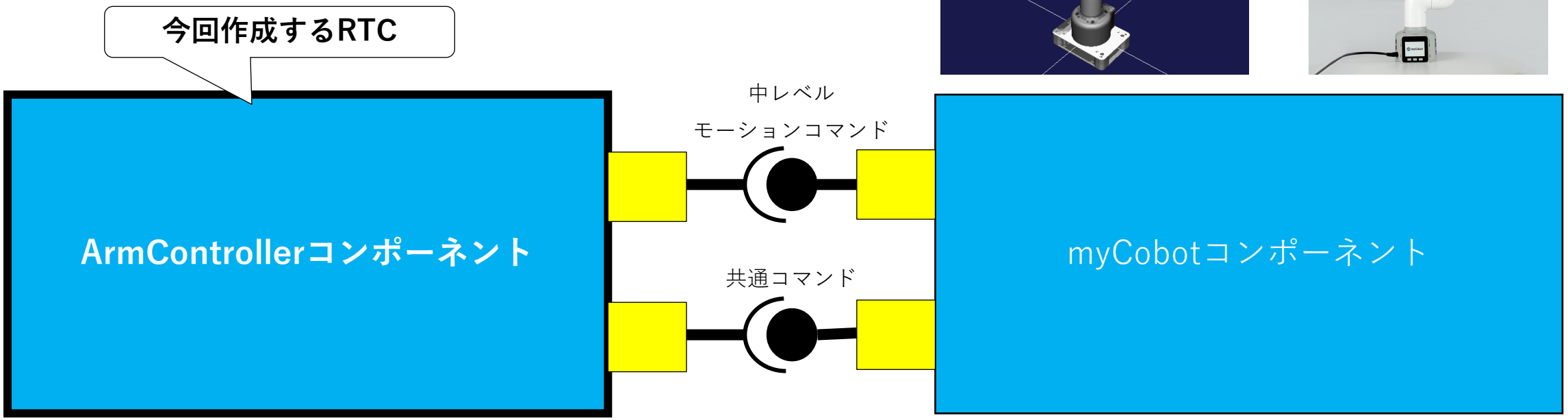
Choreonoid

- Choreonoidはオープンソースのロボット用シミュレーションソフトウェア
  - プラグインによる高い拡張性
  - 3DCGによるロボットモデルのアニメーション表示
  - 動力学シミュレーション
  - センサのシミュレーション(カメラ、レーザーレンジセンサ、力センサ、ジャイロセンサ、・・・)
  - ロボットの動作生成
  - ・・・・



実習内容





- ロボットアーム制御機能共通インタフェースで定義されたコマンドを使用する。
  - **servoON**、**servoOFF**：関節のサーボをオン・オフする(共通コマンド)
  - **goHome**：原点復帰位置に移動する(中レベル)
  - **moveLinearCartesianAbs**：絶対値で指定した目標位置に移動する(中レベル)

https://shop.elephantrobotics.com/cdn/shop/products/myCobot280M5_d22c5ddb-4963-463c-bd22-885076cfb7eb_650x.png

ArmControllerコンポーネントの作成

- ※ここからはRTC Builderで作業します
- RTC Builderで以下のRTコンポーネントを作成する

基本	
コンポーネント名	ArmController
言語	C++
アクティビティ	
onActivated()、onDeactivated()、onExecute()	
データポート	
なし	
サービスポート	
次のスライド以降で説明	
コンフィギュレーション	
次のスライド以降で説明	

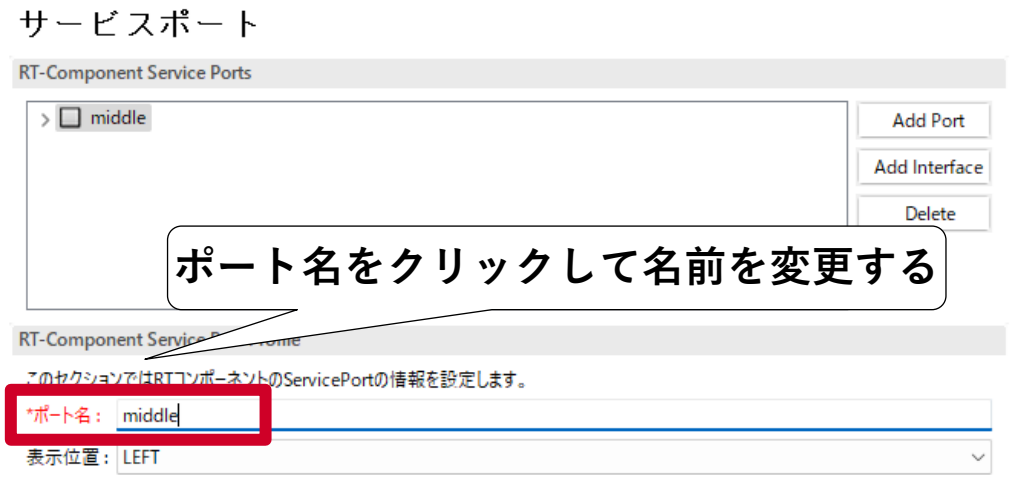
コンフィギュレーションの設定

- 以下のコンフィギュレーションパラメータを設定する
  - **pos_x**
    - データ型：double
    - デフォルト値： 0.2
    - 制約条件： -0.3<x<0.3
    - Widget：text
    - 他の項目は任意
  - **pos_y**
    - データ型：double
    - デフォルト値： 0.0
    - 制約条件： -0.3<x<0.3
    - Widget： text
    - 他の項目は任意
  - **pos_z**
    - データ型：double
    - デフォルト値： 0.16
    - 制約条件： -0.0<x<0.45
    - Widget： text
    - 他の項目は任意

サービスポートの設定

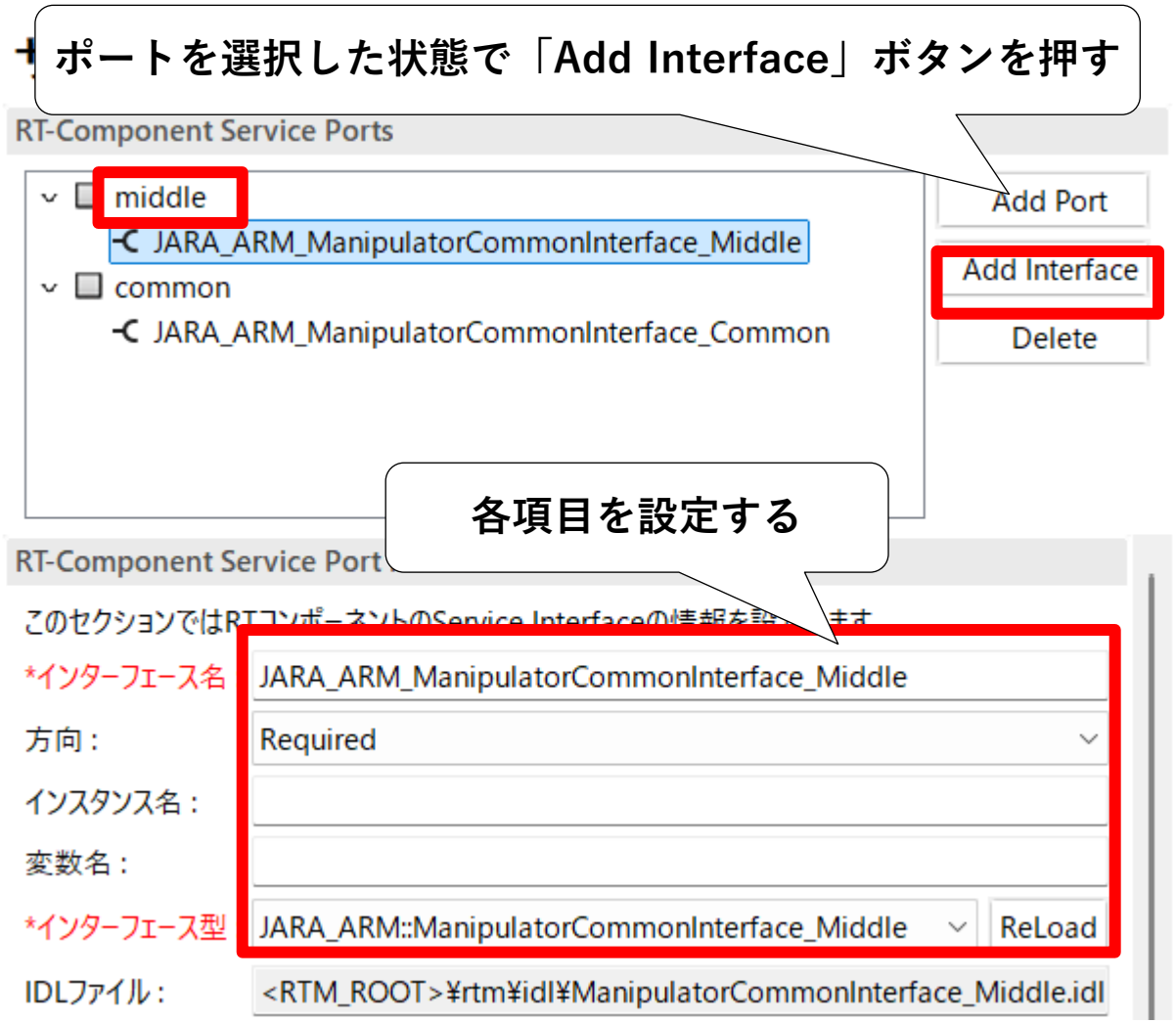
- サービスポートの追加、設定を行う

「サービスポート」タブを選択



サービスポートの設定

- サービスインターフェースの追加、設定を行う

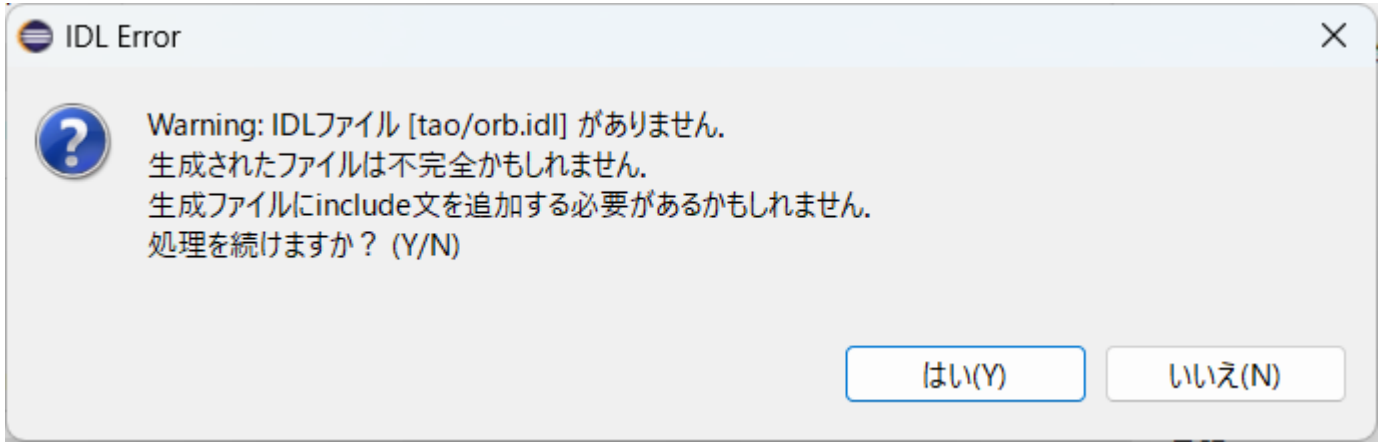


サービスポートの設定

- 以下のサービスポート・インターフェースを設定する
  - ポート名： **middle**
    - インターフェース名：JARA_ARM_ManipulatorCommonInterface_Middle
      - 方向： Required
      - インターフェース型：JARA_ARM::ManipulatorCommonInterface_**Middle**
    - 他の項目は任意
  - ポート名： **common**
    - インターフェース名：JARA_ARM_ManipulatorCommonInterface_Common
      - 方向： Required
      - インターフェース型：JARA_ARM::ManipulatorCommonInterface_**Common**
    - 他の項目は任意

実行ファイルの生成

1. RTCBuilderによるコード生成
    1. コード生成時に以下の画面が出ますが、「はい」を選択する



2. CMakeによるVisual Studioプロジェクトファイル等の生成
3. **コード編集**
4. ビルドしてArmController¥build¥src¥Release(もしくはDebug)フォルダに実行ファイルArmControllerComp.exeを生成

コード編集

```
127  RTC::ReturnCode_t ArmController::onActivated(RTC::UniqueId /*ec_id*/)
128  {
129      //サーボをONにするコマンドを実行
130      m_JARA_ARM_ManipulatorCommonInterface_Common->servoON();
131      //アームを原点復帰位置に移動する
132      m_JARA_ARM_ManipulatorCommonInterface_Middle->goHome();
133    return RTC::RTC_OK;
134  }
```

onActivated関数に追加

```
137  RTC::ReturnCode_t ArmController::onDeactivated(RTC::UniqueId /*ec_id*/)
138  {
139      //サーボをOFFにするコマンドを実行
140      m_JARA_ARM_ManipulatorCommonInterface_Common->servoOFF();
141    return RTC::RTC_OK;
142  }
143
```

onDeactivated関数に追加

コード編集

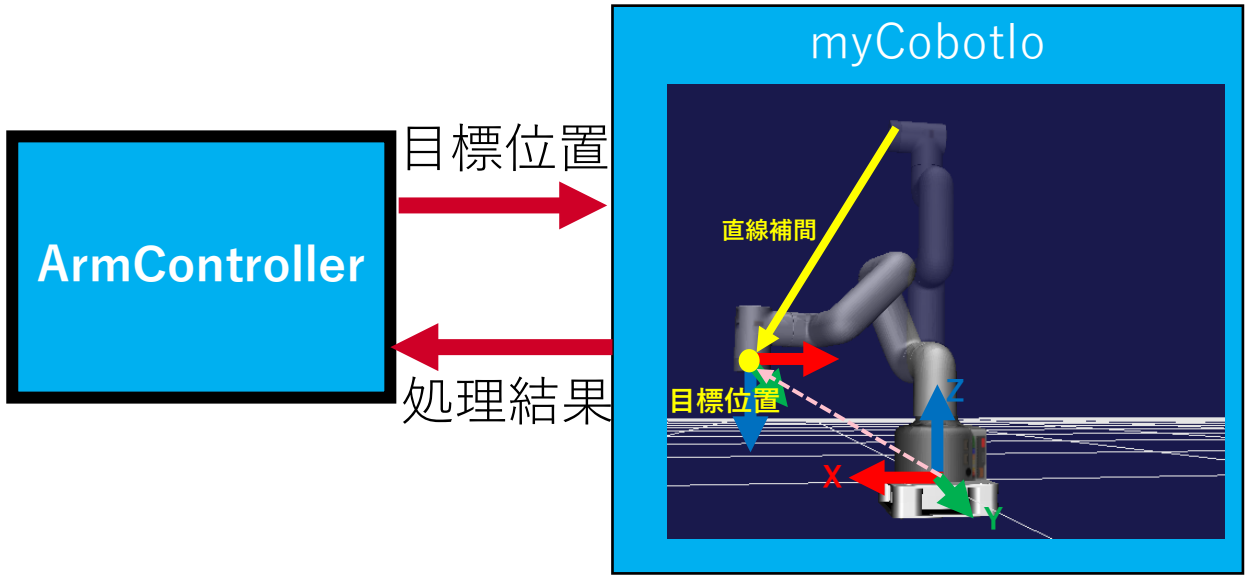
```
RTC::ReturnCode_t ArmController::onExecute(RTC::UniqueId /*ec_id*/)
{
    //コンフィギュレーションパラメータの変更時のみ実行
    if (getConfigService().isChanged())
    {
        //コンフィギュレーションパラメータを更新する
        updateParameters("default");
        //CarPosWithElbow型データに手先の位置姿勢を格納する
        JARA_ARM::CarPosWithElbow pose;
        //姿勢は固定
        pose.carPos[0][0] = -1;
        pose.carPos[0][1] = 0;
        pose.carPos[0][2] = 0;
        pose.carPos[1][0] = 0;
        pose.carPos[1][1] = 1;
        pose.carPos[1][2] = 0;
        pose.carPos[2][0] = 0;
        pose.carPos[2][1] = 0;
        pose.carPos[2][2] = -1;
        //コンフィギュレーションパラメータを位置情報として格納する
        pose.carPos[0][3] = m_pos_x;
        pose.carPos[1][3] = m_pos_y;
        pose.carPos[2][3] = m_pos_z;
        pose.elbow = 0;
        pose.structFlag = 0;


        //絶対値で指定した目標位置に移動するコマンドを実行
        //移動完了まで処理は戻らない
        m_JARA_ARM_ManipulatorCommonInterface_Middle->moveLinearCartesianAbs(pose);

    }

  return RTC::RTC_OK;
}
```

onExecute関数に追加

定義したインターフェースに対応するオブジェクトから関数呼び出し

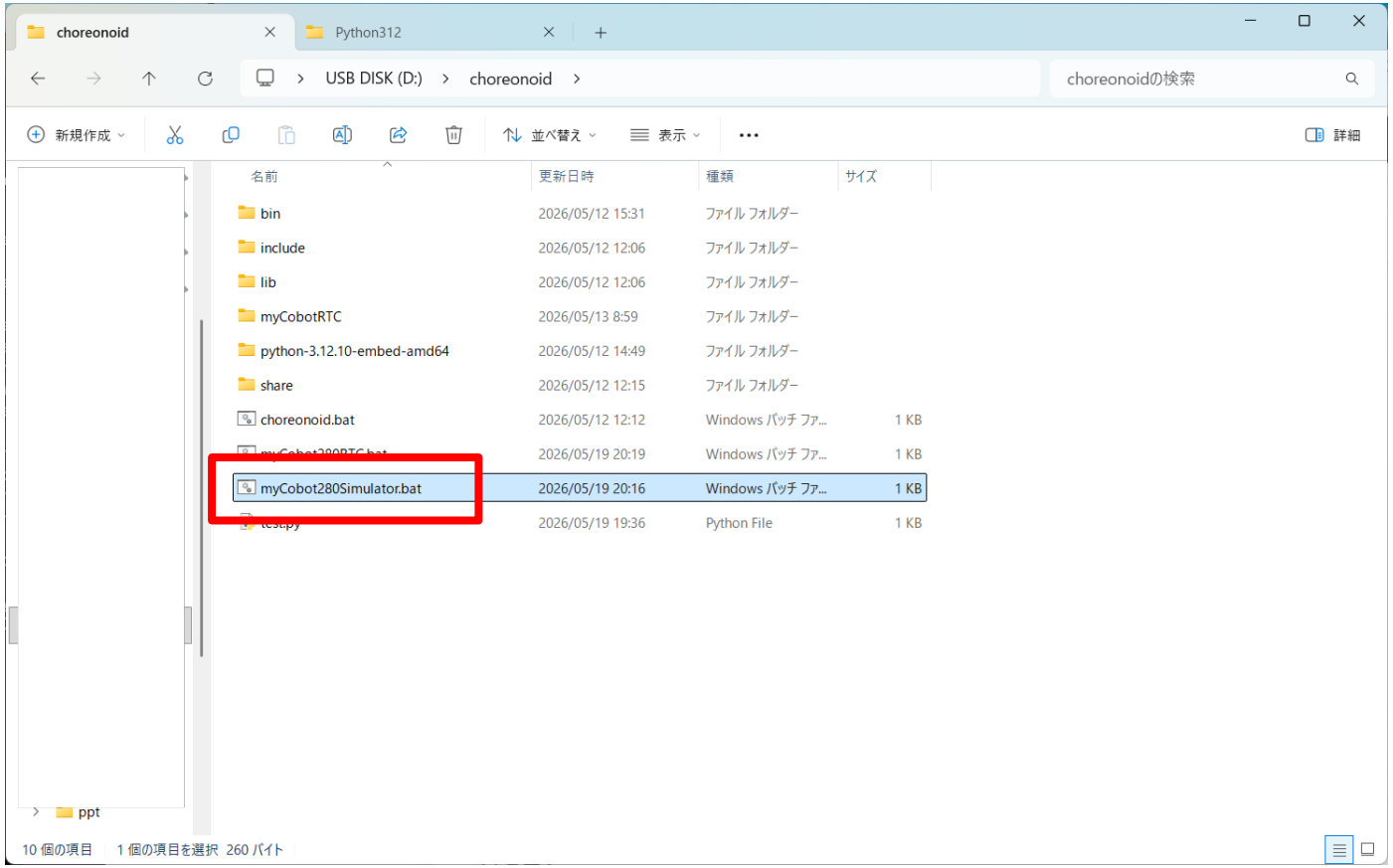
```
m_JARA_ARM_ManipulatorCommonInterface_Middle
                    ->moveLinearCartesianAbs(pose);
```

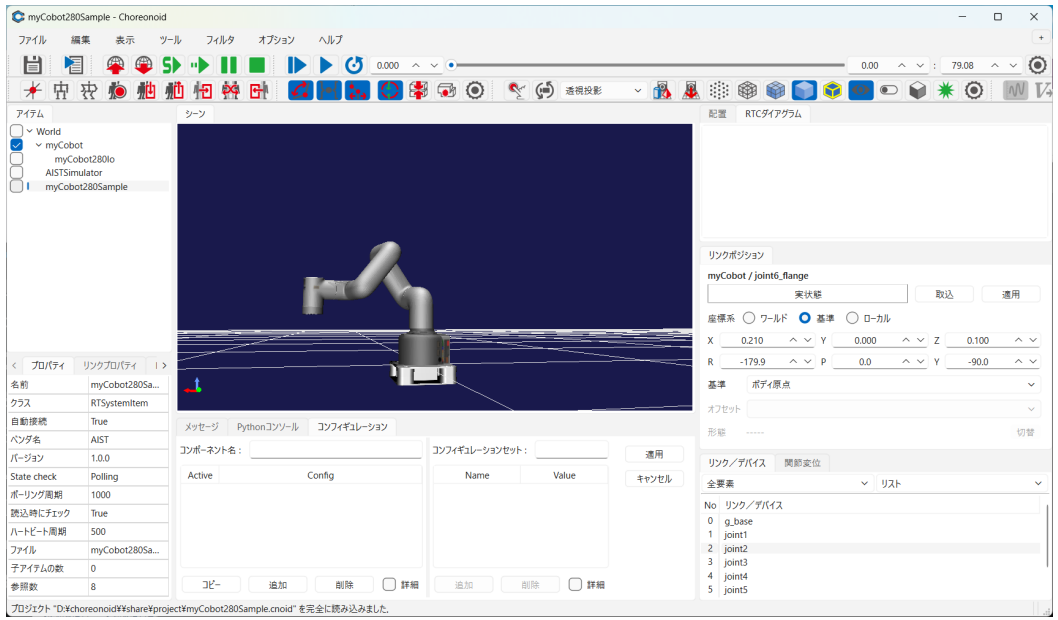
- moveLinearCartesianAbs関数は、ロボット座標系の絶対値で指定された目標位置に対し直線補間で動作する。
- Provided型インターフェースの関数を遠隔呼び出しする。



Choreonoidの起動

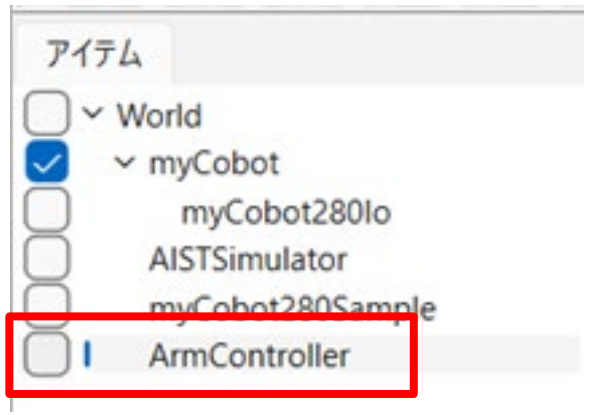
- Windowsの場合、配布資料（USBメモリ）のchoreonoidフォルダの**myCobot280Simulator.bat**を実行する
  - Ubuntuの場合は以下のコマンド
    - choreonoid ${資料のディレクトリ}/choreonoid/myCobot280Sample/myCobot280Sample.cnoid





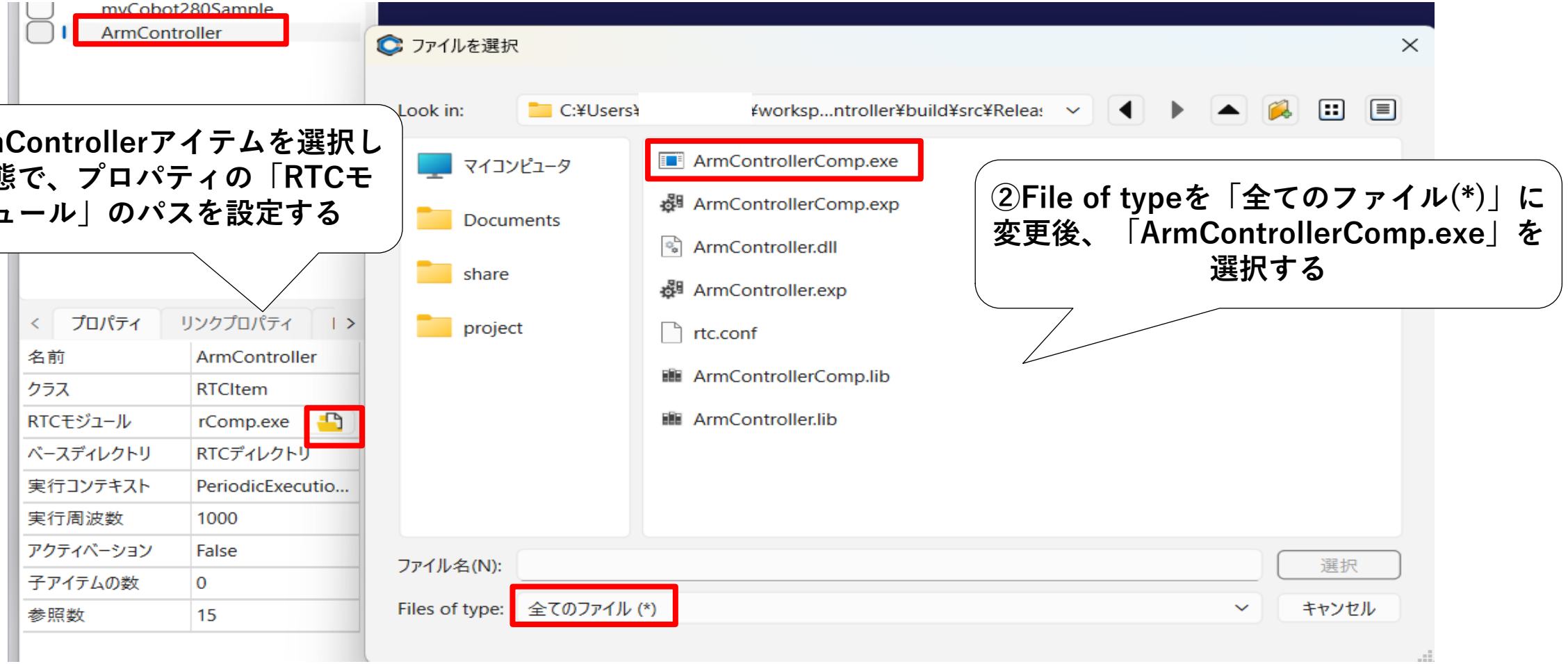
RTCアイテムの追加

- RTCアイテムを追加する
  - ファイル→新規→RTC
  - 名前はArmControllerで生成
  - アイテムツリーにArmControllerが追加される

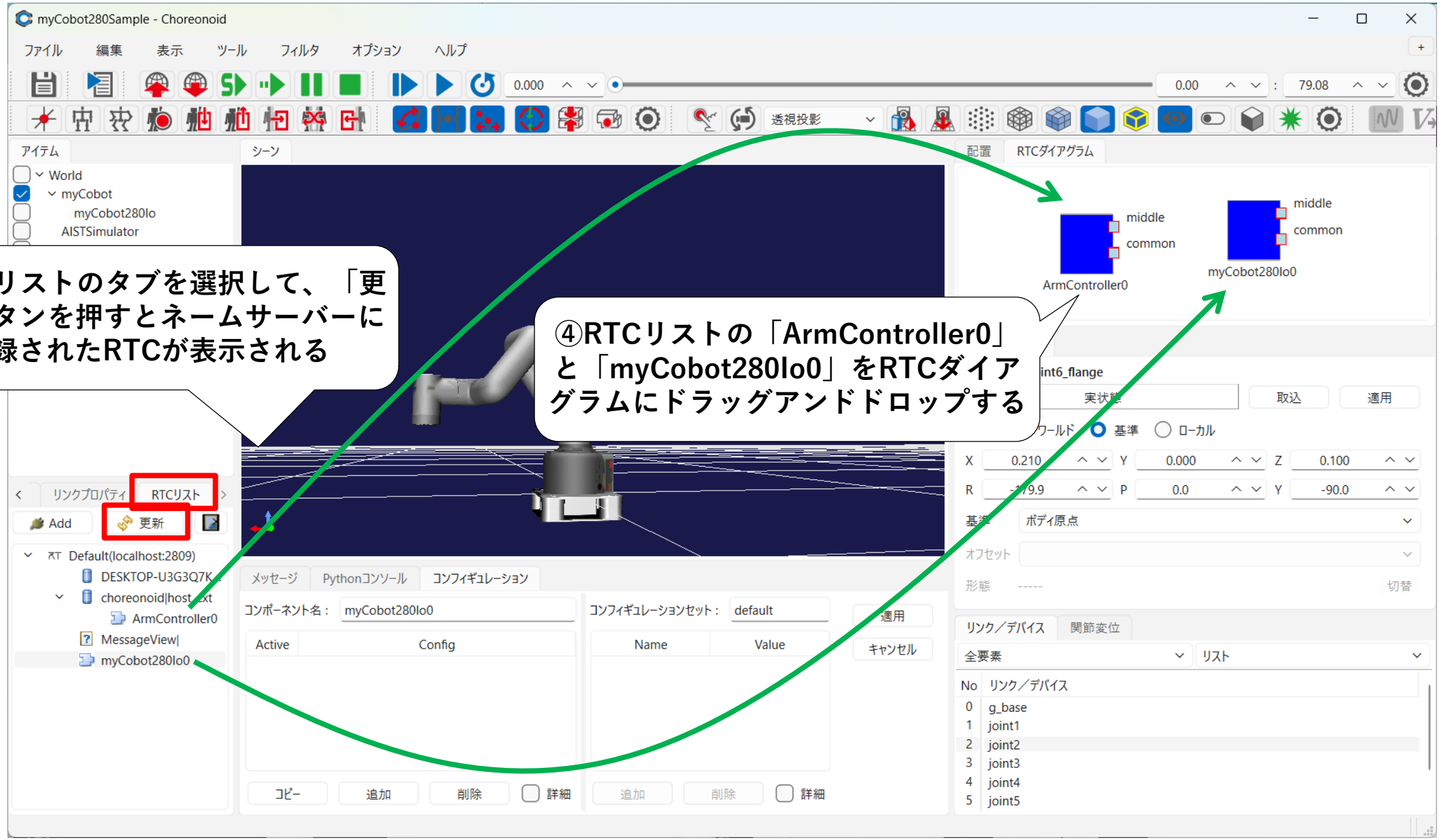


RTCアイテムの設定

- RTCモジュールのパスを設定する
  - 生成したArmControllerComp.exeを指定することにより、RTCを自動起動する
    - Ubuntuの場合はArmControlle.soを指定する

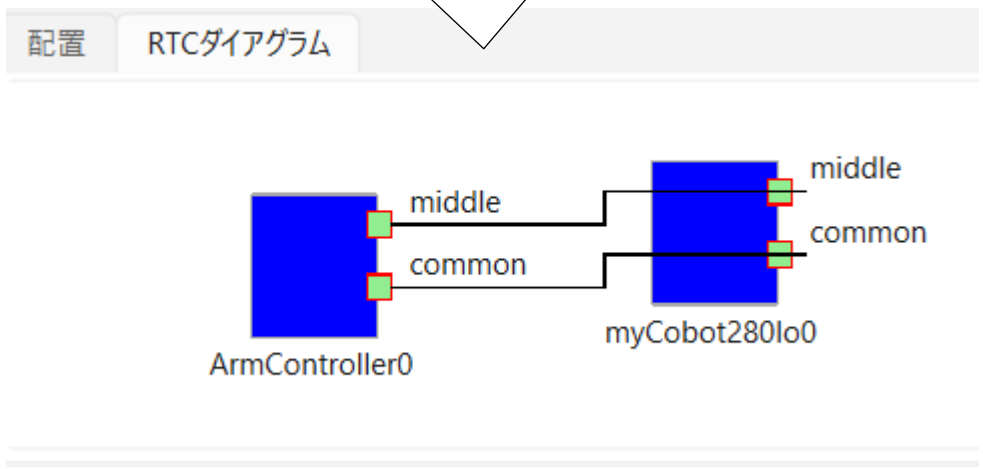


RTシステム構築

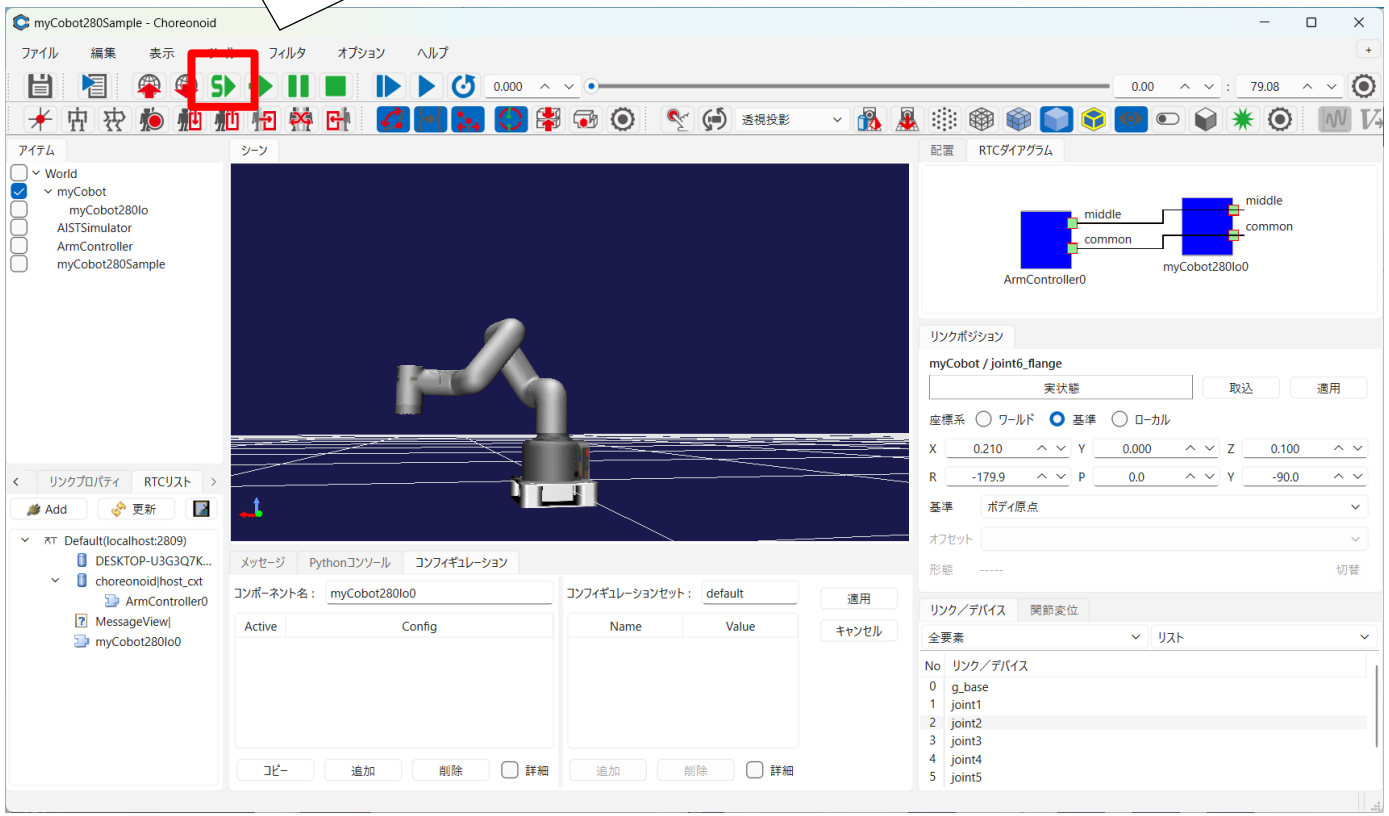

③RTCリストのタブを選択して、「更新」ボタンを押すとネームサーバーに登録されたRTCが表示される
④RTCリストの「ArmController0」と「myCobot280Io0」をRTCダイアグラムにドラッグアンドドロップする
myCobot280Sample - Choreonoid
ファイル
編集
表示
ツール
フィルタ
オプション
ヘルプ
アイテム
World
myCobot
myCobot280Io
AISTSimulator
シーン
配置
RTCダイアグラム
middle
common
ArmController0
myCobot280Io0
リンクプロパティ
RTCリスト
Add
更新
Default(localhost:2809)
DESKTOP-U3G3Q7K
choreonoid|host_ext
ArmController0
MessageView|
myCobot280Io0
メッセージ
Pythonコンソール
コンフィギュレーション
コンポーネント名：
myCobot280Io0
コンフィギュレーションセット：
default
適用
キャンセル
Active
Config
Name
Value
コピー
追加
削除
詳細
実状態
取込
適用
ワールド
基準
ローカル
X
Y
Z
R
P
Y
ボディ原点
オフセット
形態
切替
リンク／デバイス
関節変位
全要素
リスト
No
リンク／デバイス
0 g_base
1 joint1
2 joint2
3 joint3
4 joint4
5 joint5

RTシステム構築・シミュレーション開始

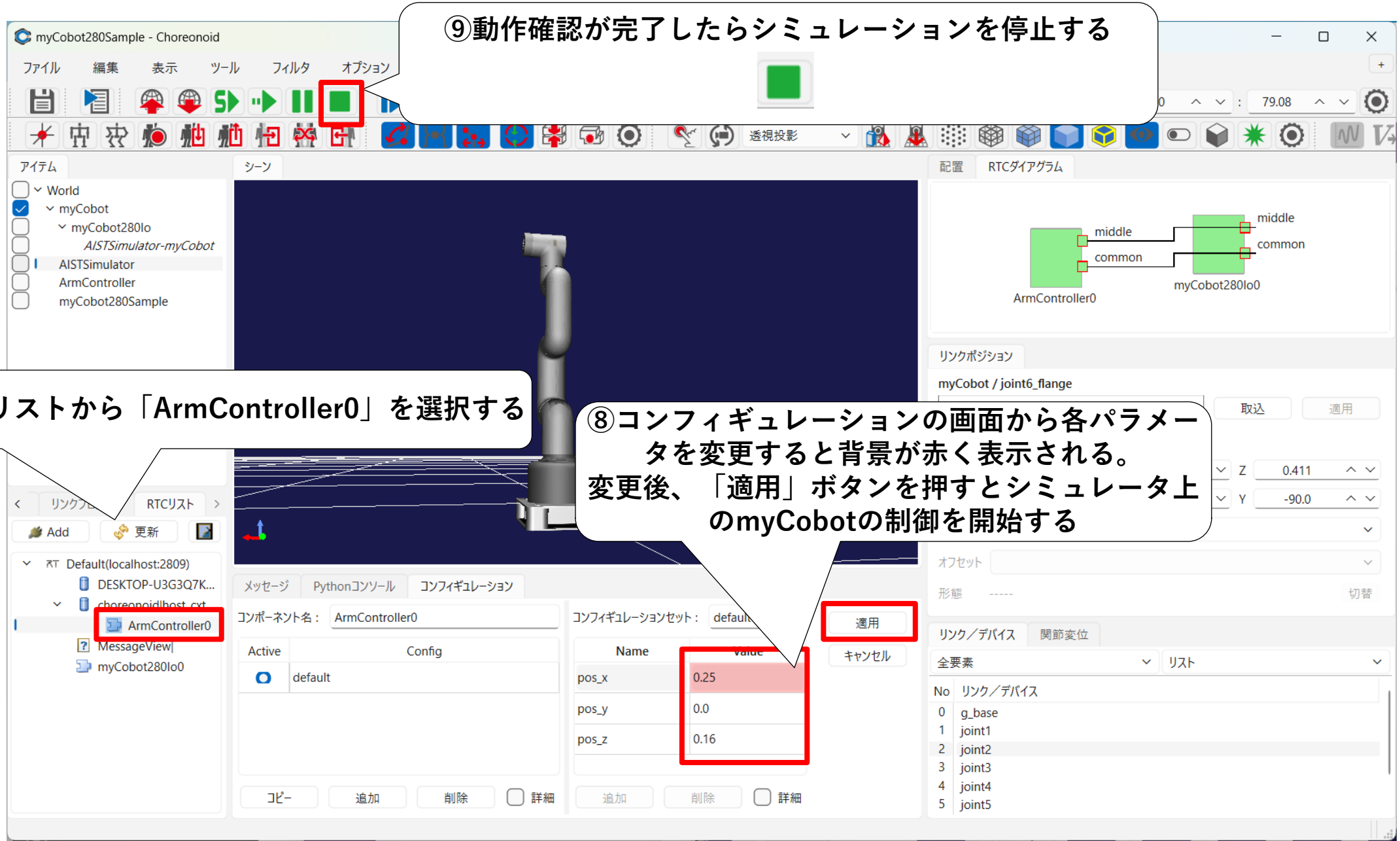
⑤RTCダイアグラム上で2つのRTCのmiddleポート同士とcommoポート同士を接続する



⑥以下のボタンでシミュレーションを開始する

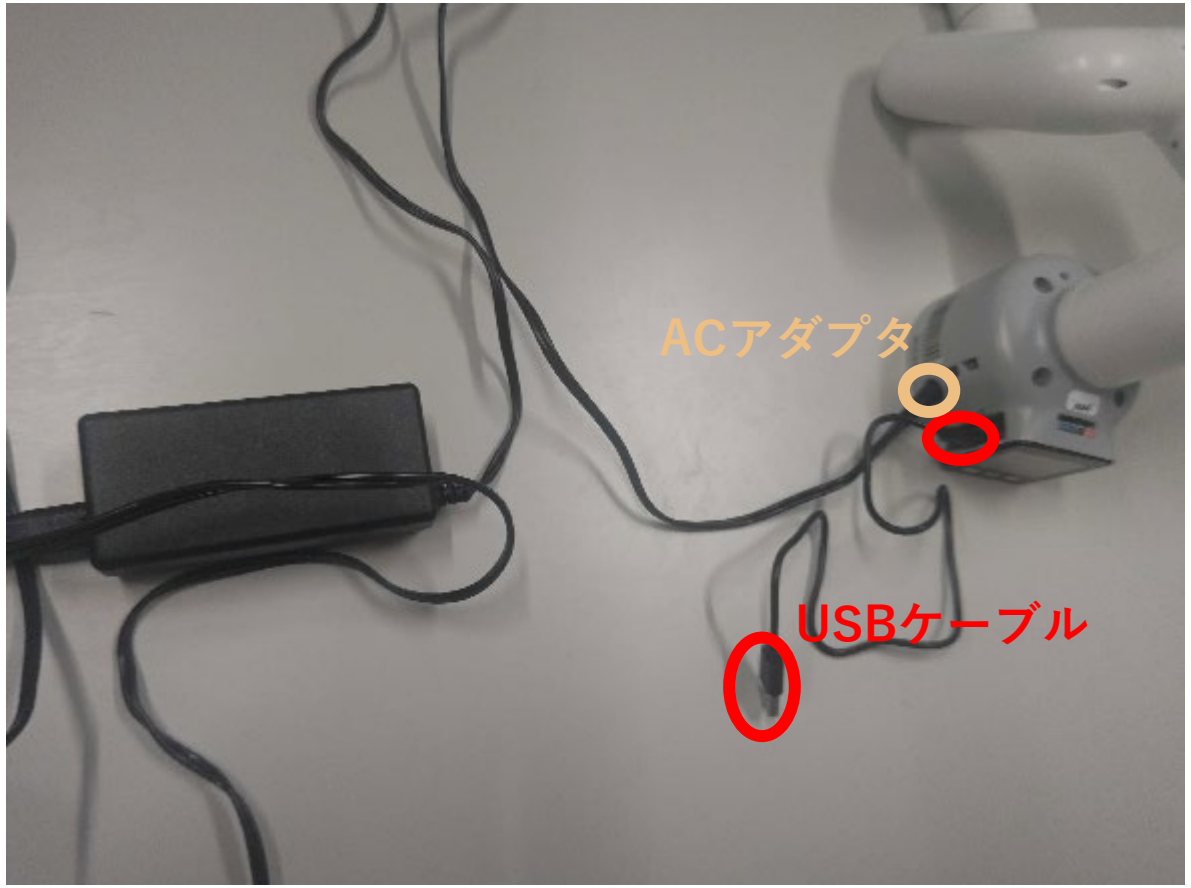


コンフィギュレーションパラメータの操作



myCobot実機の制御

- ArmControllerコンポーネントでmyCobot実機を制御する
  - myCobotにACアダプタとUSBケーブルを接続する
  - USBケーブルはノートPCと接続する

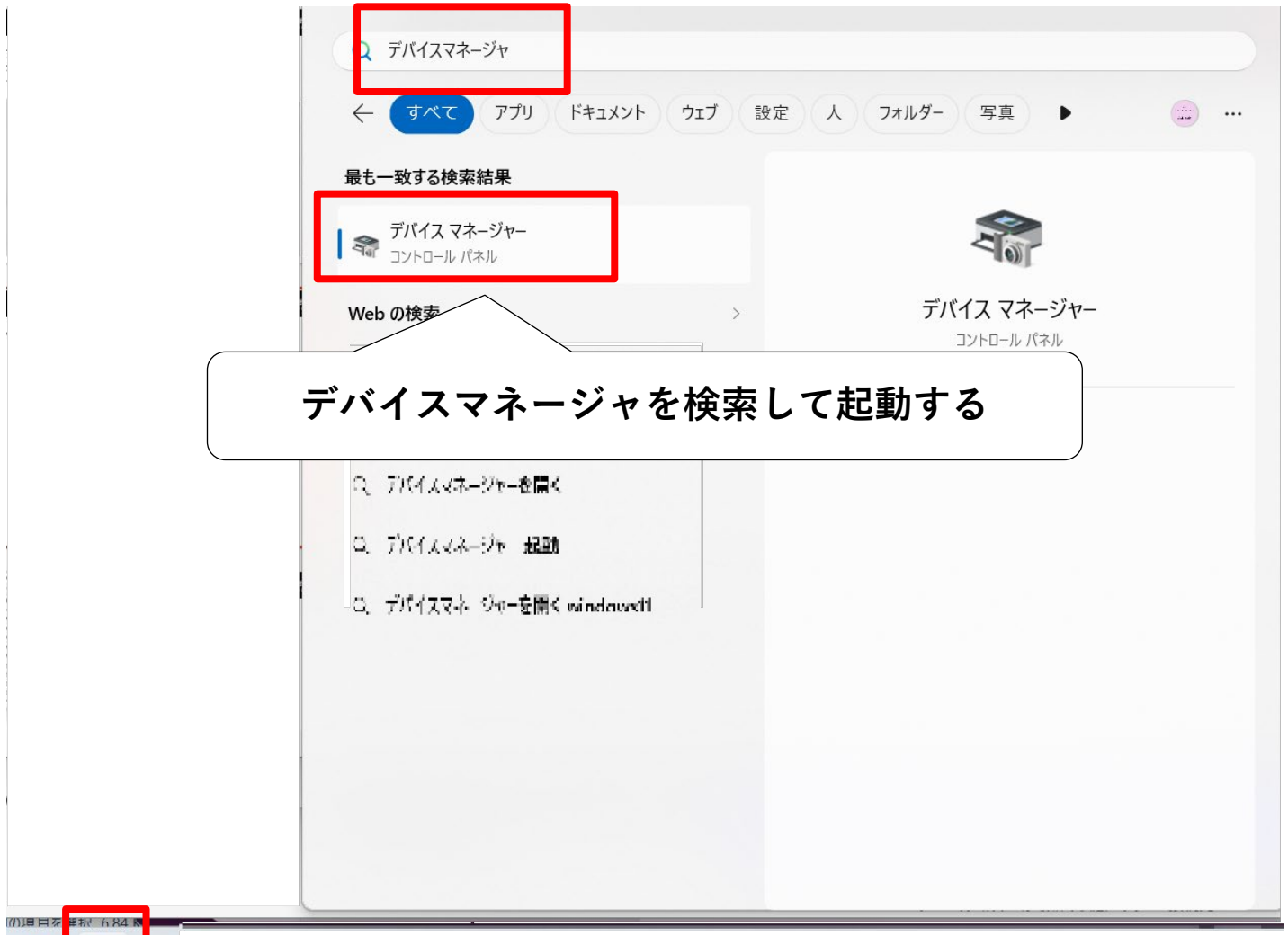


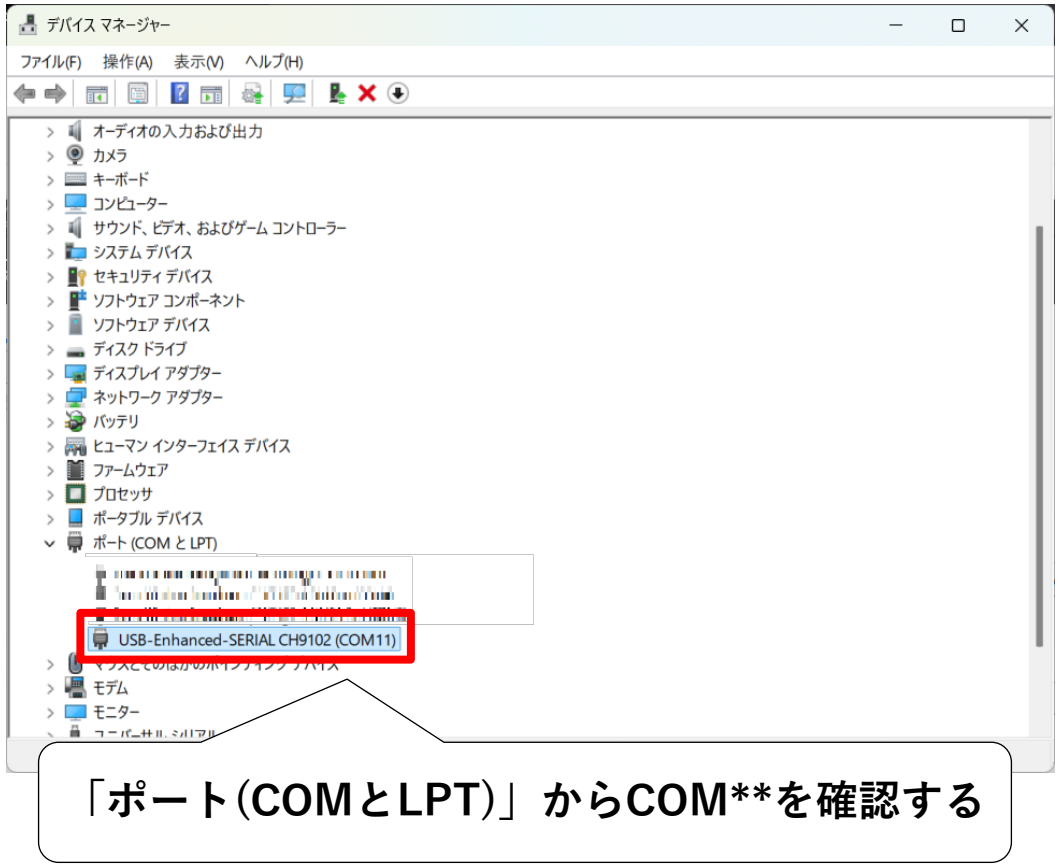
myCobot実機の制御

- myCobotの操作画面から、**Transponder**を選択して**myCobnot Basic Transponder**の画面を表示する

ポート番号の確認

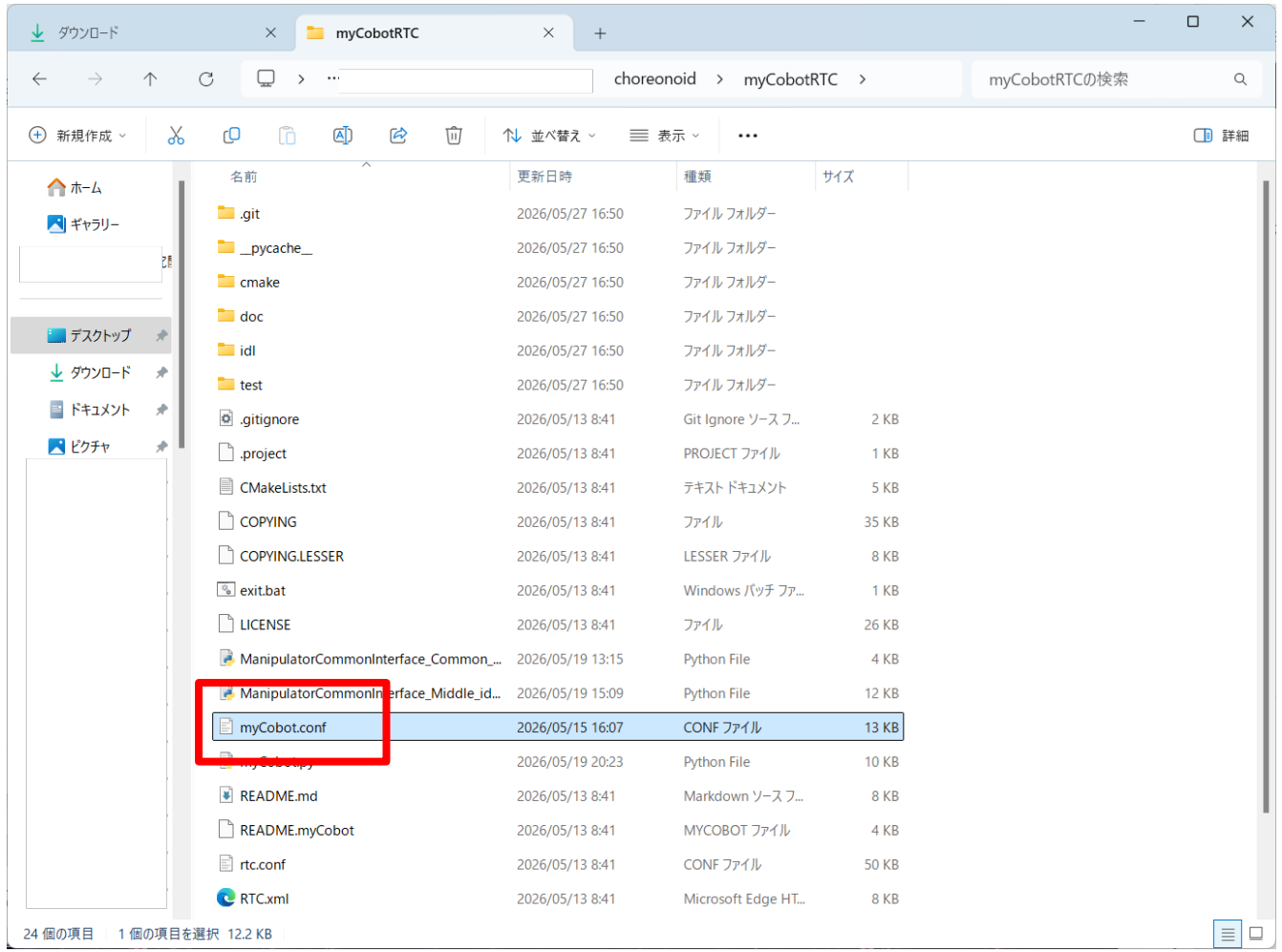
- デバイスマネージャでポート番号(COM＊＊)を確認する





myCobotコンポーネントの起動

- ポート番号を指定する
  - 配布資料（USBメモリ）のchoreonoid¥myCobotRTCフォルダの**myCobot.conf**をメモ帳などで開いて編集する

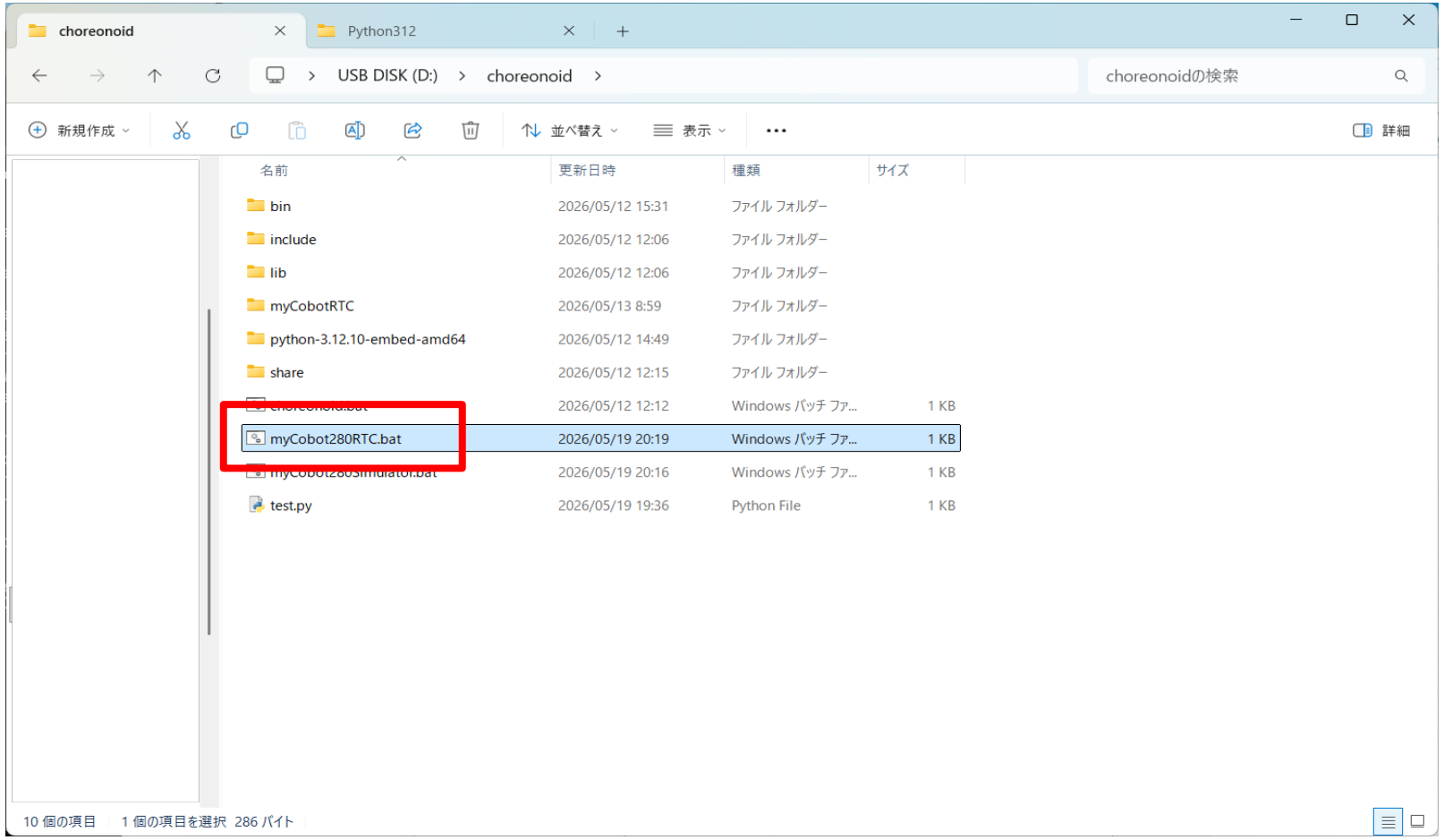



```
50  # Available configuration parameters
51  #
52  conf.default.com_port: COM11
```

com_portのコンフィギュレーションパラメータをデバイスマネージャで確認した「COM**」にしてファイルを保存する

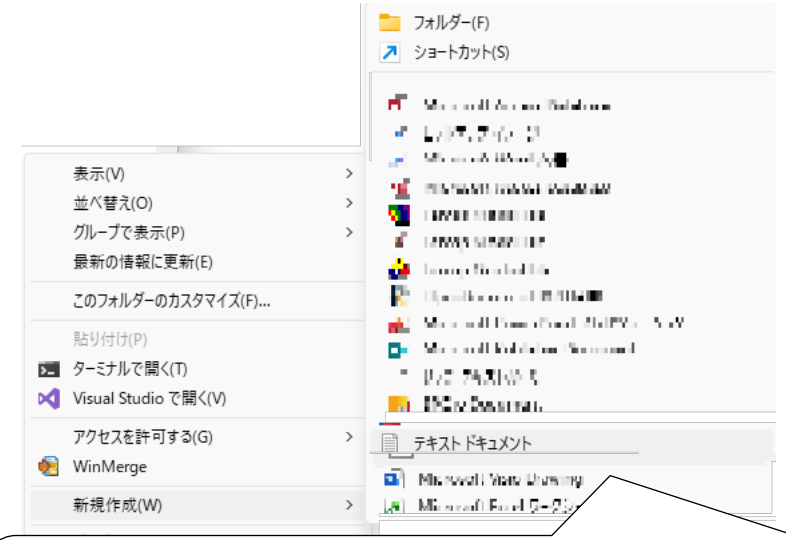
myCobotコンポーネントの起動

- myCobotコンポーネントを起動する
  - Windowsの場合は配布資料（USBメモリ）のchoreonoidフォルダの**myCobot280RTC.bat**を実行
    - Ubuntuの場合は以下のコマンドを実行
      - python3 myCobot.py



ArmControllerコンポーネントの起動

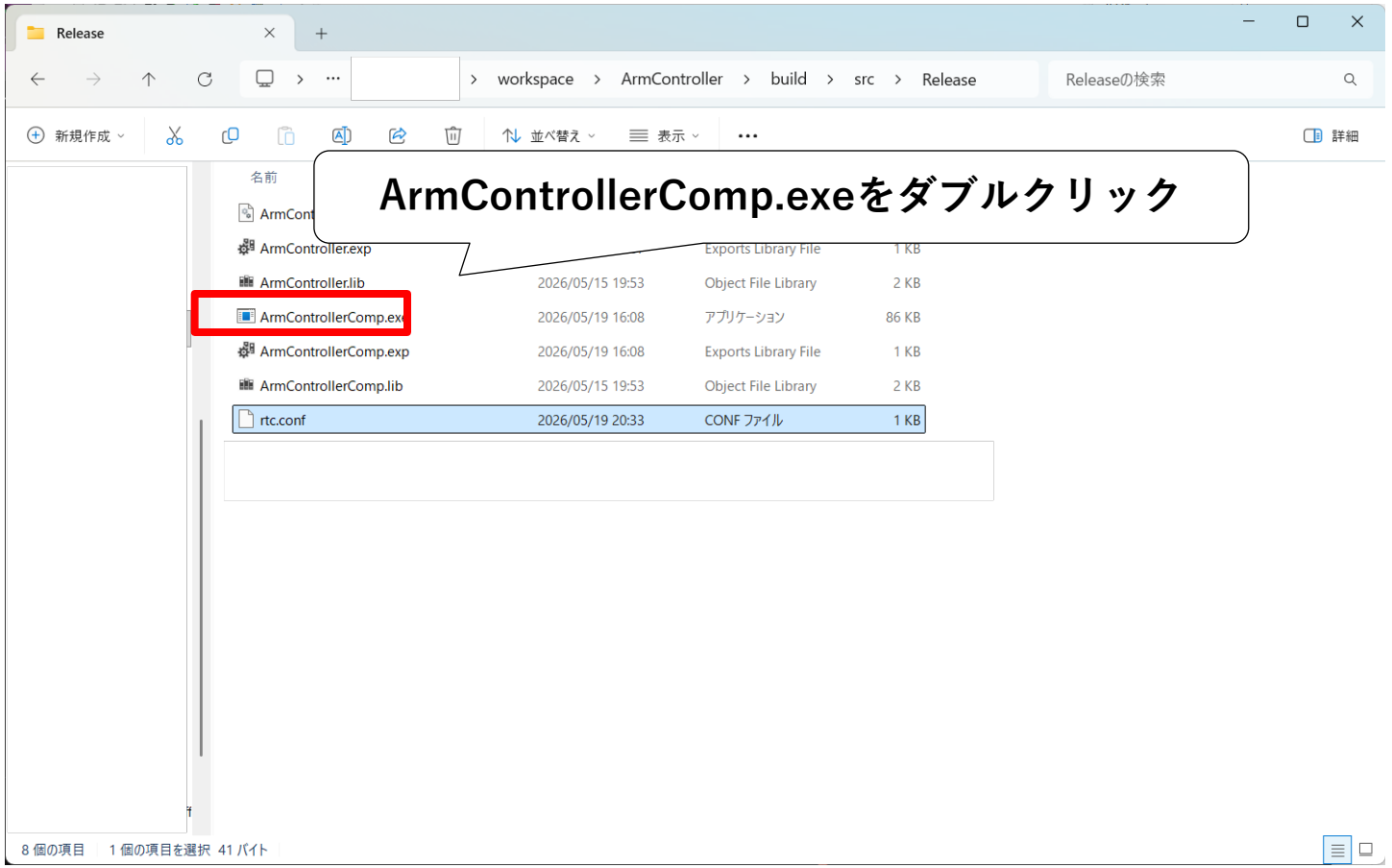
- ArmControllerコンポーネントを起動する
  - rtc.confというファイルを作成する
  - ArmControllerComp.exe (Ubuntuの場合はArmControllerComp)を起動する



ArmControllerComp.exeのフォルダで右クリック→新規作成→テキストドキュメントでrtc.conf と言う名前のファイルを作成

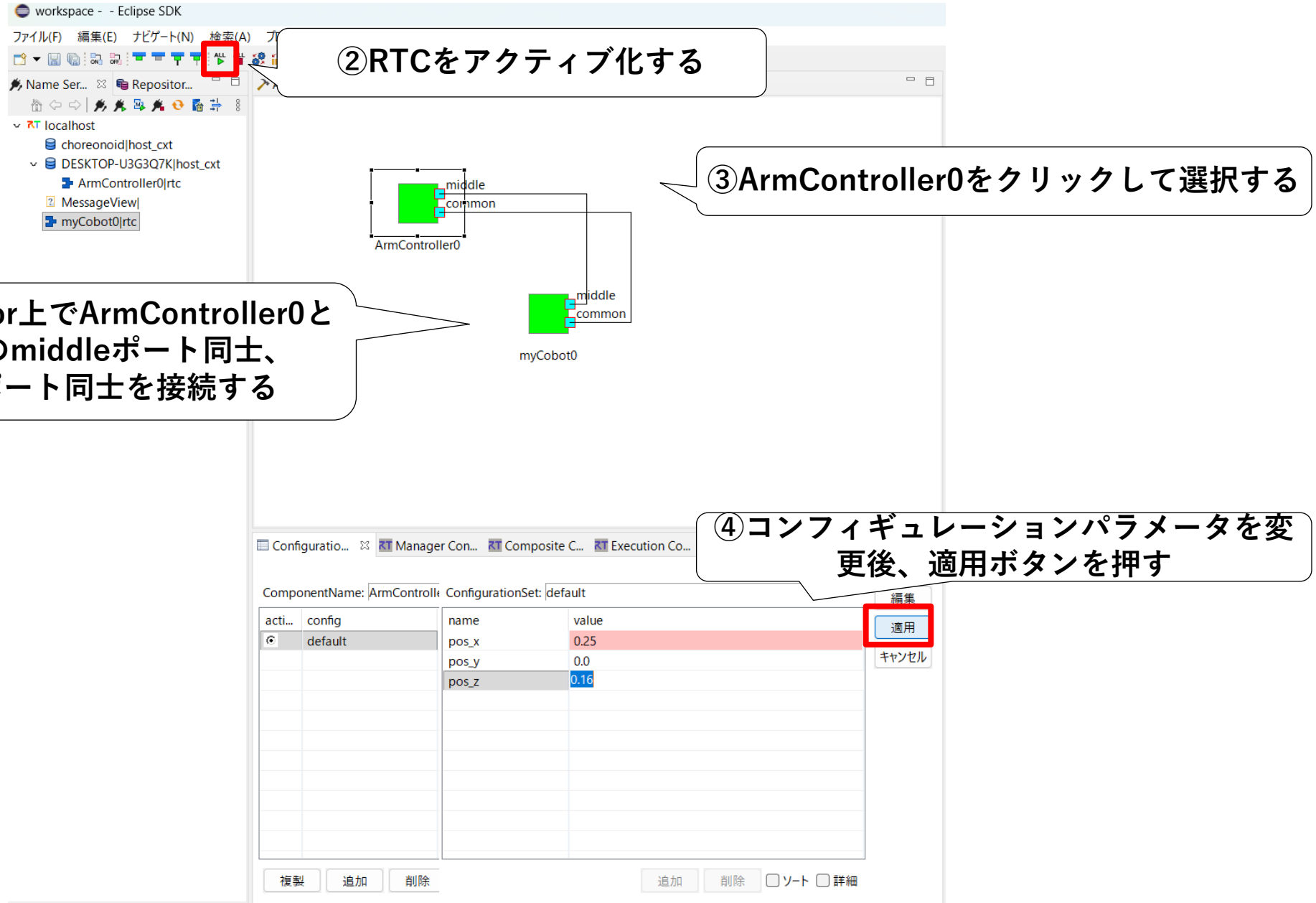
```
corba.args: -ORBclientCallTimeOutPeriod 0
```

- **rtc.confをメモ帳などで開いて、この1行だけ追加**
- **コマンド終了まで処理が戻らないため、通信がタイムアウトしないように設定する**



ArmControllerComp.exeをダブルクリック

動作確認


workspace - - Eclipse SDK
ファイル(F) 編集(E) ナビゲート(N) 検索(A)
Name Ser... Repositor...
localhost
choreonoid|host_cxt
DESKTOP-U3G3Q7K|host_cxt
ArmController0|rtc
MessageView|
myCobot0|rtc
②RTCをアクティブ化する
middle
common
ArmController0
middle
common
myCobot0
③ArmController0をクリックして選択する
①SystemEditor上でArmController0とmyCobot0のmiddleポート同士、commonポート同士を接続する
④コンフィギュレーションパラメータを変更後、適用ボタンを押す
Configuratio... Manager Con... Composite C... Execution Co...
ComponentName: ArmControll ConfigurationSet: default
acti... config
default
name value
pos_x 0.25
pos_y 0.0
pos_z 0.16
編集
適用
キャンセル
複製 追加 削除
追加 削除 ソート 詳細