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

インストールの確認(Windows)

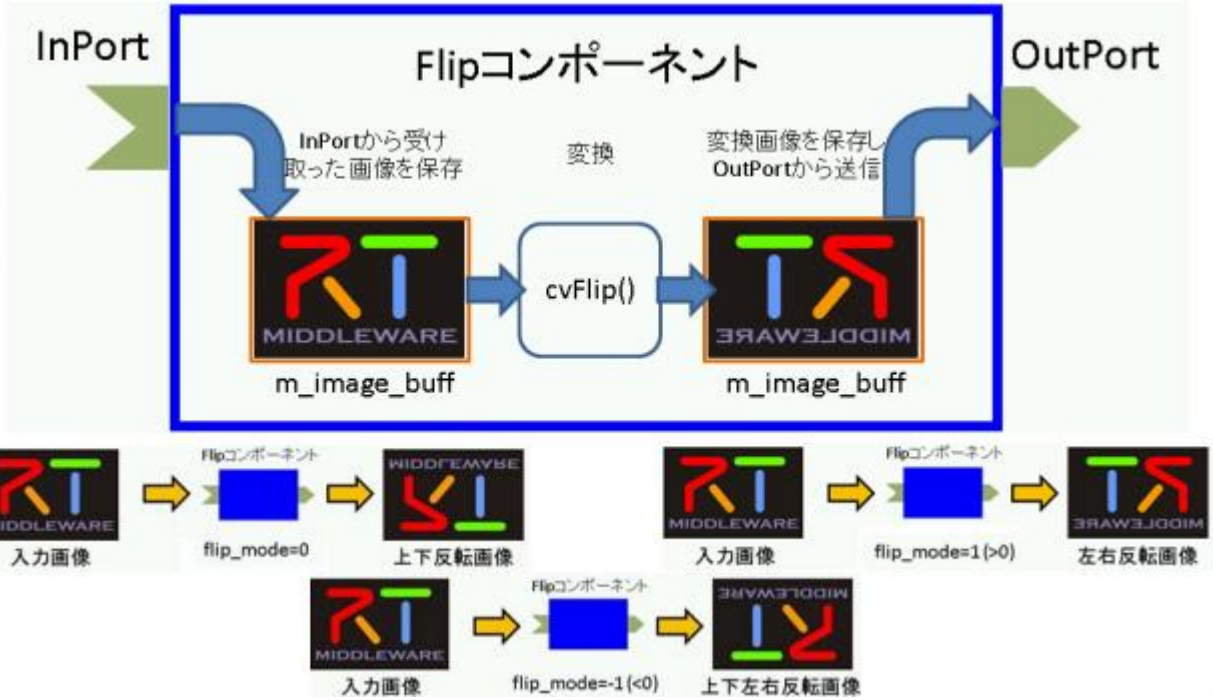
- OpenRTM-aist
  - OpenRTM-aist-1.1.2-RELEASE_x86.msi
  - インストール後に再起動する
  - Visual Studio 2013以外(2010、2012、2015)を使用する場合は環境変数を変更
    - 「RTM_VC_VERSION」をvc10、vc11、vc13
    - 配布のUSBメモリに同梱してあるツールでも設定可能
- Python
  - python-2.7.10.msi
    - 2.7.11は不具合が発生するため非推奨
  - ※OpenRTM-aistの32bit版をインストールする場合Pythonも32bit版をインストールする。
    OpenRTM-aistの64bitをインストールする場合はPythonも64bit版をインストールする。
- PyYAML
  - PyYAML-3.11.win32-py2.7.exe
- CMake
  - cmake-3.5.2-win32-x86.msi
- Doxygen
  - doxygen-1.8.11-setup.exe
- Visual Studio
  - Visual Studio 2013 Community Edition

インストールの確認(Ubuntu)

- OpenRTM-aist
  - $ sudo sh pkg_install_ubuntu.sh
- CMake
  - $ sudo apt-get install cmake cmake-gui
- Doxygen
  - $ sudo apt-get install doxygen
- RT System Editor、RTC Builder
  - eclipse442-openrtp112v20160526-ja-linux-gtk-x86_64.tar.gzを適当な場所に展開
- Java
  - $ sudo apt-get default-jre
- OpenCV
  - $ sudo apt-get install libopencv-dev libcv2.4 libcvaux2.4 libhighgui2.4
- OpenCVのサンプルコンポーネント
  - 自分でビルドする
    - $ svn co http://svn.openrtm.org/ImageProcessing/trunk/ImageProcessing/opencv/
    - $ cd opencv
    - $ mkdir work
    - $ cd work
    - $ cmake ..
    - $ make
    - $ sudo make install
- Code::Blocks(任意)
  - $ sudo apt-get install codeblocks

実習内容

- 画像の反転を行うコンポーネントの作成
  - InPortで受信した画像データを処理してOutPortから出力
    - データポートの使用方法を習得
  - コンフィギュレーションパラメータにより反転する方向を設定
    - コンフィギュレーションパラメータの使用方法を習得
  - RT System Editorにより他のRTCと接続、RTCをアクティブ化
    - RT System Editorの使い方を習得



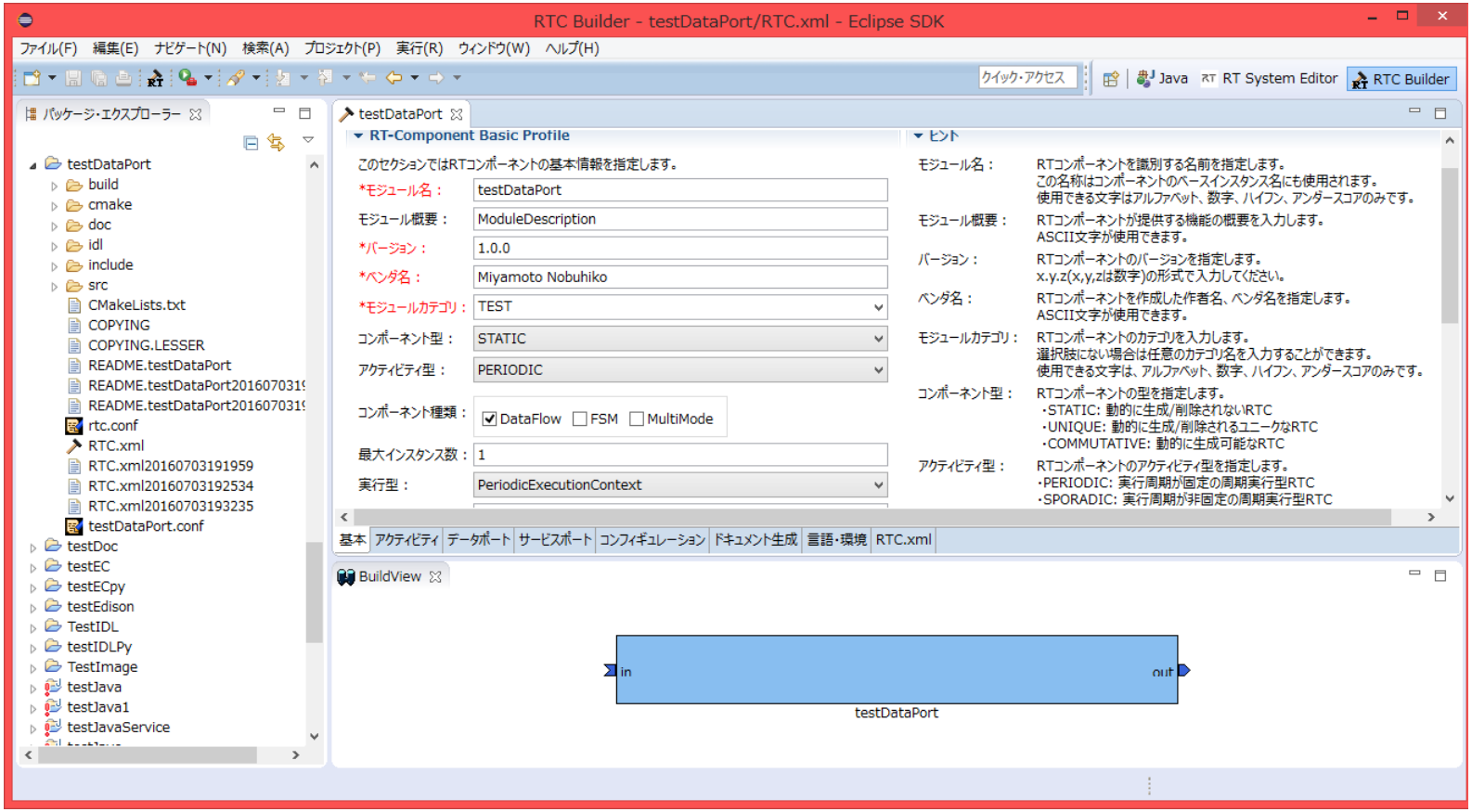
全体の手順

- RTC Builderによるソースコード等のひな型の作成
- ソースコードの編集、ビルド
  - ビルドに必要な各種ファイルを生成
    - CMakeLists.txtの編集
    - CMakeにより各種ファイル生成
  - ソースコードの編集
    - Flip.hの編集
    - Flip.cppの編集
  - ビルド
    - Visual Studio、Code::Blocks
- RTシステムエディタによるRTシステム作成、動作確認
  - RTシステム作成
    - データポート接続、コンフィギュレーションパラメータ設定

コンポーネント開発ツール RTC Builderについて

RTC Builder

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



RTC Builderの起動

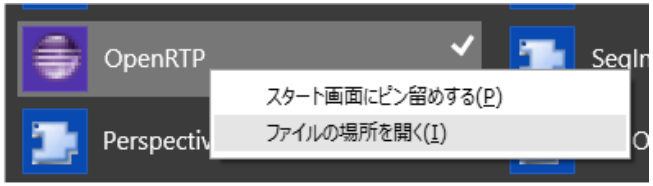
- 起動する手順
  - Windows 7
    - 「スタート」→「すべてのプログラム」→「OpenRTM-aist 1.1.2」→「Tools」→「OpenRTP」
  - Windows 8.1
    - 「スタート」→「アプリビュー(右下矢印)」→「OpenRTM-aist 1.1.2」→「OpenRTP」
    - ※同じフォルダに「RTSystemEditorRCP」がありますが、これはRTC Builderが使えないので今回は「OpenRTP」を起動してください。
  - Ubuntu
    - Eclipseを展開したディレクトリに移動して以下のコマンド
    - $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

RTC Builderの起動

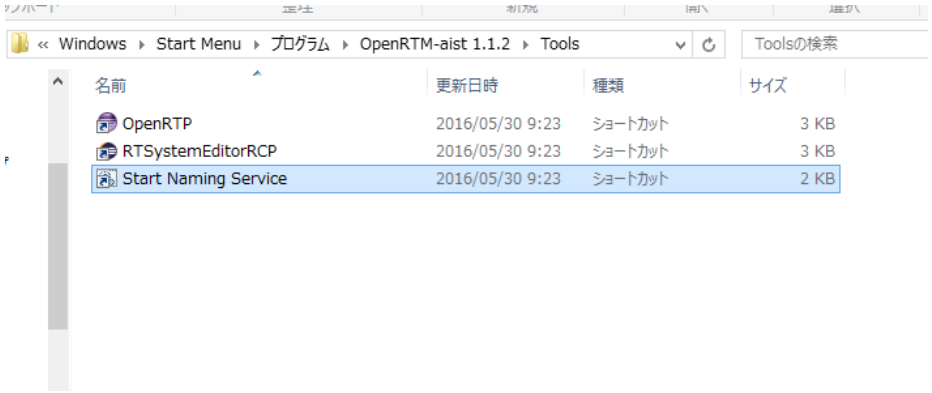
- Windows 8.1

RTC Builderの起動

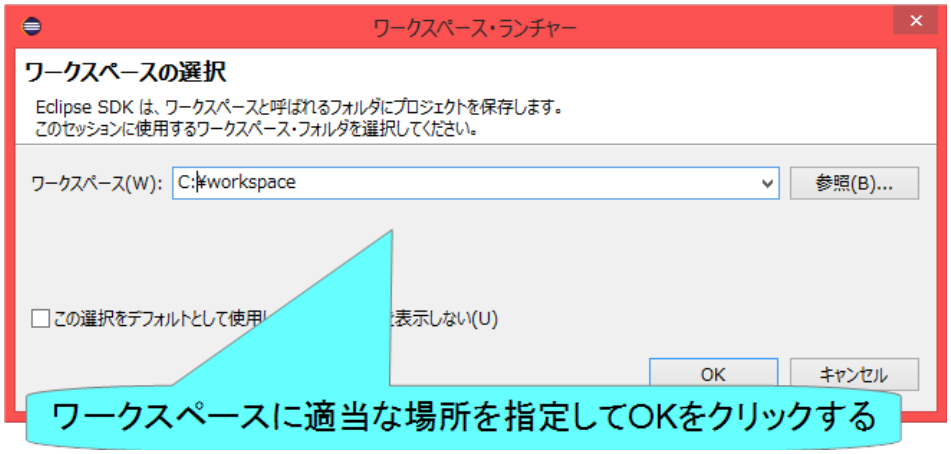
- いちいちアプリビューから起動するのは非常に手間がかかるため、以下の作業をしてスタートメニューのフォルダを開いておくことをお勧めします。



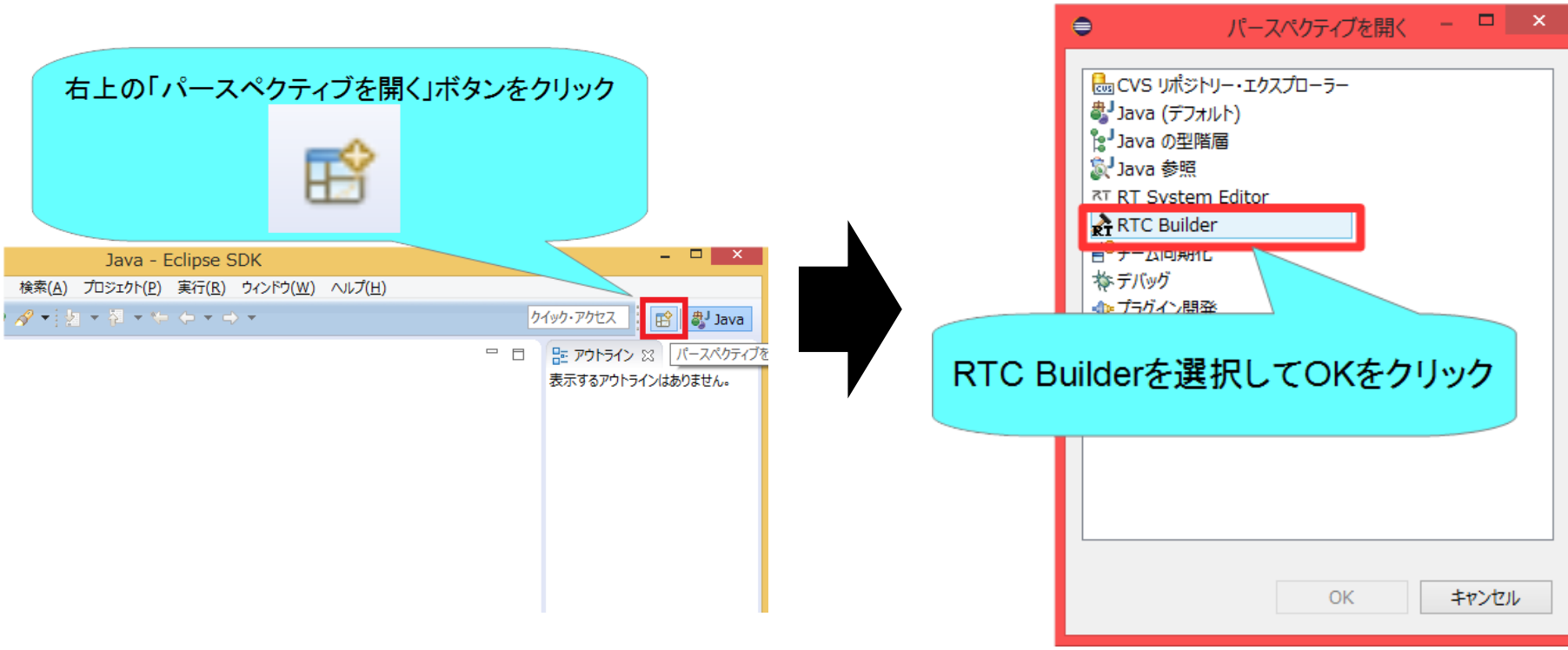
Start naming Serviceを右クリックして
「ファイルの場所を開く」を選択



RTC Builderの起動

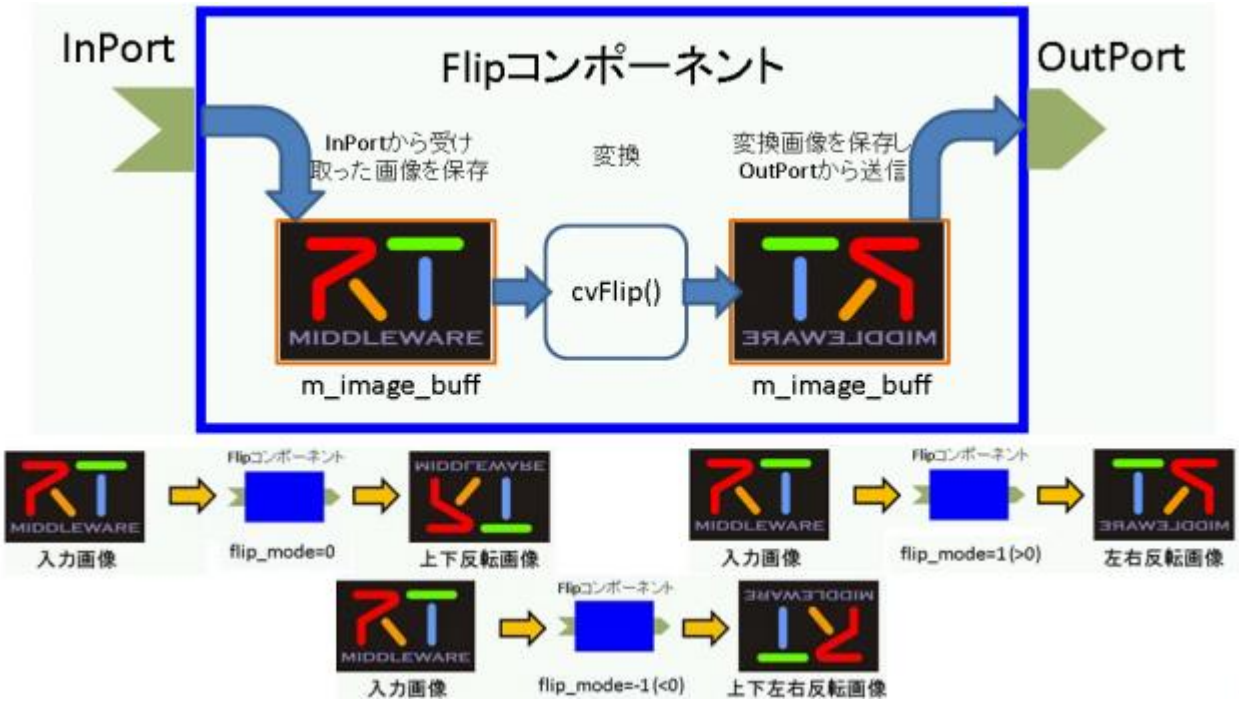


RTC Builderの起動



プロジェクト作成

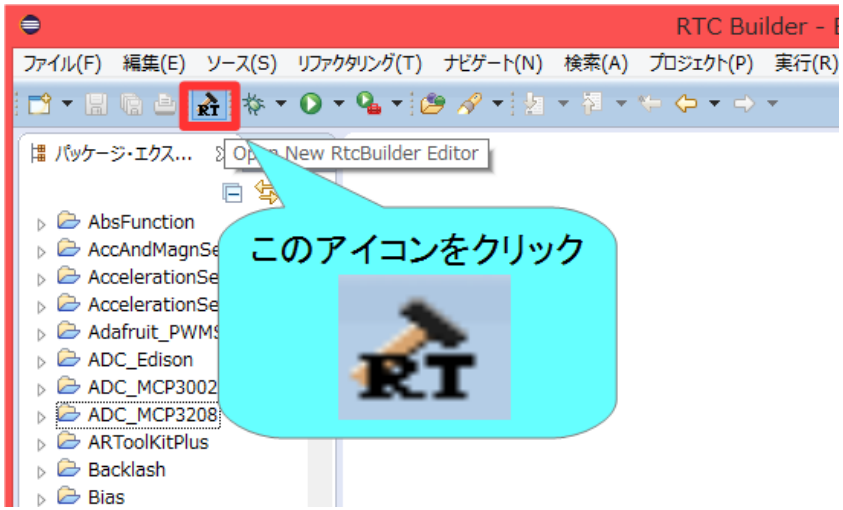
- Flipコンポーネントのスケルトンコードを作成する。
  - 画像の反転を行うコンポーネント
    - InPortで受信した画像データを処理してOutPortから出力
    - コンフィギュレーションパラメータにより反転する方向を設定
    - RT System Editorにより他のRTCと接続、RTCをアクティブ化

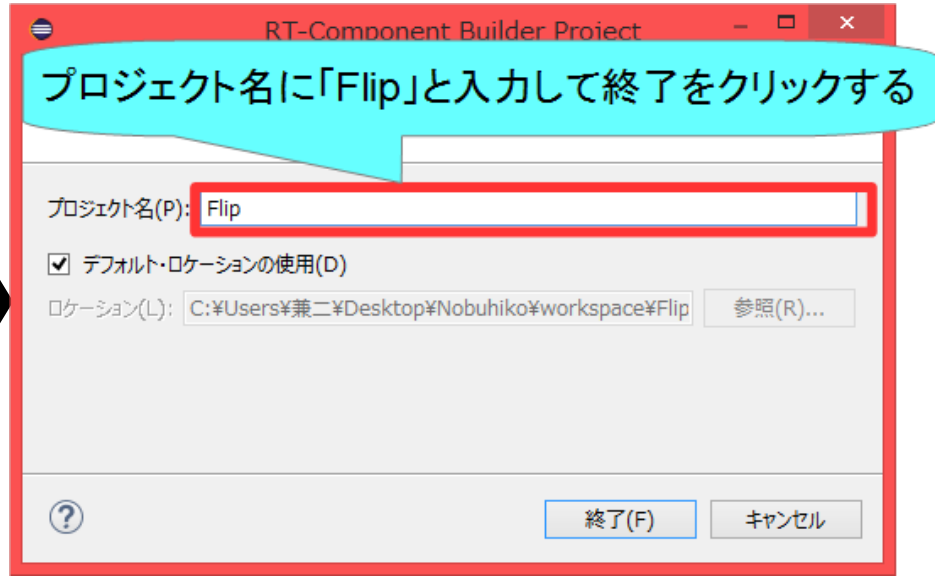


資料

- 右図のようにOpenRTM-aist公式サイトからページを開く
- もしくは配布のUSBメモリのhtmlファイルを開く。
  - 「Flip」→「作成手順」→「Windows」or「Ubuntu」→「画像処理コンポーネントの作成～.html」
- Flipコンポーネントのソースコードは USBメモリの以下のフォルダに同梱してあります。
  - 「Flip」→「ソースコード」

プロジェクト作成

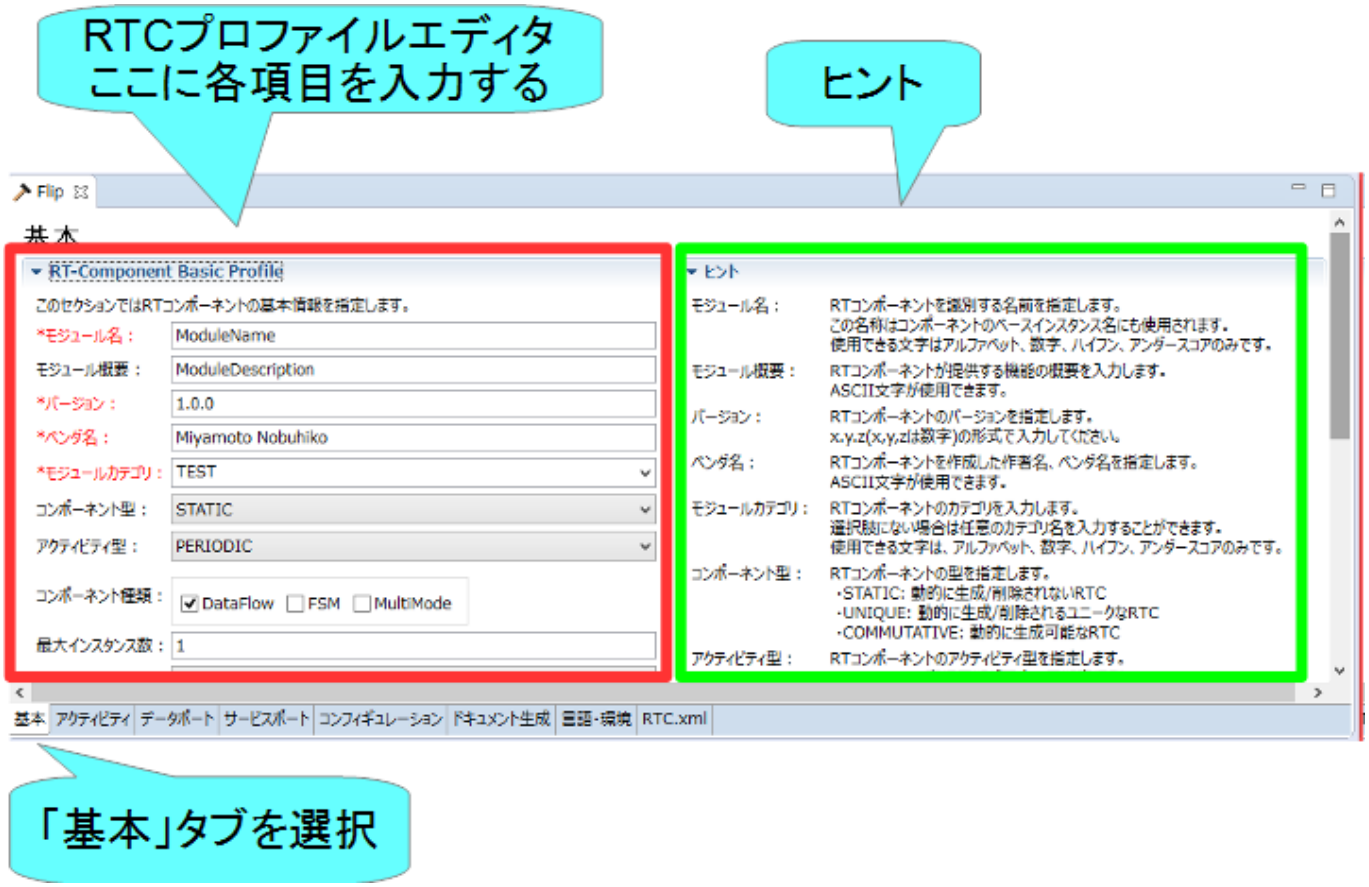




- Eclipse起動時にワークスペースに指定したディレクトリに「Flip」というフォルダが作成される
  - この時点では「RTC.xml」と「.project」のみが生成されている
- 以下の項目が設定する
  - 基本プロファイル
  - アクティビティ・プロファイル
  - データポート・プロファイル
  - サービスポート・プロファイル
  - コンフィギュレーション
  - ドキュメント
  - 言語環境
  - RTC.xml

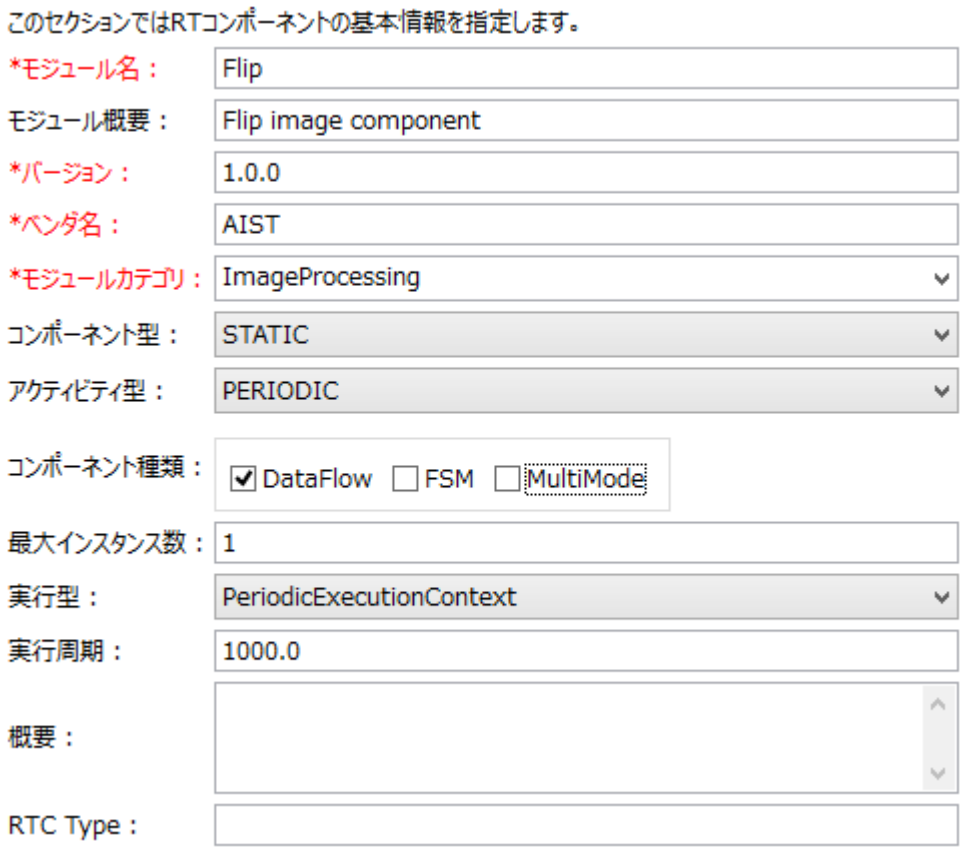
基本プロファイルの入力

- RTコンポーネントのプロファイル情報など，コンポーネントの基本情報を設定．
- コード生成，インポート/エクスポート，パッケージング処理を実行



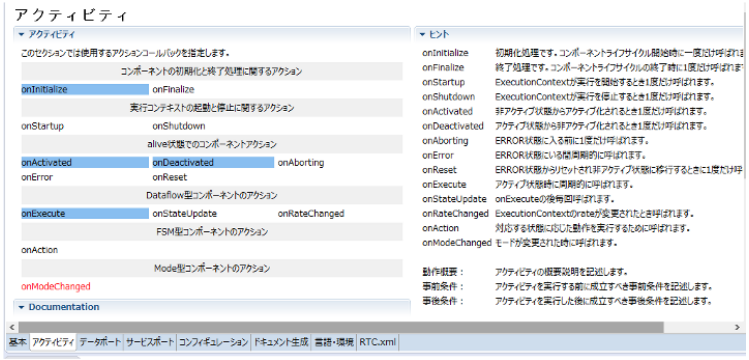
基本プロファイルの入力

- モジュール名
  - Flip
- モジュール概要
  - 任意(Flip image 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(ImageProcessing)
- コンポーネント型
  - STATIC
- アクティビティ型
  - PERIODIC
- コンポーネントの種類
  - DataFlow
- 最大インスタンス数
  - 1
- 実行型
  - PeriodicExecutionContext
- 実行周期
  - 1000.0
- 概要
  - 任意



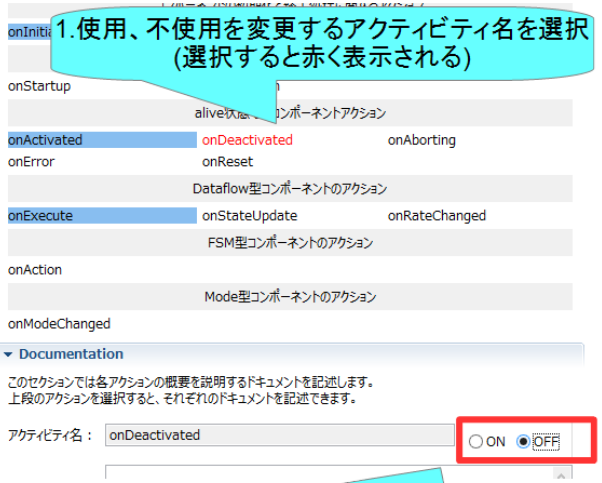
アクティビティの設定

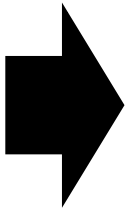
- 使用するアクティビティを設定する

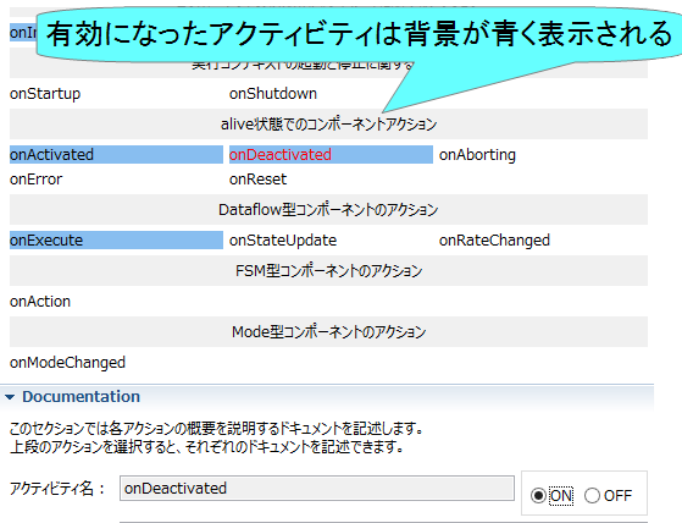


「アクティビティ」タブを選択

- 指定アクティビティを有効にする手順





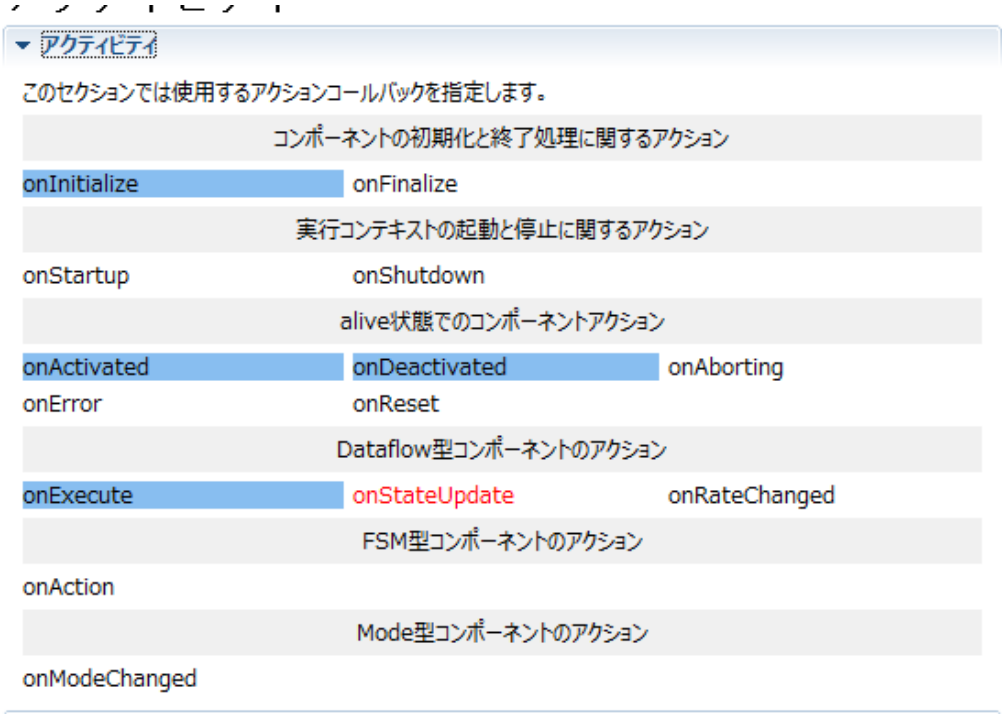


アクティビティの設定

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

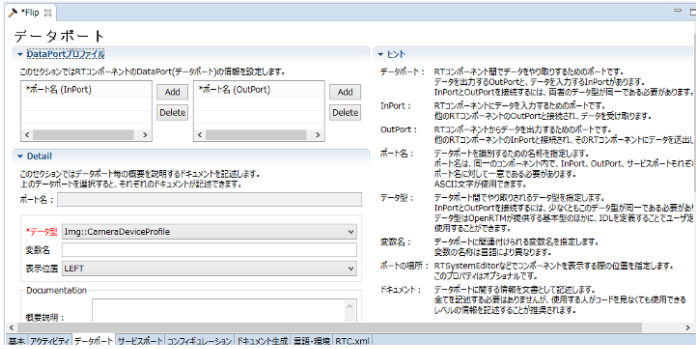
アクティビティの設定

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



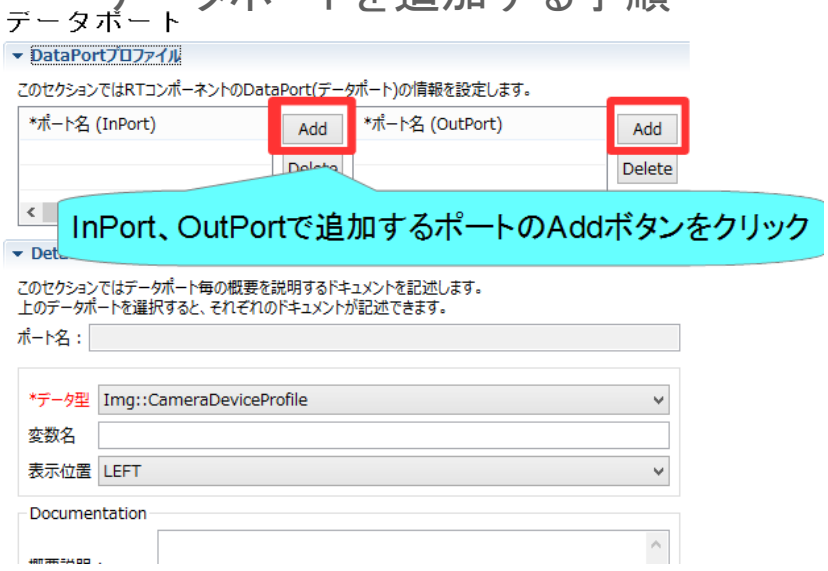
データポートの設定

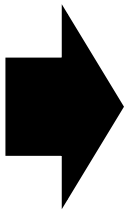
- InPort、OutPortの追加、設定を行う

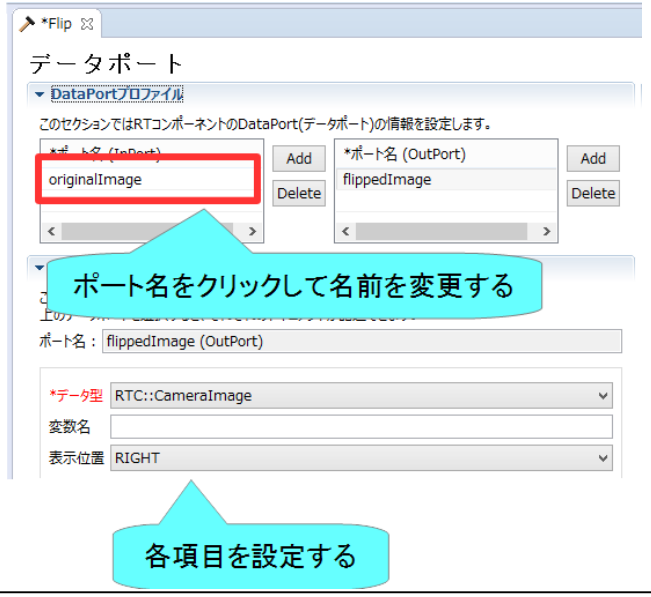


「データポート」タブを選択

- データポートを追加する手順



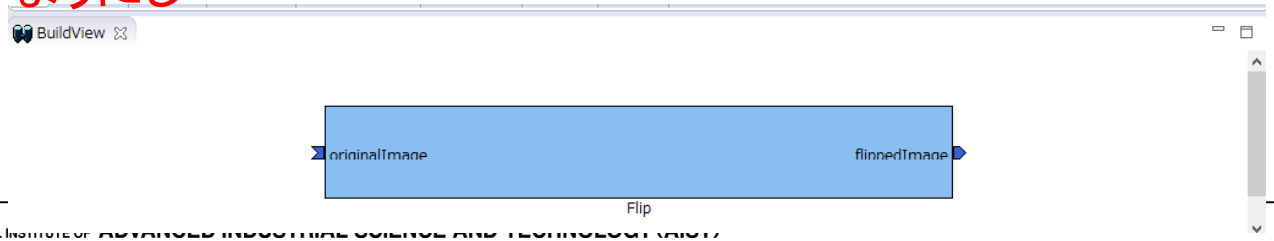




データポートの設定

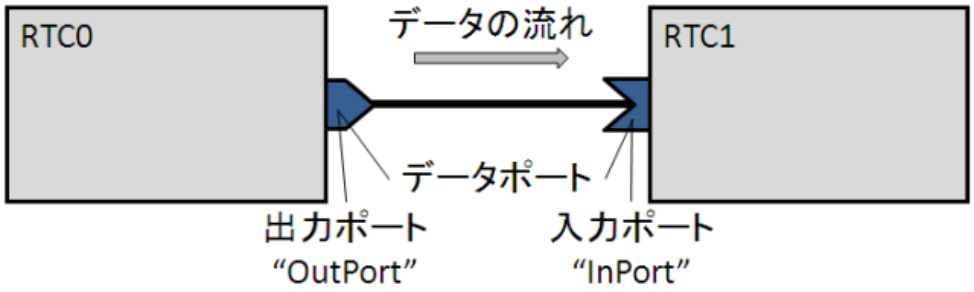
- 以下のInPortを設定する
  - originalImage
    - データ型：RTC::CameraImage
    - 他の項目は任意
- 以下のOutPortを設定する
  - flippedImage
    - データ型：RTC::CameraImage
    - 他の項目は任意

- ※今回使用するのはRTC::CameraImageなのでImg::CameraImageと間違えないようにする。
- ※ポート名を間違えないようにしてください。

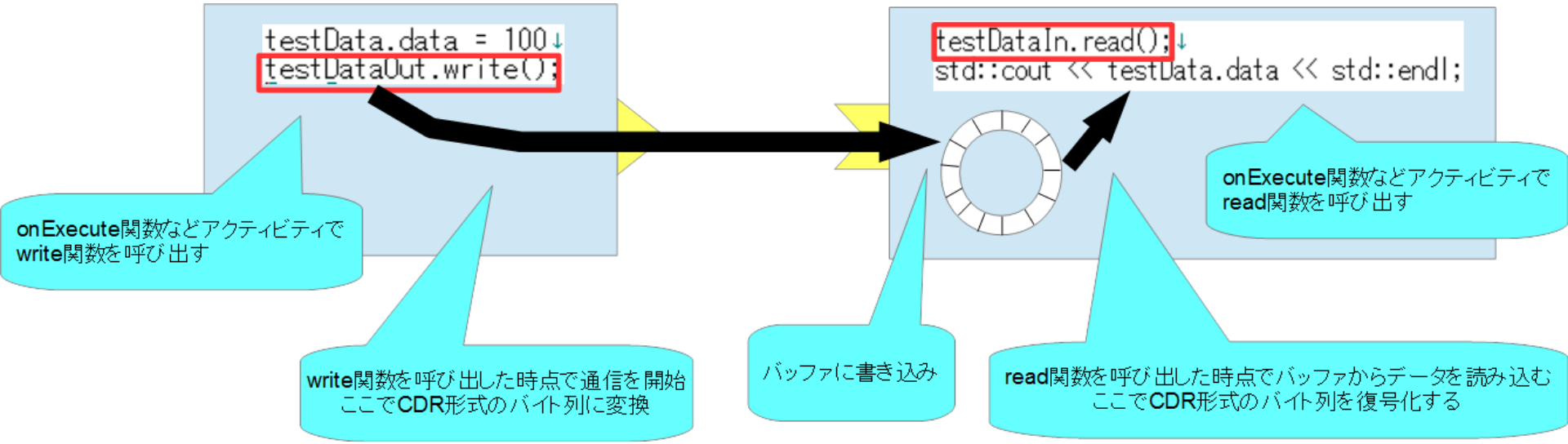


データポートについて

- 連続したデータを通信するためのポート

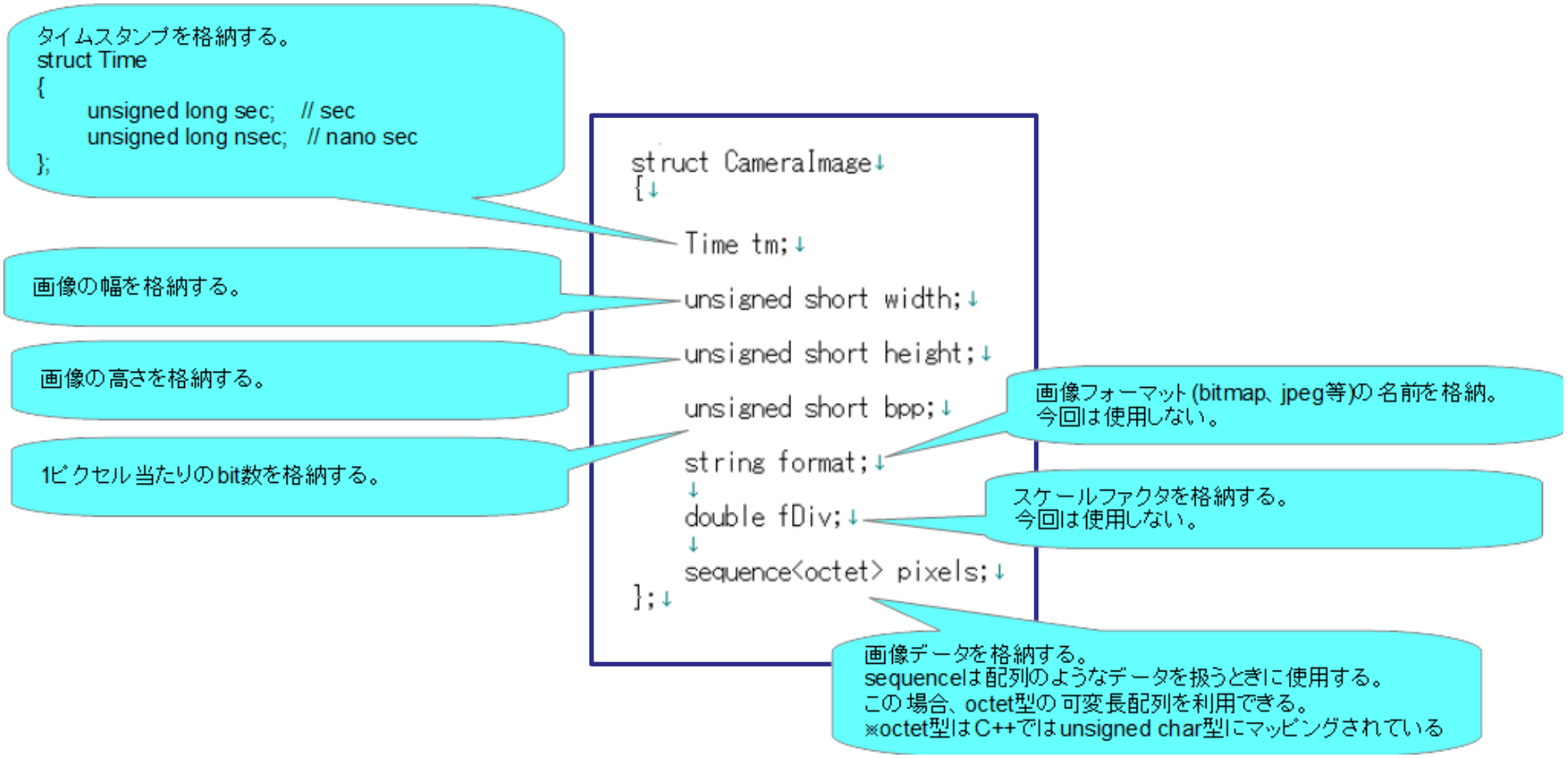


- 以下の例はデータフロー型がpush、サブスクリプション型がflush、インターフェース型がcorba_cdrの場合



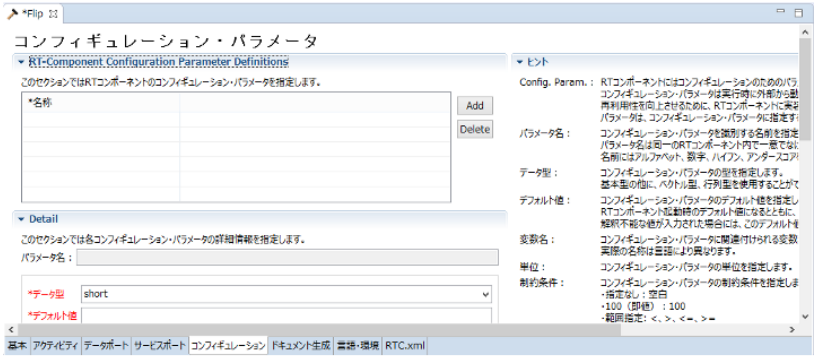
RTC::CameraImage型について

- InterfaceDataTypes.idlで定義されている画像データ通信のためのデータ型



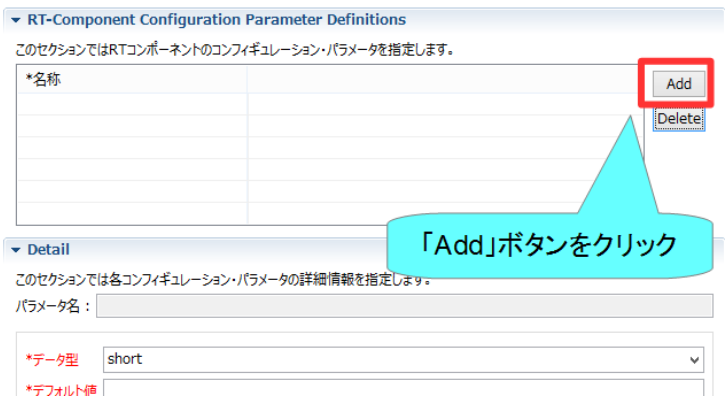
コンフィギュレーションの設定

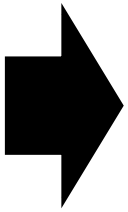
- コンフィギュレーションパラメータの追加、設定を行う



「コンフィギュレーション」タブを選択

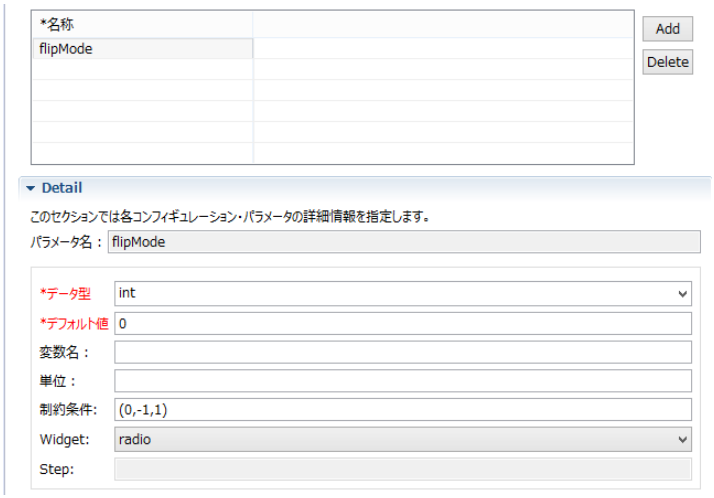
- コンフィギュレーションパラメータを追加する手順

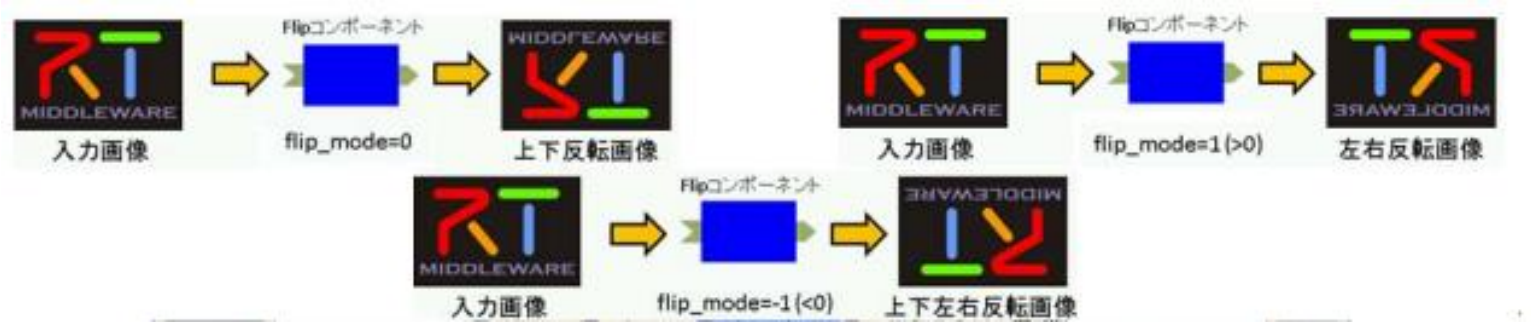




コンフィギュレーションの設定

- 以下のコンフィギュレーションパラメータを設定する
  - flipMode
    - データ型：int
    - デフォルト値：0
    - 制約条件：(0,-1,1)
    - Widget：radio
    - 他の項目は任意





- 反転する方向を設定可能にする

コンフィギュレーションパラメータの制約、Widgetの設定

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

- Widget：text

- 制約条件：0<=x<=100
- Widget：spin
- Step：10

- 制約条件：0<=x<=100
- Widget：slider
- Step：10

コンフィギュレーションパラメータの制約、Widgetの設定

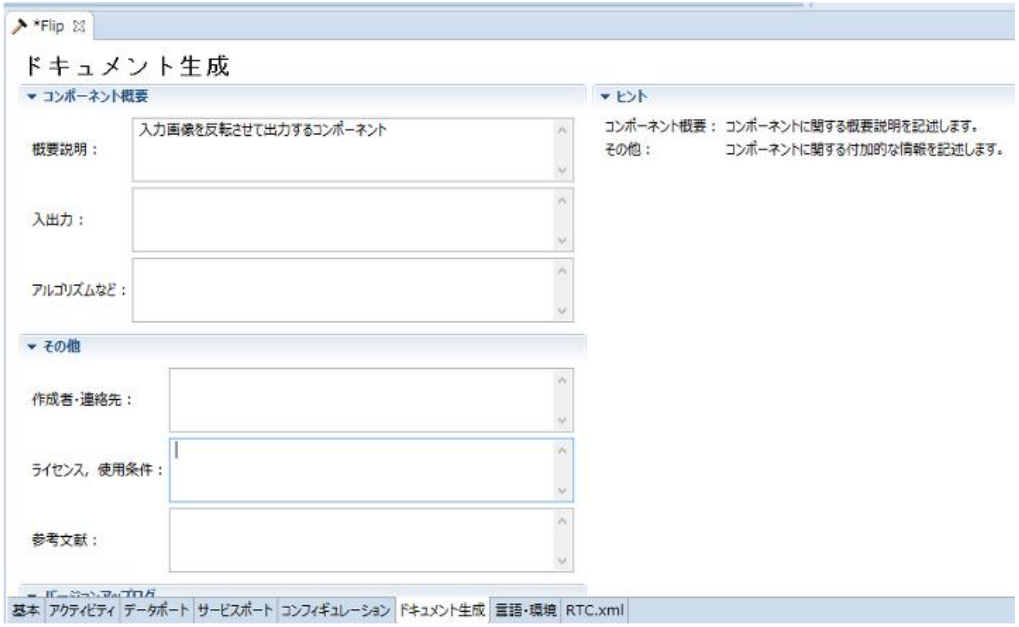
- 制約条件：(0,1,2,3)
- Widget：radio

- 制約条件：(0,1,2,3)
- Widget：checkbox

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

ドキュメントの設定

- 各種ドキュメント情報を設定

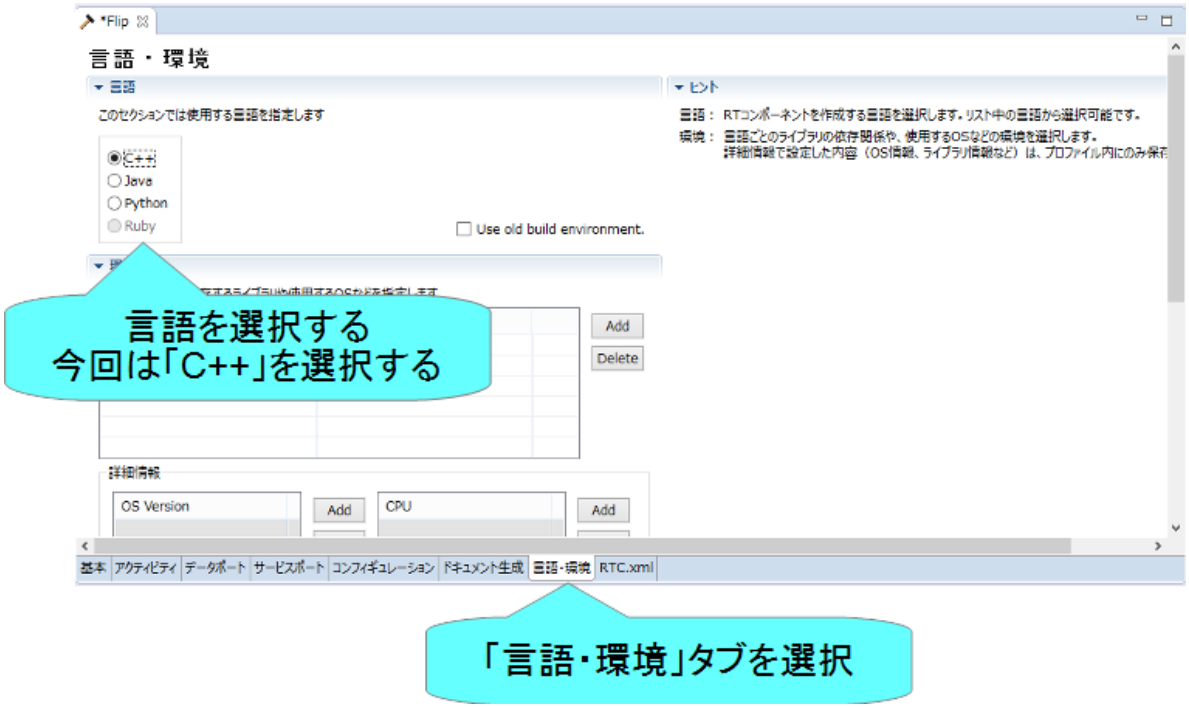


「ドキュメント生成」タブを選択

- 今回は適当に設定しておいてください。
  - 空白でも大丈夫です

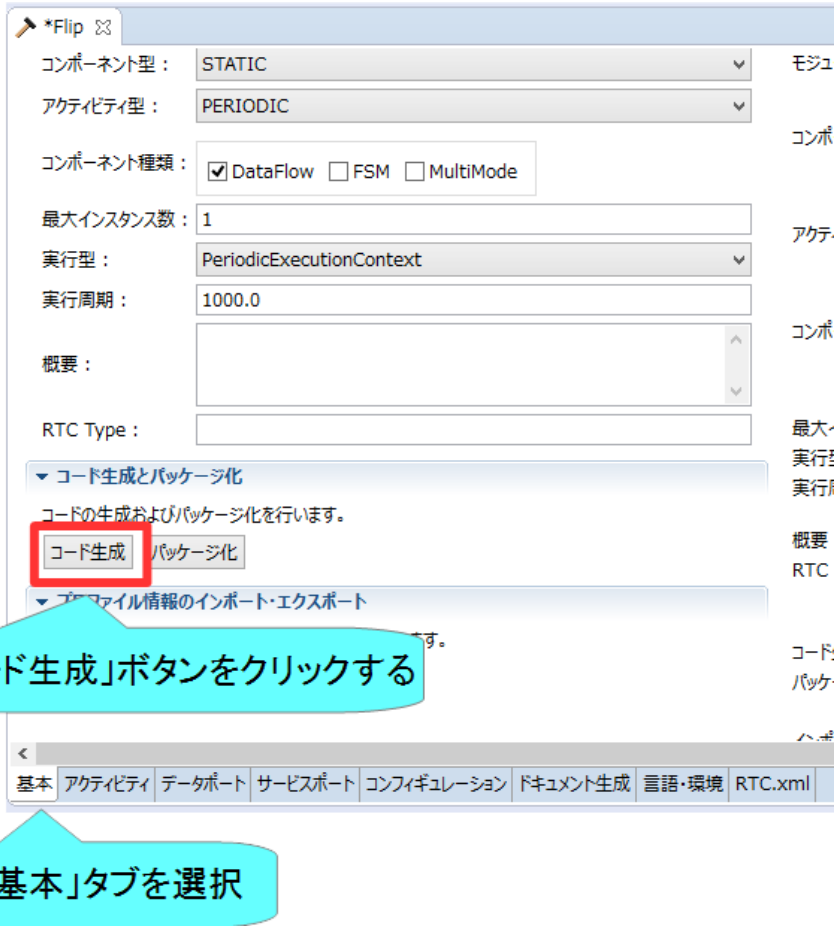
言語の設定

- 実装する言語，動作環境に関する情報を設定



スケルトンコードの生成

- 基本タブからコード生成ボタンを押すことでスケルトンコードが生成される
  - Workspace¥Flip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画像を反転させる処理を記述する
    - CMakeの設定ファイル
      - CMakeLists.txt
    - rtc.conf、Flip.conf
    - 以下略
  - ファイルが生成できているかを確認してください



ソースコードの編集、RTCのビルド

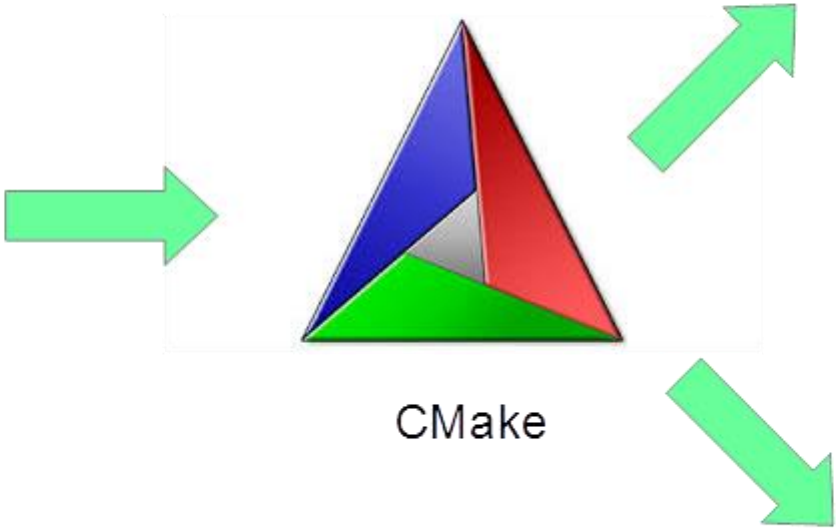
手順

- ビルドに必要な各種ファイルを生成
  - CMakeLists.txtの編集
  - CMakeにより各種ファイル生成
- ソースコードの編集
  - Flip.hの編集
  - Flip.cppの編集
- ビルド
  - Windows：Visual Studio
  - Ubuntu：Code::Blocks

CMake

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

CMakeLists.txt



CMake

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

Makefile

CMakeLists.txtの編集

- OpenCVを利用するためにCMakeLists.txtを修正する
  - worksapce¥Flipの**srcフォルダ**の**CMakeLists.txt**をメモ帳などで開いて編集する

```
1  set(comp_srcs Flip.cpp )
2  set(standalone_srcs FlipComp.cpp)
3
4  find_package(OpenCV REQUIRED)
5
6  if (DEFINED OPENRTM_INCLUDE_DIRS)
7    string(REGEX REPLACE "-I" "" ";"
8      OPENRTM_INCLUDE_DIRS "${OPENRTM_INCLUDE_DIRS}")
```

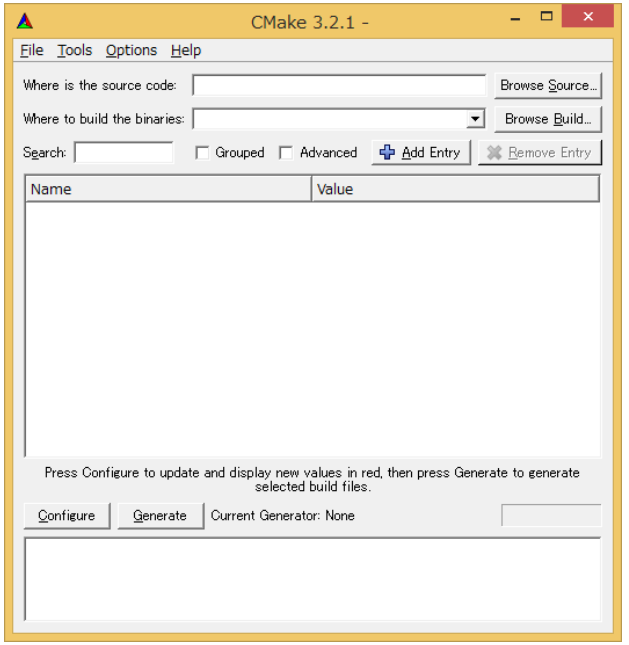
find_package(OpenCV REQUIRED)
を追加

```
47 endif(NOT TARGET ALL_IDL_TGT)
48 add_dependencies(${PROJECT_NAME} ALL_IDL_TGT)
49 target_link_libraries(${PROJECT_NAME} ${OPENRTM_LIBRARIES} ${OpenCV_LIBS})
50
51 add_executable(${PROJECT_NAME}Comp ${standalone_srcs}
52   ${comp_srcs} ${comp_headers} ${ALL_IDL_SRCS})
53 target_link_libraries(${PROJECT_NAME}Comp ${OPENRTM_LIBRARIES} ${OpenCV_LIBS})
54
```

target_link_librariesの引数に${OpenCV_LIBS}を追加
※2箇所あるので注意

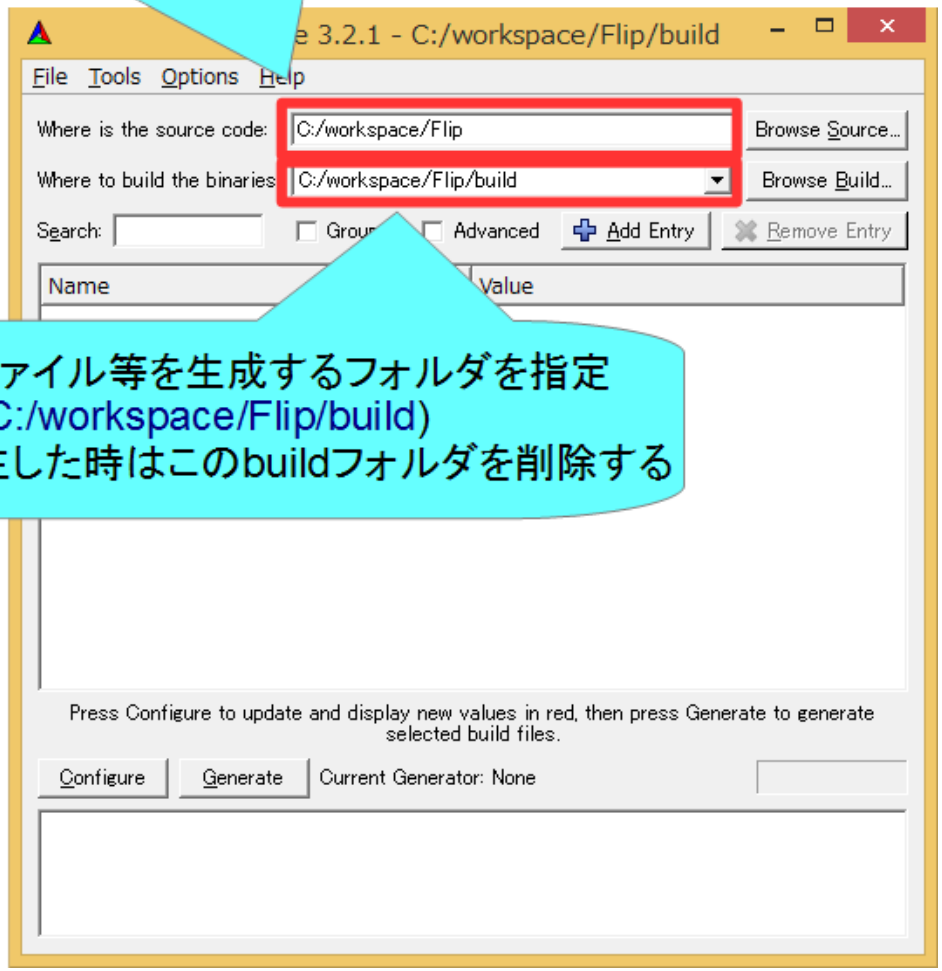
ビルドに必要なファイルの生成

- CMakeを使用する
  - Windows 7
    - 「スタート」→「すべてのプログラム」→「CMake 3.5.2」→「CMake (cmake-gui)」
  - Windows 8.1
    - 「スタート」→「アプリビュー(右下矢印)」→「CMake 3.5.2」→「CMake (cmake-gui)」
  - Ubuntu
    - コマンドで「cmake-gui」を入力



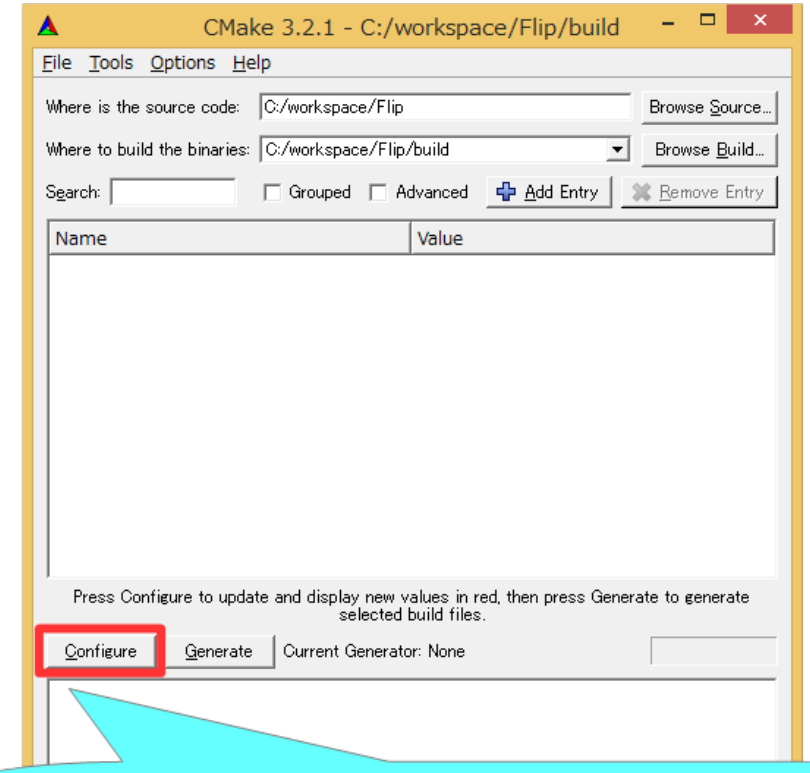
ビルドに必要なファイルの生成

RTC Builderで生成したプロジェクトのフォルダを指定
(例：C:/workspace/Flip)

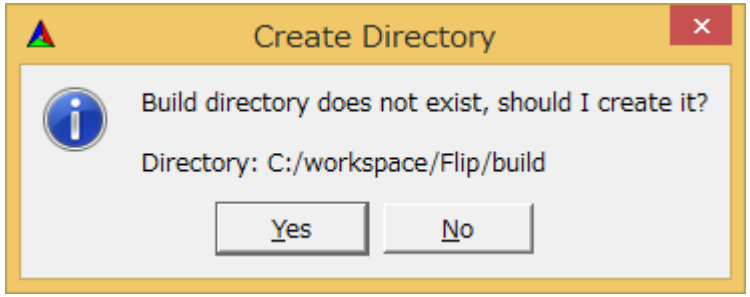


ソリューションファイル等を生成するフォルダを指定
(例：C:/workspace/Flip/build)
何かトラブルが発生した時はこのbuildフォルダを削除する

ビルドに必要なファイルの生成



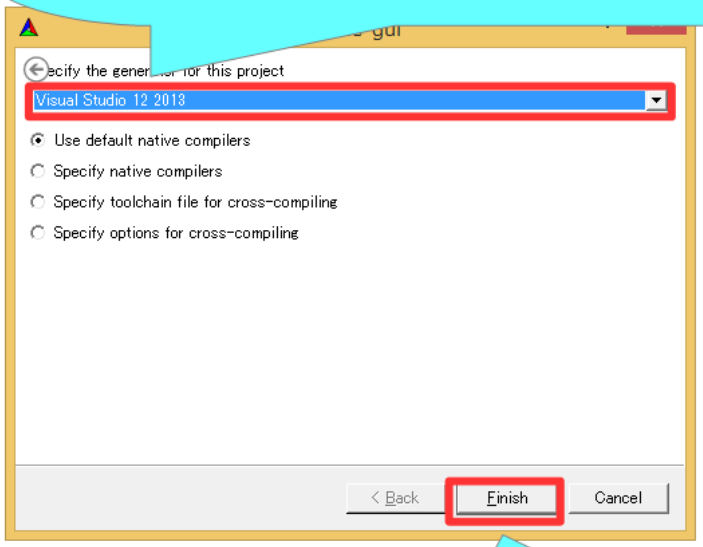
Configureボタンを押す
コンパイルに必要な情報の収集(必要なライブラリの検出など)を行う



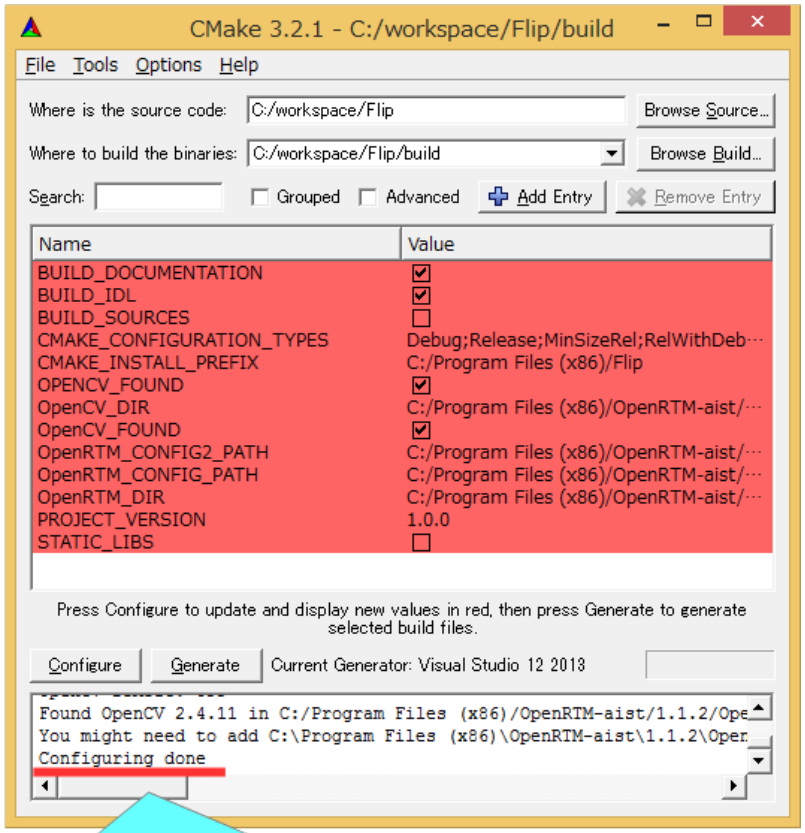
buildフォルダが存在しない場合は
作成するかどうかきかれるため「Yes」を選択

ビルドに必要なファイルの生成

ビルド環境の設定
Visual Studio 2010 32bit → Visual Studio 10 2010
Visual Studio 2012 32bit → Visual Studio 11 2012
Visual Studio 2012 64bit → Visual Studio 11 2012 Win 64
Visual Studio 2013 32bit → Visual Studio 12 2013
Visual Studio 2013 64bit → Visual Studio 12 2013 Win 64
Code::Blocks → CodeBlocks -Unix Makefiles

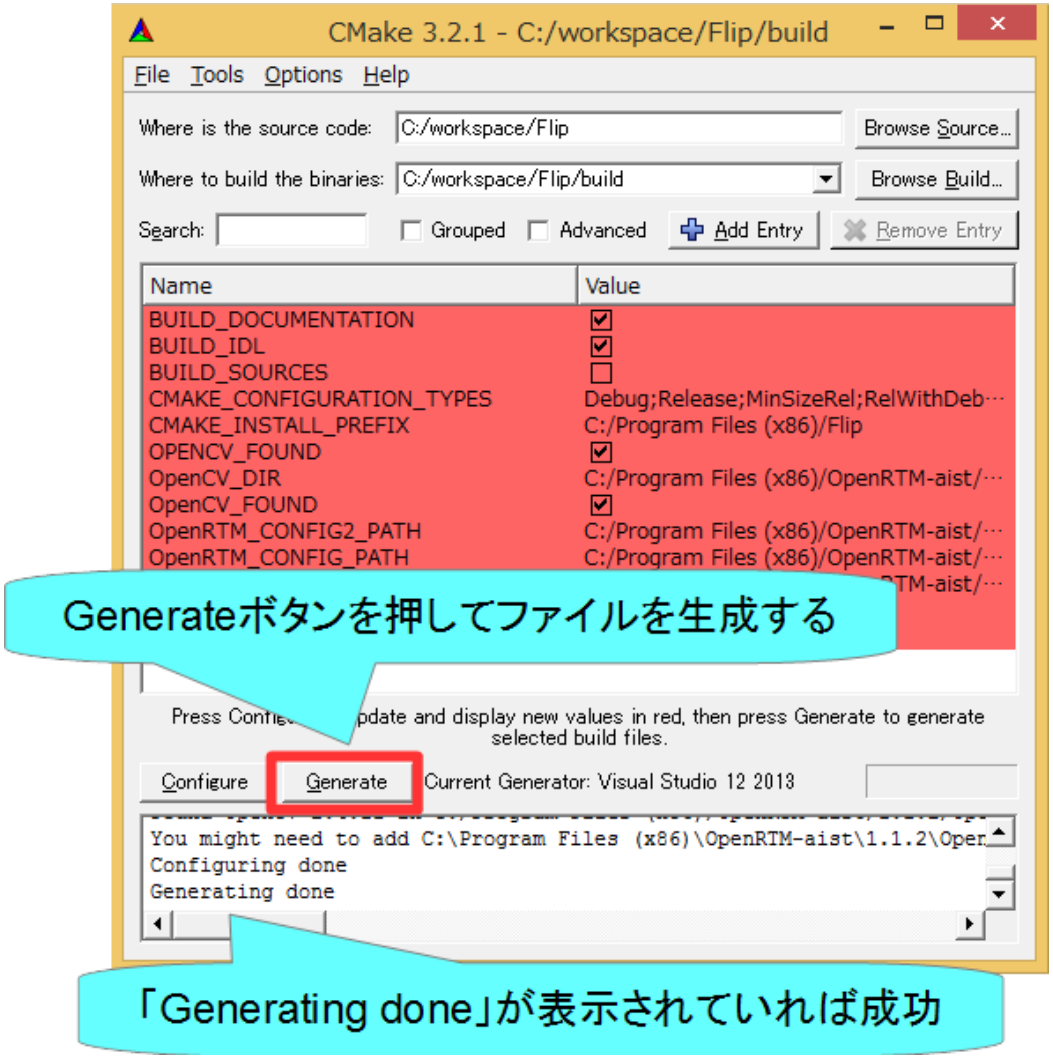


設定後、Finishボタンを押す



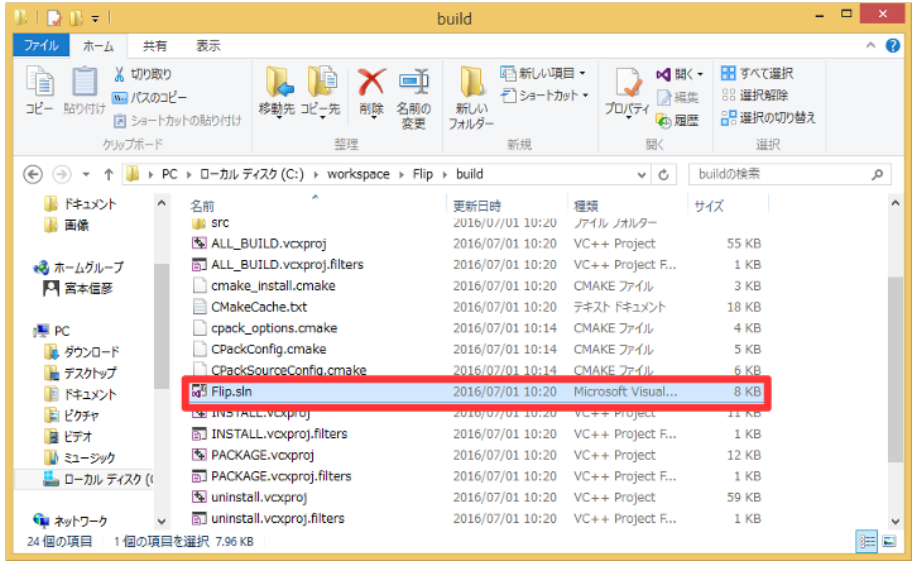
「Configure done」が表示されていれば成功

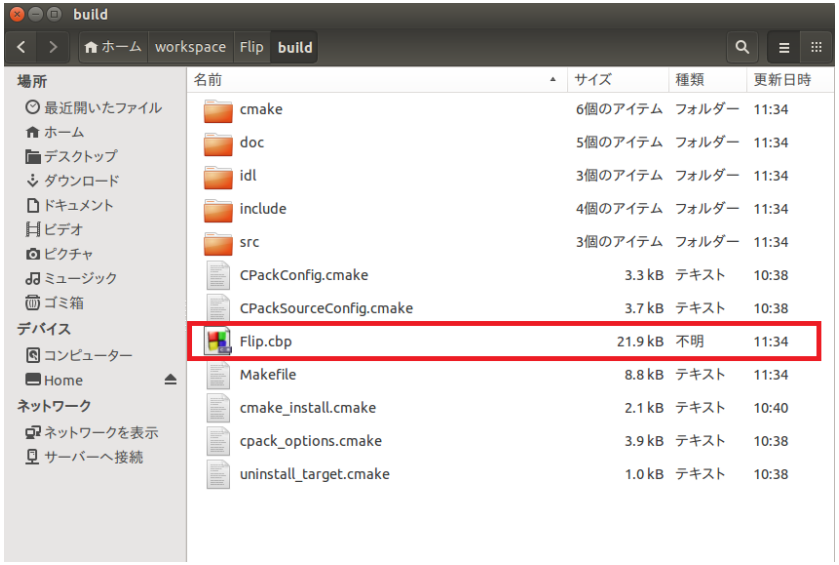
ビルドに必要なファイルの生成



ソースコードの編集

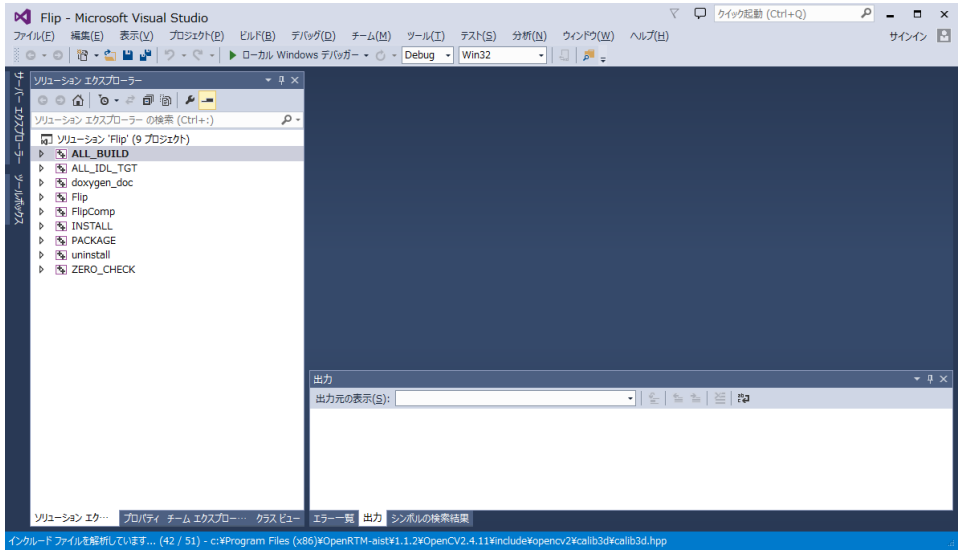
- Windows
  - buildフォルダの「Flip.sln」をダブルクリックして開く
- Ubuntu
  - buildフォルダの「Flip.cbp」をダブルクリックして開く

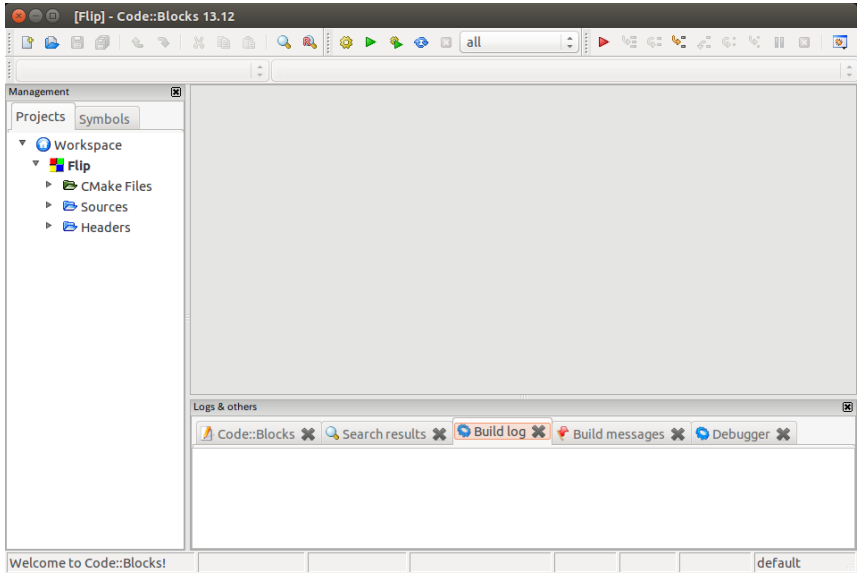




ソースコードの編集

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

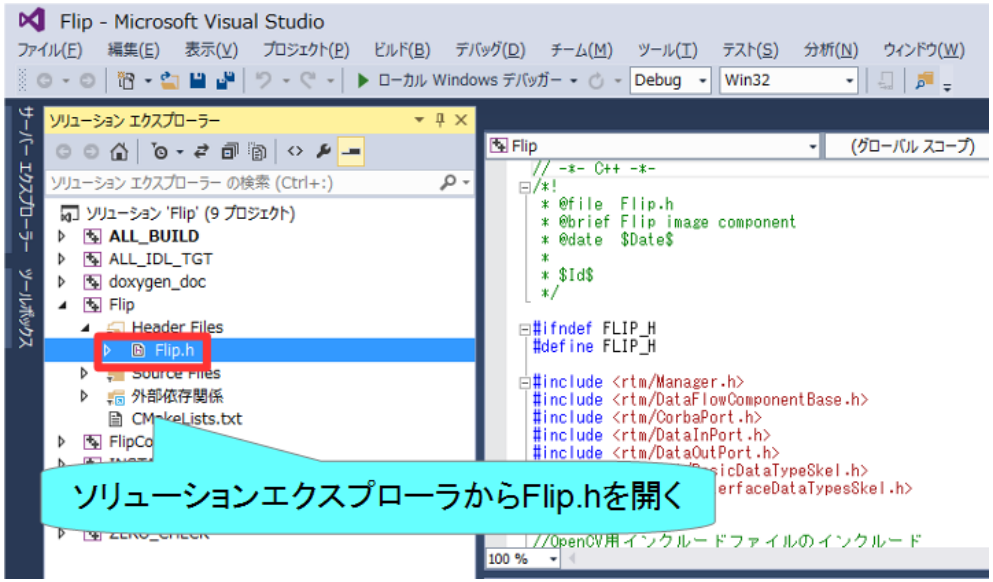




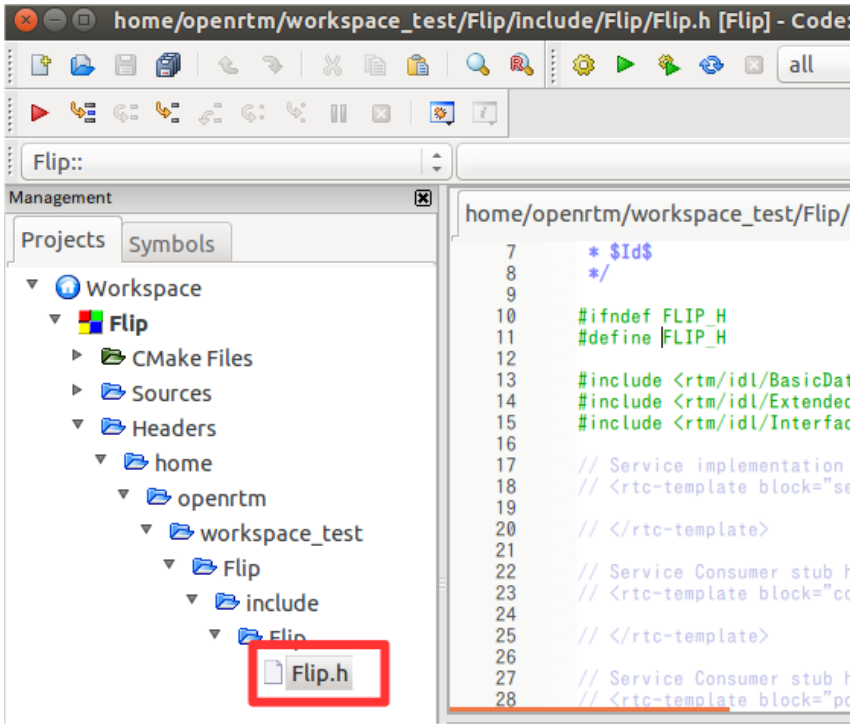
ソースコードの編集

- Flip.hの編集

Visual Studio



Code::Blocks



ソースコードの編集

- Flip.hの編集

```
37  #include <rtm/CorbaPort.h>
38  #include <rtm/DataInPort.h>
39  #include <rtm/DataOutPort.h>
40
41  //OpenCV用インクルードファイルのインクルード
42  #include <opencv2/opencv.hpp>
43
44  using namespace RTC;
45
```

OpenCVのヘッダーファイルをインクルードする
#include <opencv2/opencv.hpp>

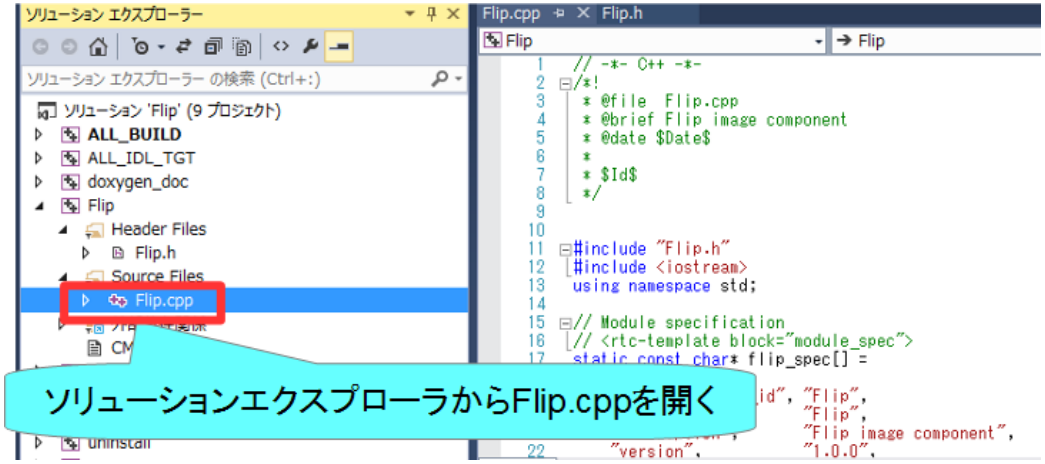
```
272  private:
273    // <rtc-template block="private_attribute">
274
275    // </rtc-template>
276
277    // <rtc-template block="private_operation">
278
279    // </rtc-template>
280      cv::Mat m_imageBuff;
281      cv::Mat m_flipImageBuff;
282
283  };
```

変数の宣言
cv::Mat m_imageBuff;
cv::Mat m_flipImageBuff;

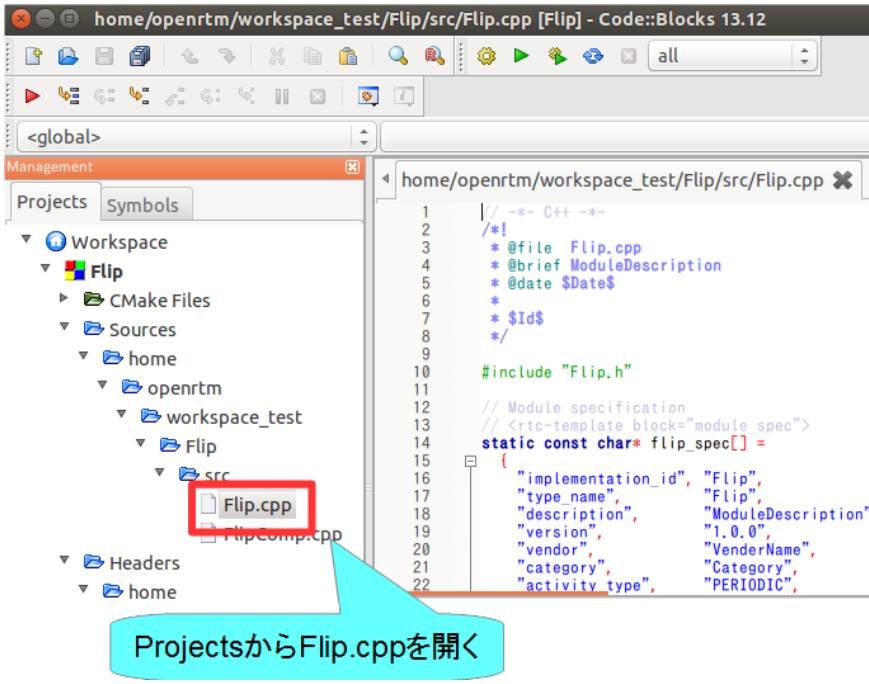
ソースコードの編集

- Flip.cppの編集

Visual Studio



Code::Blocks



ソースコードの編集

- Flip.cppの編集

```
110 RTC::ReturnCode_t Flip::onActivated(RTC::UniqueId ec_id)
111 {
112
113     // OutPortの画面サイズの初期化
114     m_flippedImage.width = 0;
115     m_flippedImage.height = 0;
116
117     return RTC::RTC_OK;
118 }
```

onActivateに追加

```
121 RTC::ReturnCode_t Flip::onDeactivated(RTC::UniqueId ec_id)
122 {
123     if (!m_imageBuff.empty())
124     {
125         // 画像用メモリの解放
126         m_imageBuff.release();
127         m_flipImageBuff.release();
128     }
129
130     return RTC::RTC_OK;
131 }
```

onDeactivateに追加

ソースコードの編集

- Flip.cppの編集

```
RTC::ReturnCode_t Flip::onExecute(RTC::UniqueId ec_id)
{
    // 新しいデータのチェック
    if (m_originalImageIn.isNew()) {
        // InPortデータの読み込み
        m_originalImageIn.read();

        // InPortとOutPortの画面サイズ処理およびイメージ用メモリの確保
        if (m_originalImage.width != m_flippedImage.width || m_originalImage.height != m_flippedImage.height)
        {
            m_flippedImage.width = m_originalImage.width;
            m_flippedImage.height = m_originalImage.height;

            m_imageBuff.create(cv::Size(m_originalImage.width, m_originalImage.height), CV_8UC3);
            m_flipImageBuff.create(cv::Size(m_originalImage.width, m_originalImage.height), CV_8UC3);


        }

        // InPortの画像データをm_imageBuffにコピー
        memcpy(m_imageBuff.data, (void *)&(m_originalImage.pixels[0]), m_originalImage.pixels.length());

        // InPortからの画像データを反転する。 m_flipMode 0: X軸周り, 1: Y軸周り, -1: 両方の軸周り
        cv::flip(m_imageBuff, m_flipImageBuff, m_flipMode);

        // 画像データのサイズ取得
        int len = m_flipImageBuff.channels() * m_flipImageBuff.cols * m_flipImageBuff.rows;
        m_flippedImage.pixels.length(len);

        // 反転した画像データをOutPortにコピー
        memcpy((void *)&(m_flippedImage.pixels[0]), m_flipImageBuff.data, len);

        // 反転した画像データをOutPortから出力する。
        m_flippedImageOut.write();
    }

    return RTC::RTC_OK;
}
```

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

```
// 新しいデータのチェック
if (m_originalImageIn.isNew()) {
    // InPortデータの読み込み
    m_originalImageIn.read();

    // InPortとOutPortの画面サイズ処理およびイメージ用メモリの確保
    if (m_originalImage.width != m_flippedImage.width || m_originalImage.height != m_flippedImage.height)
    {
```

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_originalImageにデータが格納される

- データを書き込む手順

変数m_flippedImageにデータを格納

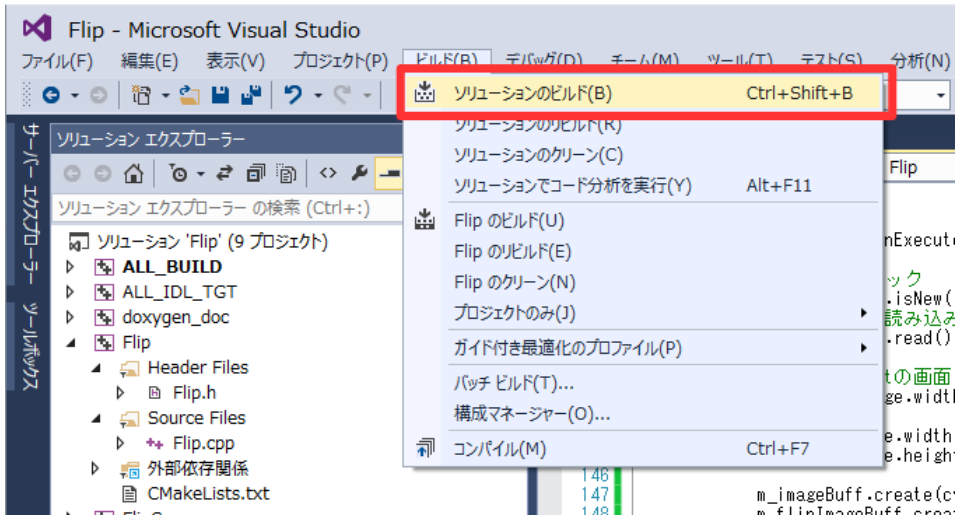
```
// 反転した画像データをOutPortにコピー
memcpy((void *)&(m_flippedImage.pixels[0]), m_flipImageBuff.data, len);

// 反転した画像データをOutPortから出力する。
m_flippedImageOut.write();
```

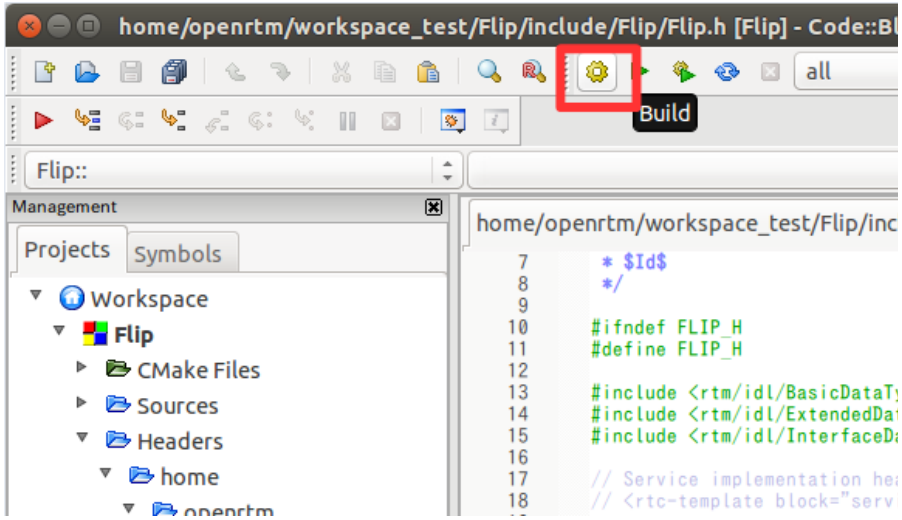
write関数でデータの書き込み

ソースコードのコンパイル

Visual Studio



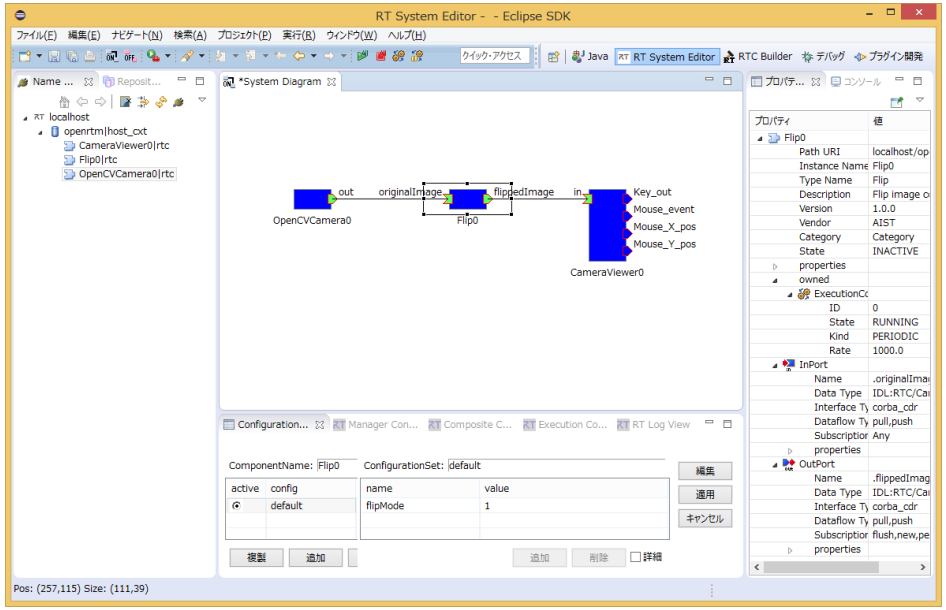
Code::Blocks



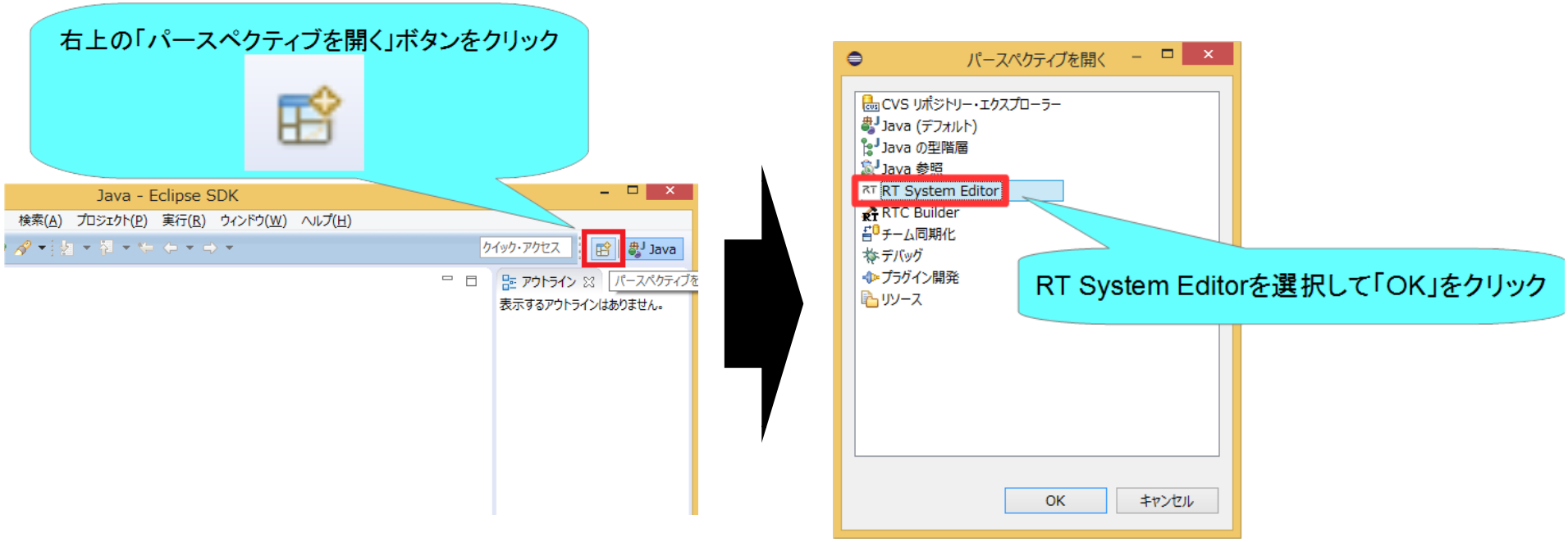
システム構築支援ツール
RT System Editorについて

RT System Editor

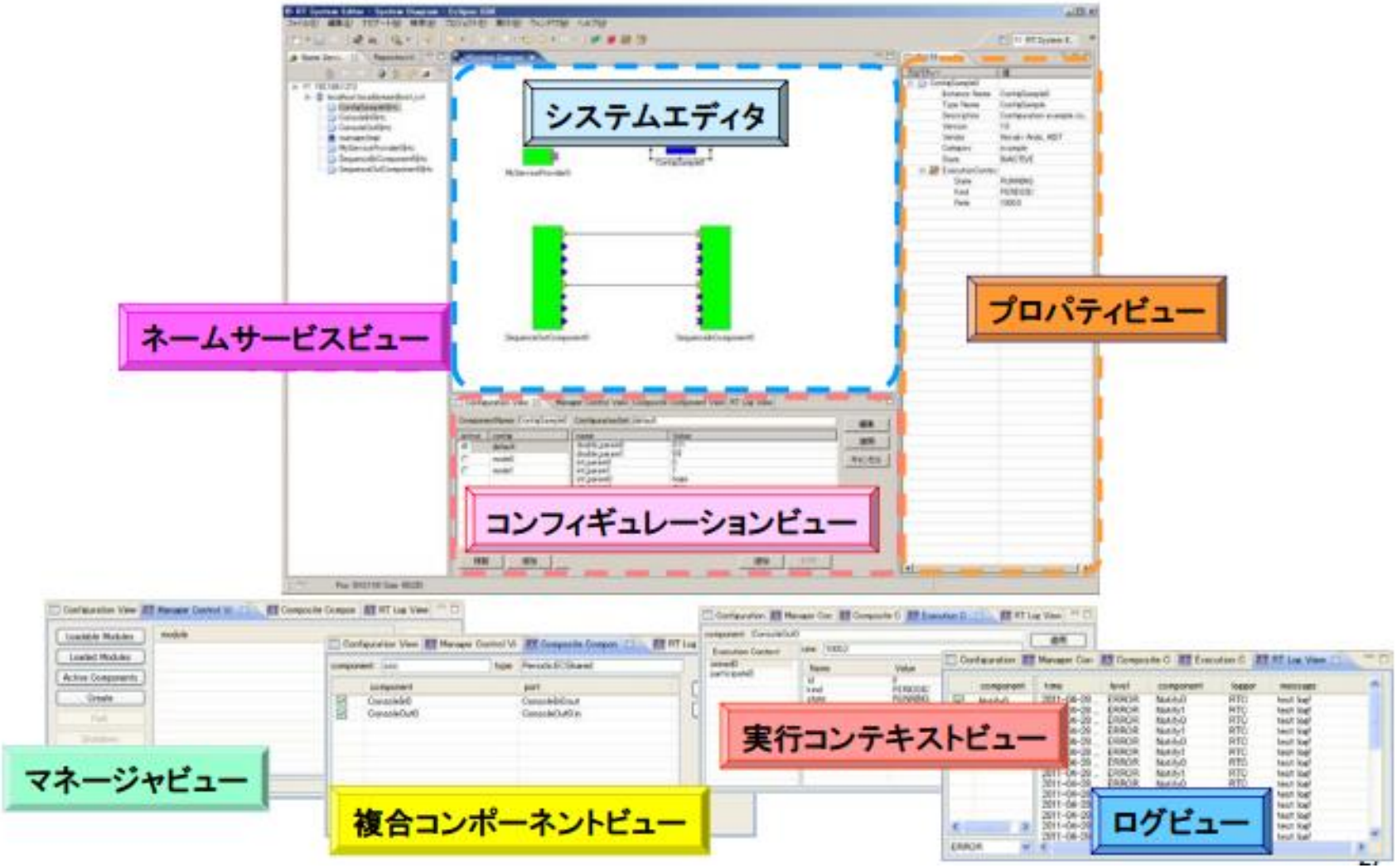
- RTCをGUIで操作するためのツール
  - データポート、サービスポートの接続
  - アクティブ化、非アクティブ化、リセット、終了
  - コンフィギュレーションパラメータの操作
  - 実行コンテキストの操作
    - 実行周期変更
    - 実行コンテキストの関連付け
  - 複合化
  - マネージャからRTCを起動
  - 作成したRTシステムの保存、復元



RT System Editorの起動

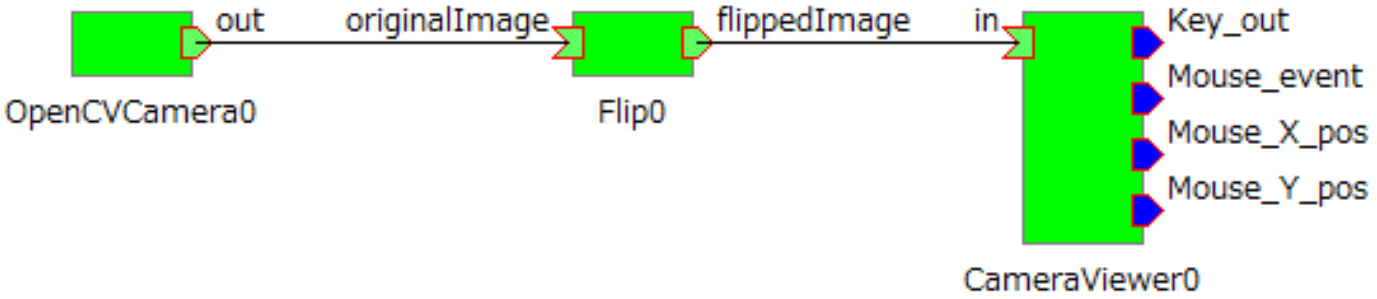


RT System Editorの画面構成



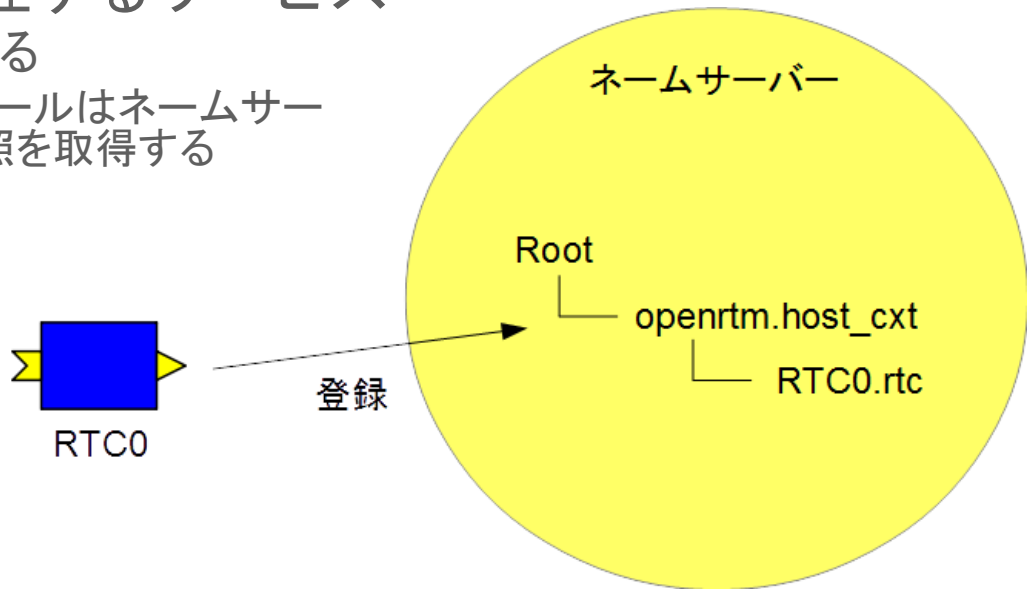
Flipコンポーネントの動作確認

- WEBカメラで撮影した画像を反転させて表示するRTシステムを作成する
  - ネームサーバーを起動する
  - CameraViewerコンポーネント、OpenCVCameraコンポーネントを起動する
    - Windows
      - 「OpenRTM-1.1.2」→「C++」→「Components」→「OpenCVExamples」
    - Ubuntu
      - $ /usr/local/share/openrtm-1.1/components/c++/opencv-rtcs/CameraViewerComp
      - $ /usr/local/share/openrtm-1.1/components/c++/opencv-rtcs/OpenCVCameraComp
  - Flip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FlipComp.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FlipCompが生成されているためこれを起動する
  - CameraViewerコンポーネント、OpenCVCameraコンポーネント、Flipコンポーネントを接続して「All Activate」を行う



ネームサーバーの起動

- オブジェクトを名前で管理するサービス
  - RTCを一意の名前で登録する
    - RTシステムエディタ等のツールはネームサーバーから名前でRTCの参照を取得する



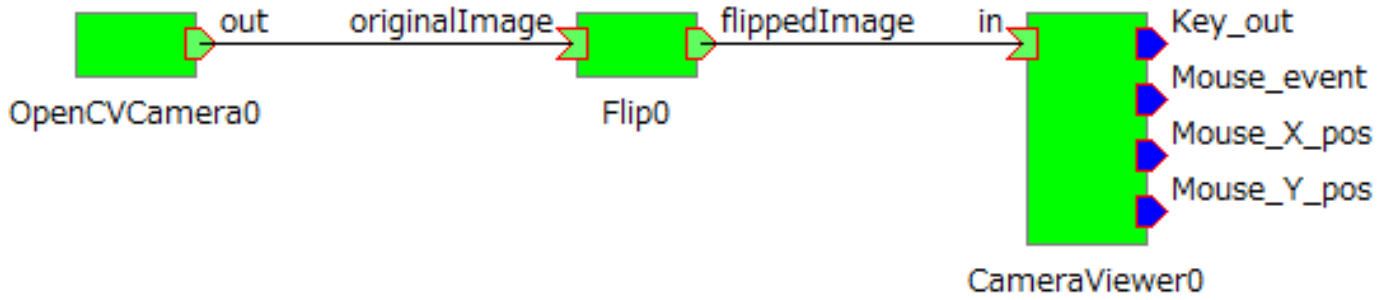
- 起動する手順
  - Windows 7
    - 「スタート」→「すべてのプログラム」→「OpenRTM-aist 1.1.2」→「Tools」→「Start Naming Service」
  - Windows 8.1
    - 「スタート」→「アプリビュー(右下矢印)」→「OpenRTM-aist 1.1.2」→「Start Naming Service」
  - Ubuntu
    - $ rtm-naming

ネームサーバーの起動

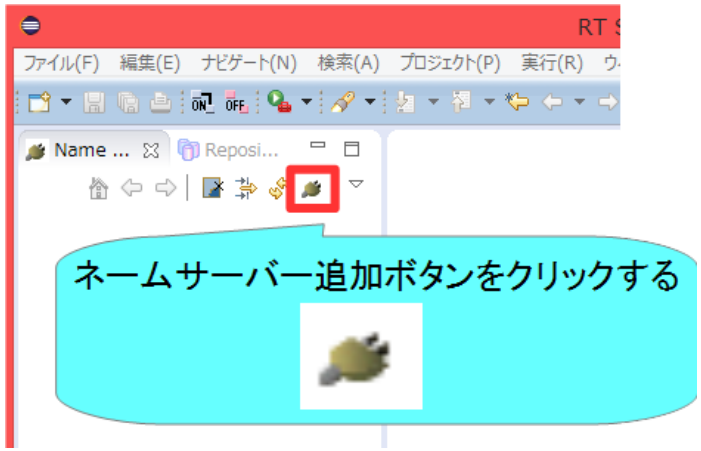
- Windows 8.1

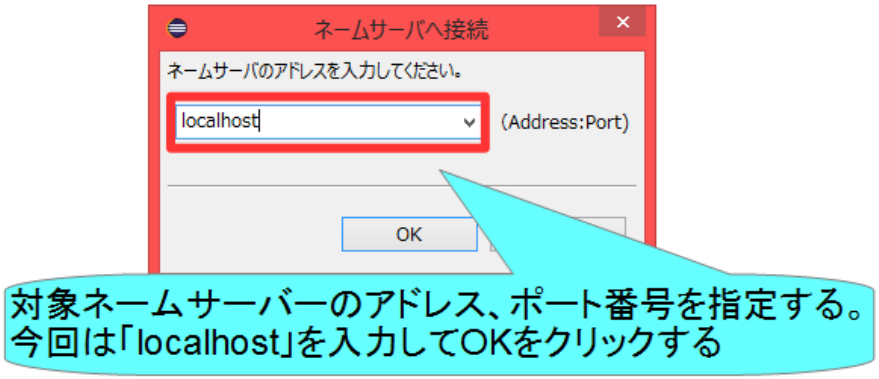
Flipコンポーネントの動作確認

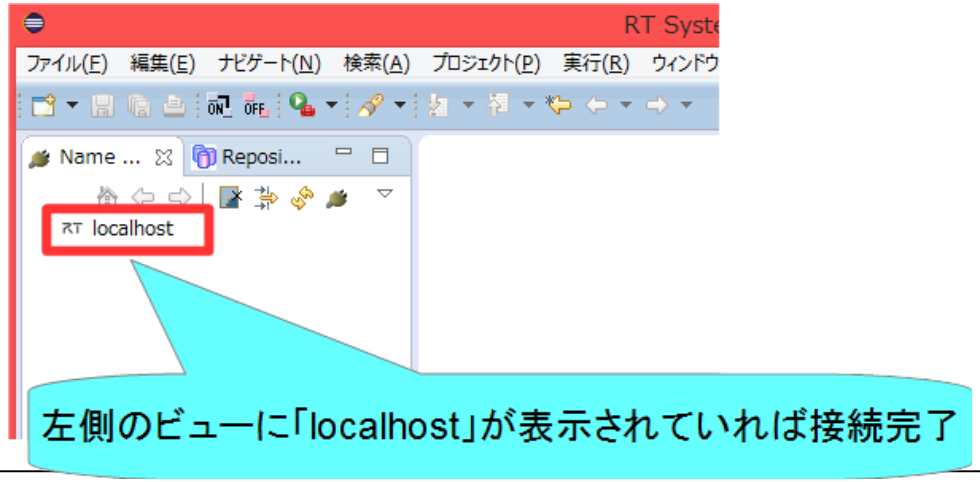
- WEBカメラで撮影した画像を反転させて表示するRTシステムを作成する
  - ネームサーバーを起動する
  - CameraViewerコンポーネント、OpenCVCameraコンポーネントを起動する
    - Windows
      - 「OpenRTM-1.1.2」→「C++」→「Components」→「OpenCVExamples」
    - Ubuntu
      - $ /usr/local/share/openrtm-1.1/components/c++/opencv-rtcs/CameraViewerComp
      - $ /usr/local/share/openrtm-1.1/components/c++/opencv-rtcs/OpenCVCameraComp
  - Flip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FlipComp.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FlipCompが生成されているためこれを起動する
  - CameraViewerコンポーネント、OpenCVCameraコンポーネント、Flipコンポーネントを接続して「All Activate」を行う



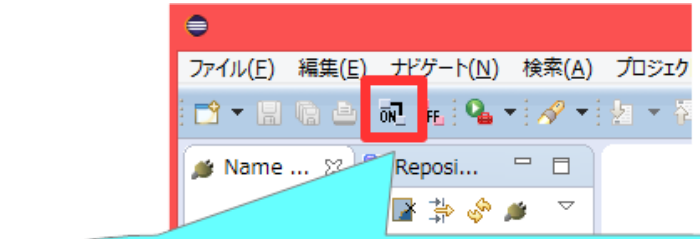
ネームサーバーへ接続


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

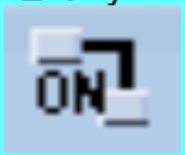

ネームサーバへ接続
ネームサーバのアドレスを入力してください。
localhost
(Address:Port)
OK
対象ネームサーバーのアドレス、ポート番号を指定する。
今回は「localhost」を入力してOKをクリックする

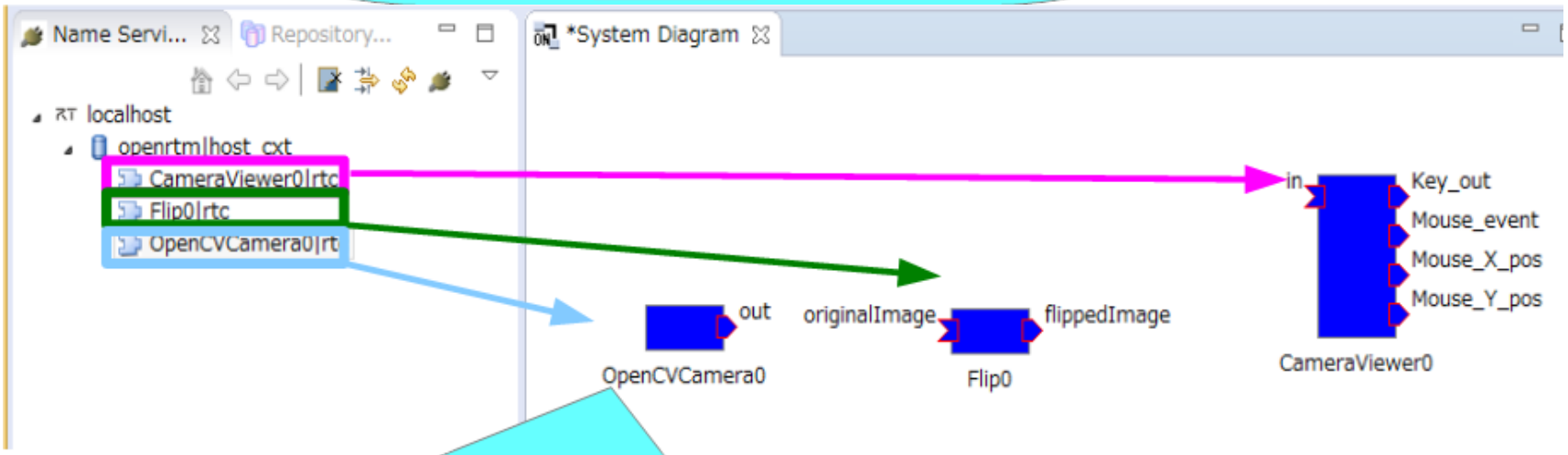

ファイル(F) 編集(E) ナビゲート(N) 検索(A) プロジェクト(P) 実行(R) ウィンドウ
Name ...
Reposi...
localhost
左側のビューに「localhost」が表示されていれば接続完了

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



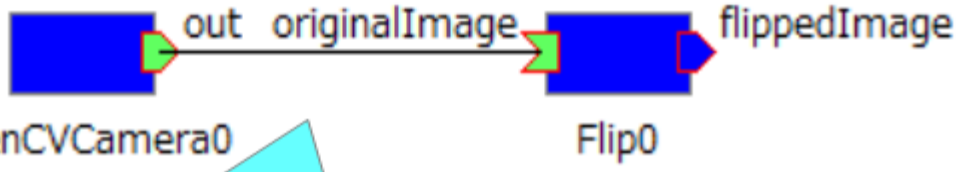


左側のネームサービスビューから
CameraViewer0.rtc、OpenCVCamera0.rtc、Flip0.rtcを
System Diagramにドラッグアンドドロップ

データポートの接続

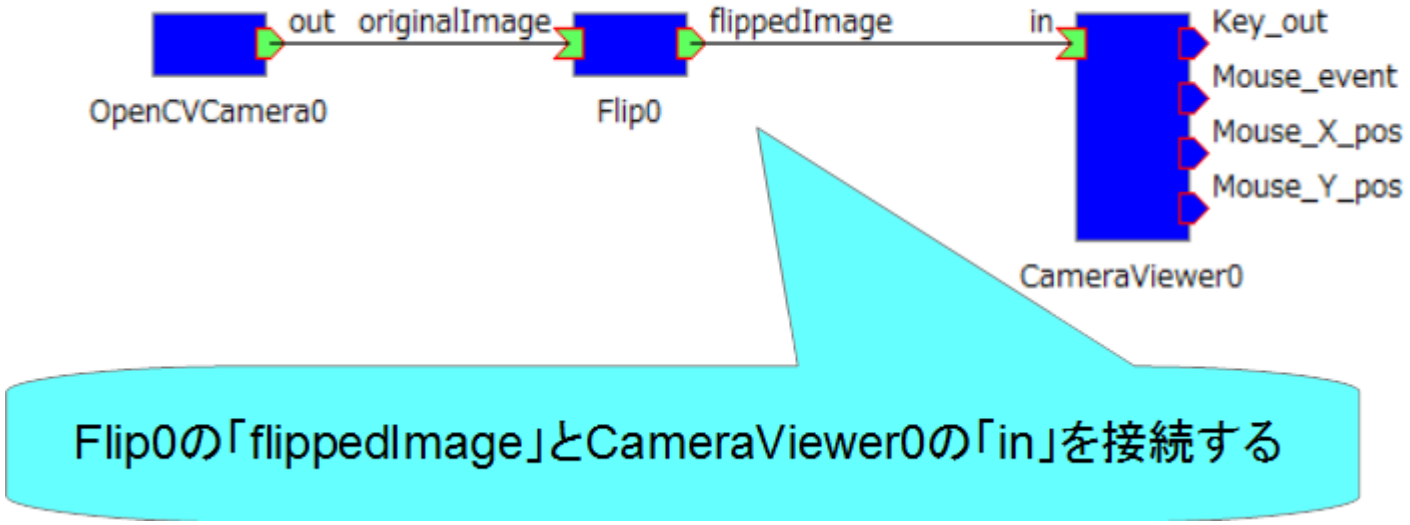
OpenCVCamera0の「out」を選択して、
Flip0の「originalImage」にドラッグアンドドロップ

そのままの設定でOKをクリック

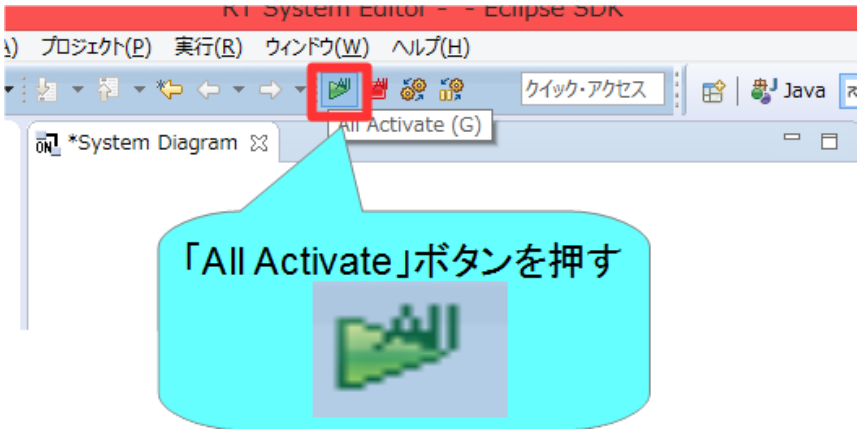


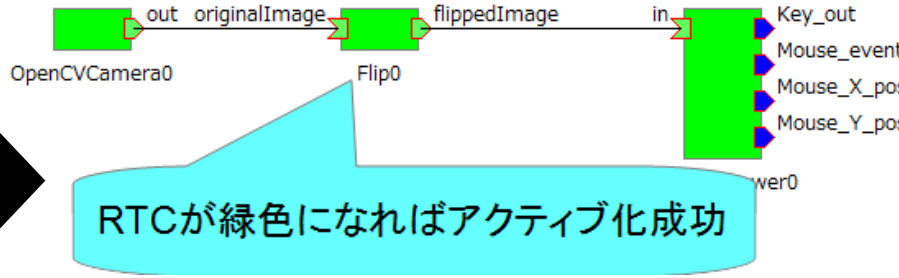
1. ポートの間に線が表示される
2. InPort、OutPortが緑色で表示される

データポートの接続


out
originalImage
flippedImage
in
Key_out
Mouse_event
Mouse_X_pos
Mouse_Y_pos
OpenCVCamera0
Flip0
CameraViewer0
Flip0の「flippedImage」とCameraViewer0の「in」を接続する

アクティブ化

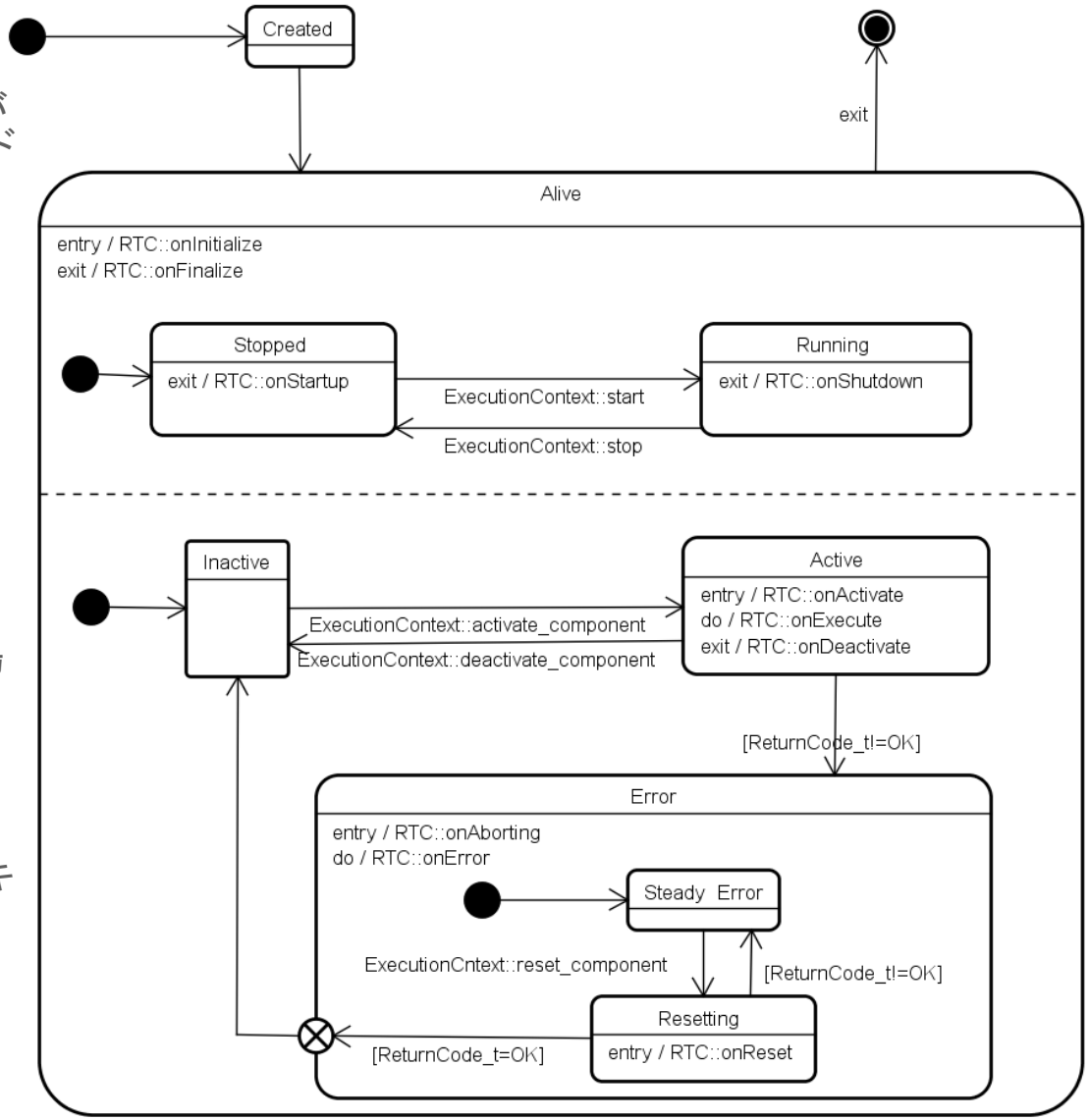




- WEBカメラで撮影した画像が反転して表示されるかを確認してください
  - 表示されない場合
    - カメラがPCに接続されていない
    - データポートを接続していない
    - RTCがアクティブになっていない

RTコンポーネントの状態遷移

- RTCには以下の状態が存在する
  - Created
    - 生成状態
    - 実行コンテキストを生成し、start()が呼ばれて実行コンテキストのスレッドが実行中(Runnning)状態になる
    - 自動的にInactive状態に遷移する
  - Inactive
    - 非活性状態
    - activate_componentメソッドを呼び出すと活性状態に遷移する
    - RT System Editor上での表示は青
  - Active
    - 活性状態
    - onExecuteコールバックが実行コンテキストにより実行される
    - リターンコードがRTC_OK以外の場合はエラー状態に遷移する
    - RT System Editor上での表示は緑
  - Error
    - エラー状態
    - onErrorコールバックが実行コンテキストにより実行される
    - reset_componentメソッドを呼び出すと非活性状態に遷移する
    - RT System Editor上での表示は赤
  - 終了状態

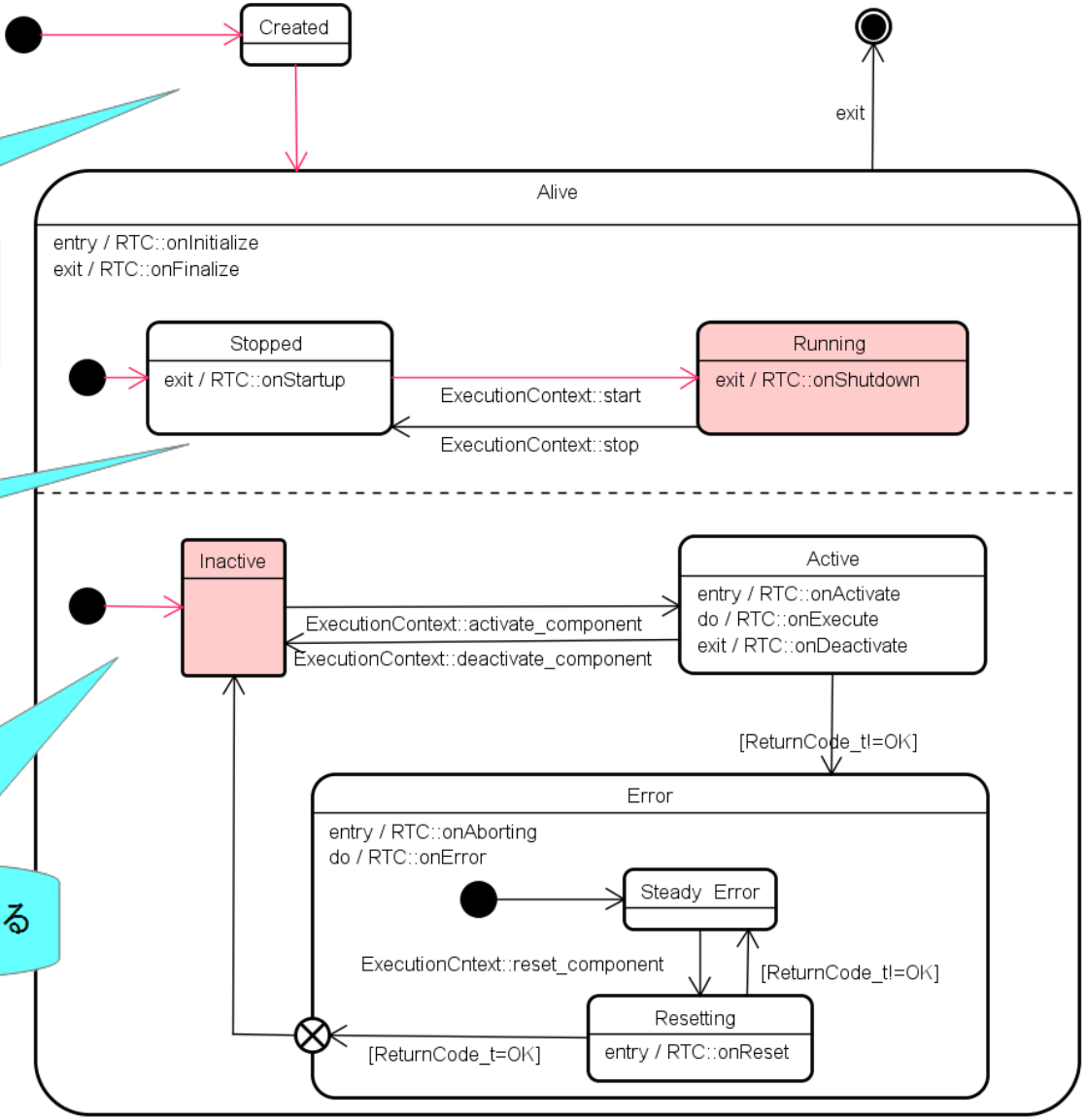


RTコンポーネントの状態遷移(生成直後)

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

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

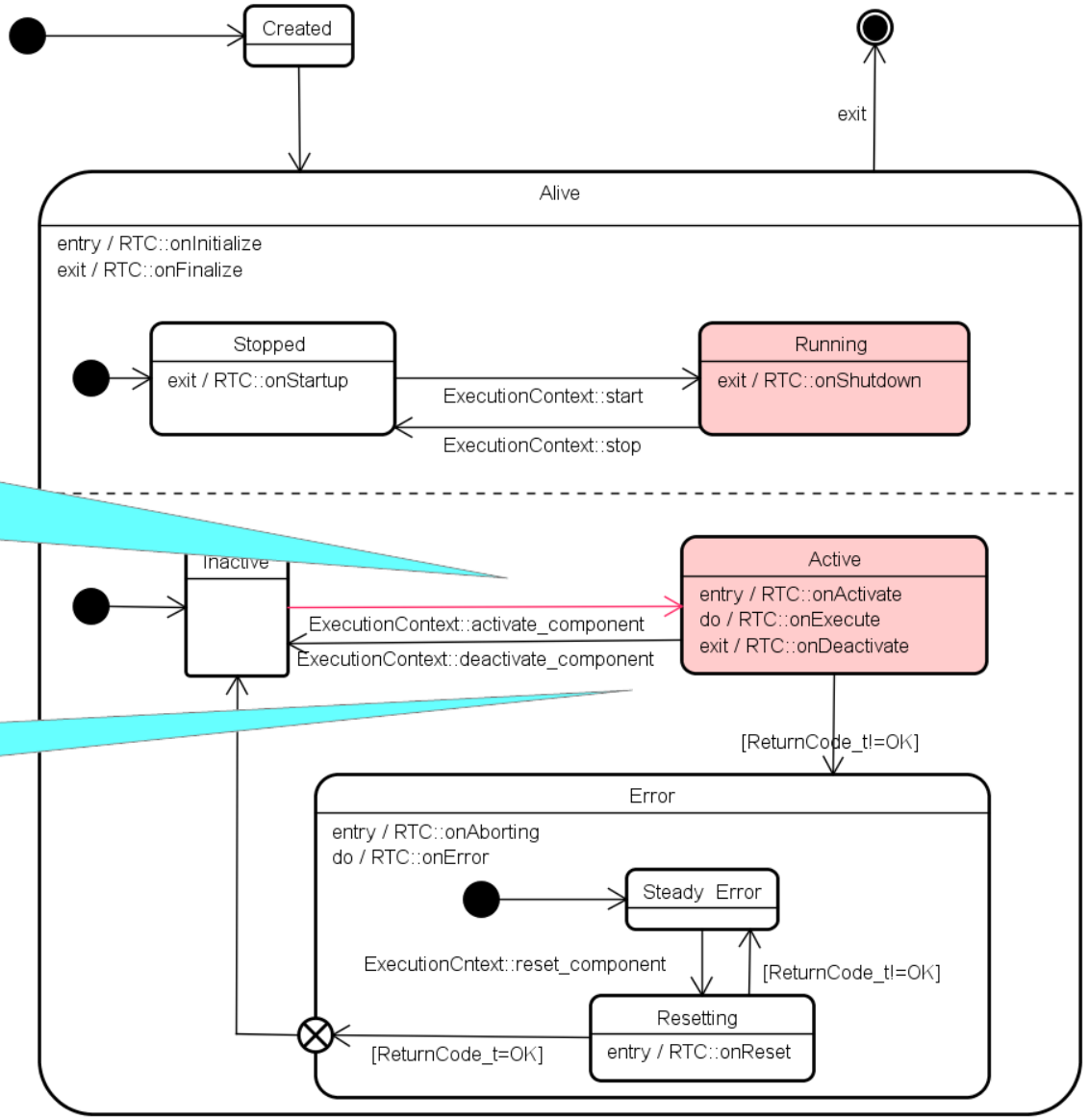
Created状態の次にInactive状態に遷移する



RTコンポーネントの状態遷移(アクティブ化)

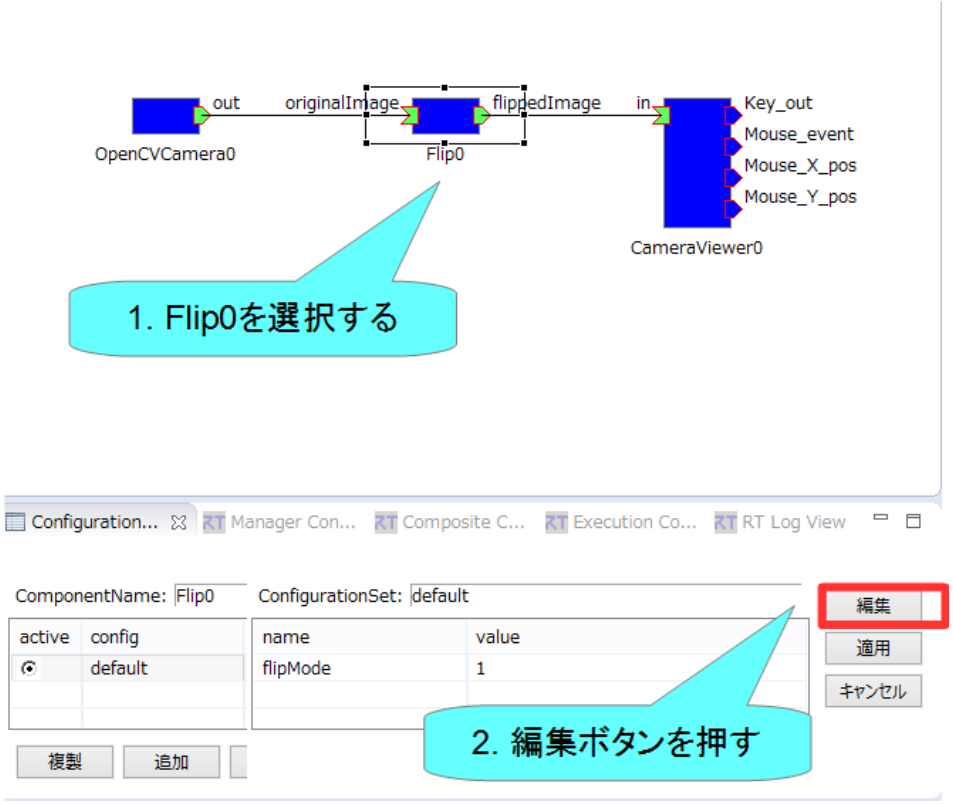
RTシステムエディタの操作によりRTコンポーネントのアクティブ化を行うとactivate_componentメソッドが呼び出される。
activate_componentメソッドによりコンポーネントがActive状態に遷移する。
この時onActivatedコールバックが実行される

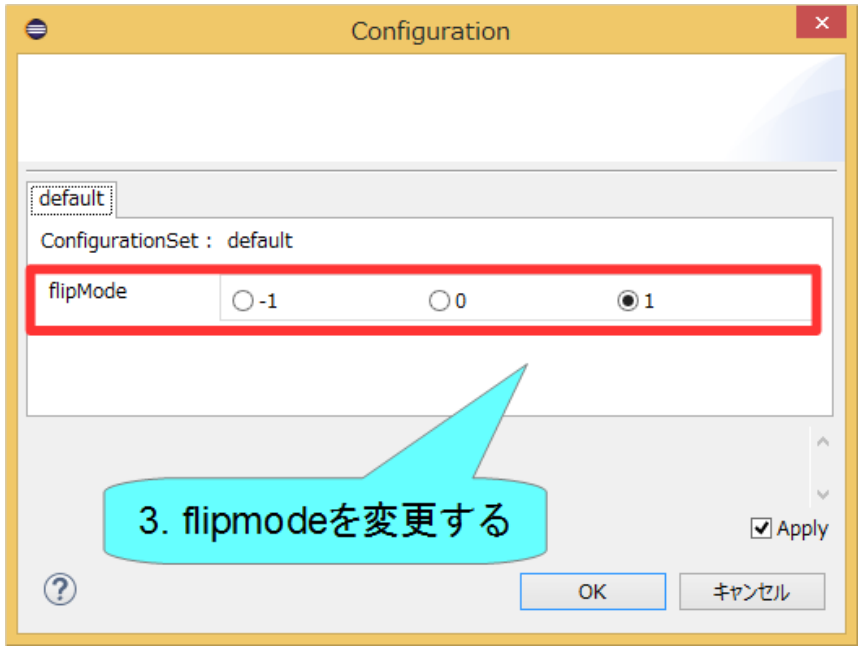
周期実行の実行コンテキストの場合、
onExecuteコールバックが周期的に呼び出される。



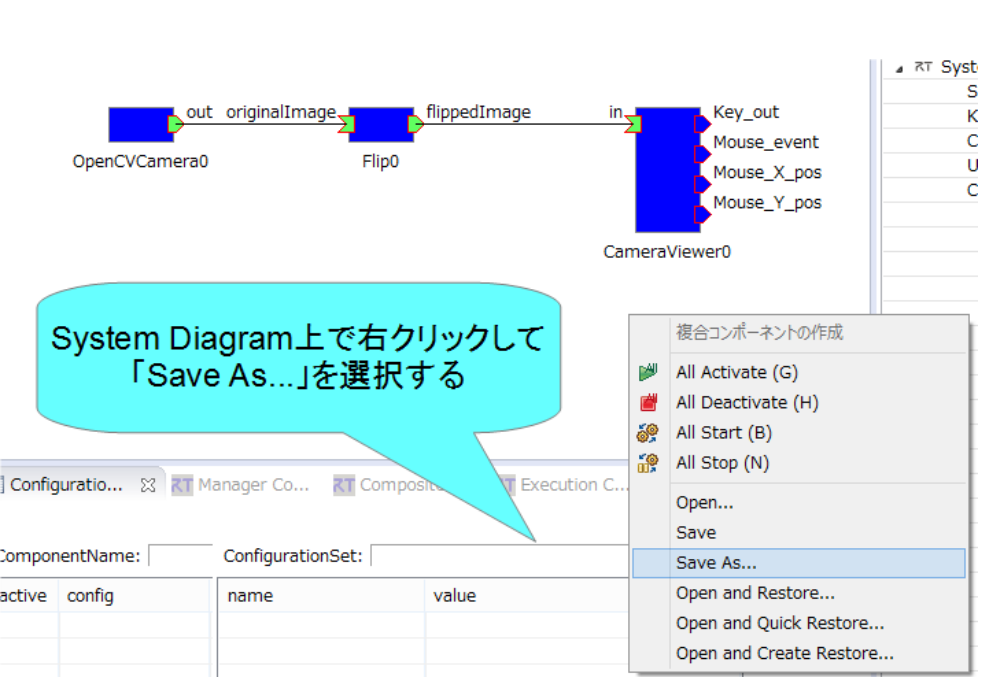
コンフィギュレーションパラメータの操作

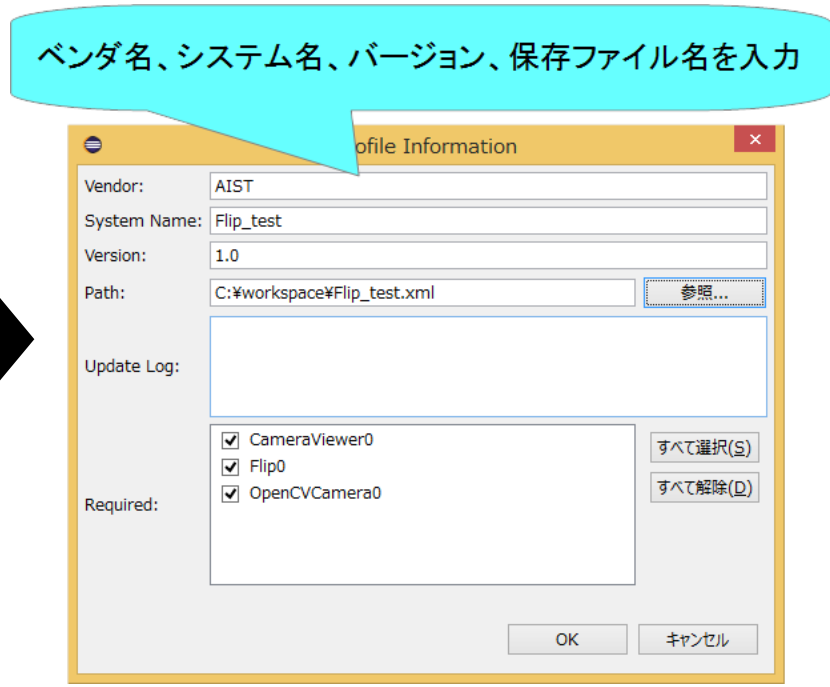
- コンフィギュレーションパラメータをRTシステムエディタから操作する
  - 反転する方向を設定



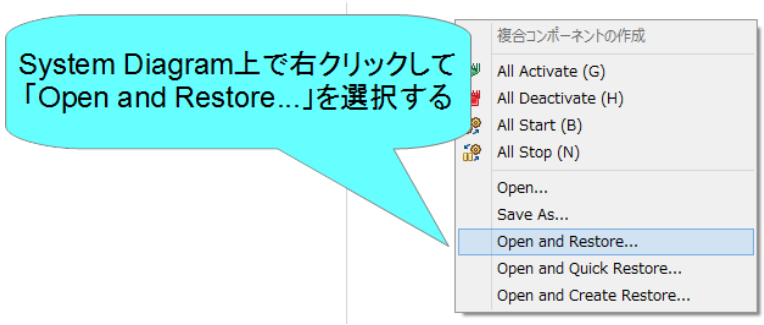


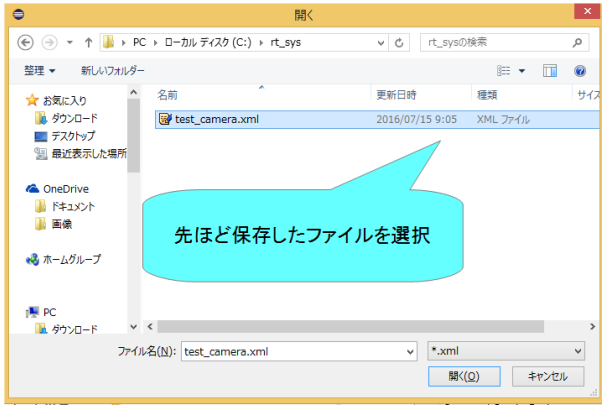
システムの保存

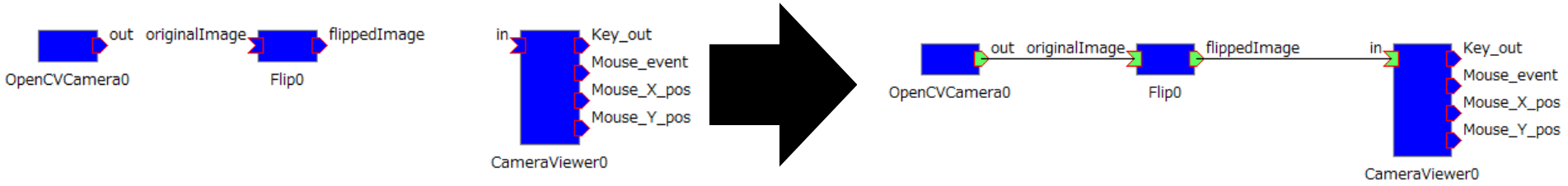

out
originalImage
flippedImage
in
Key_out
Mouse_event
Mouse_X_pos
Mouse_Y_pos
OpenCVCamera0
Flip0
CameraViewer0
System Diagram上で右クリックして
「Save As...」を選択する
複合コンポーネントの作成
All Activate (G)
All Deactivate (H)
All Start (B)
All Stop (N)
Open...
Save
Save As...
Open and Restore...
Open and Quick Restore...
Open and Create Restore...
ComponentName:
ConfigurationSet:
active
config
name
value


ベンダ名、システム名、バージョン、保存ファイル名を入力
Vendor:
AIST
System Name:
Flip_test
Version:
1.0
Path:
C:¥workspace¥Flip_test.xml
参照...
Update Log:
Required:
CameraViewer0
Flip0
OpenCVCamera0
すべて選択(S)
すべて解除(D)
OK
キャンセル

システムの復元





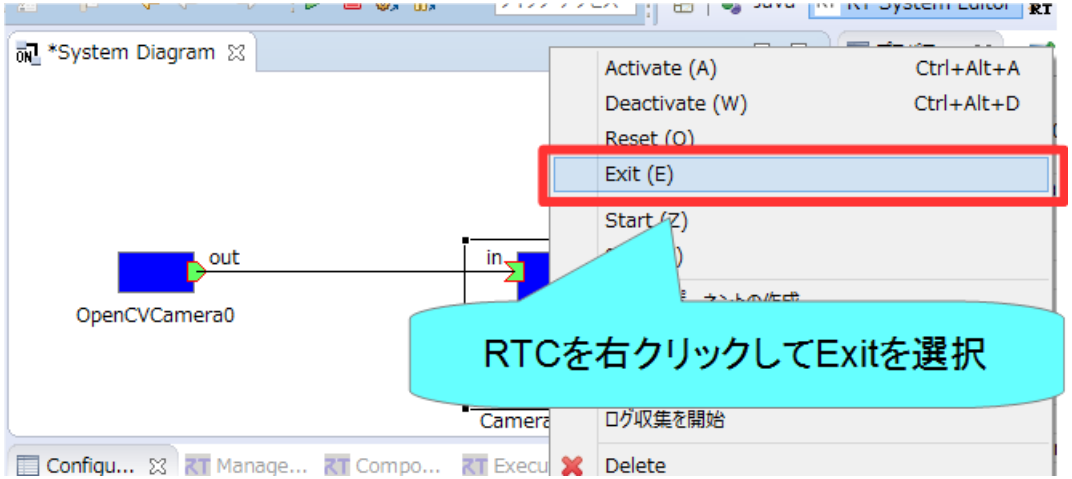


- 以下の内容を復元
  - ポート間の接続
  - コンフィギュレーション
  - 「Open and Create Restore」を選択した場合はマネージャからコンポーネント起動

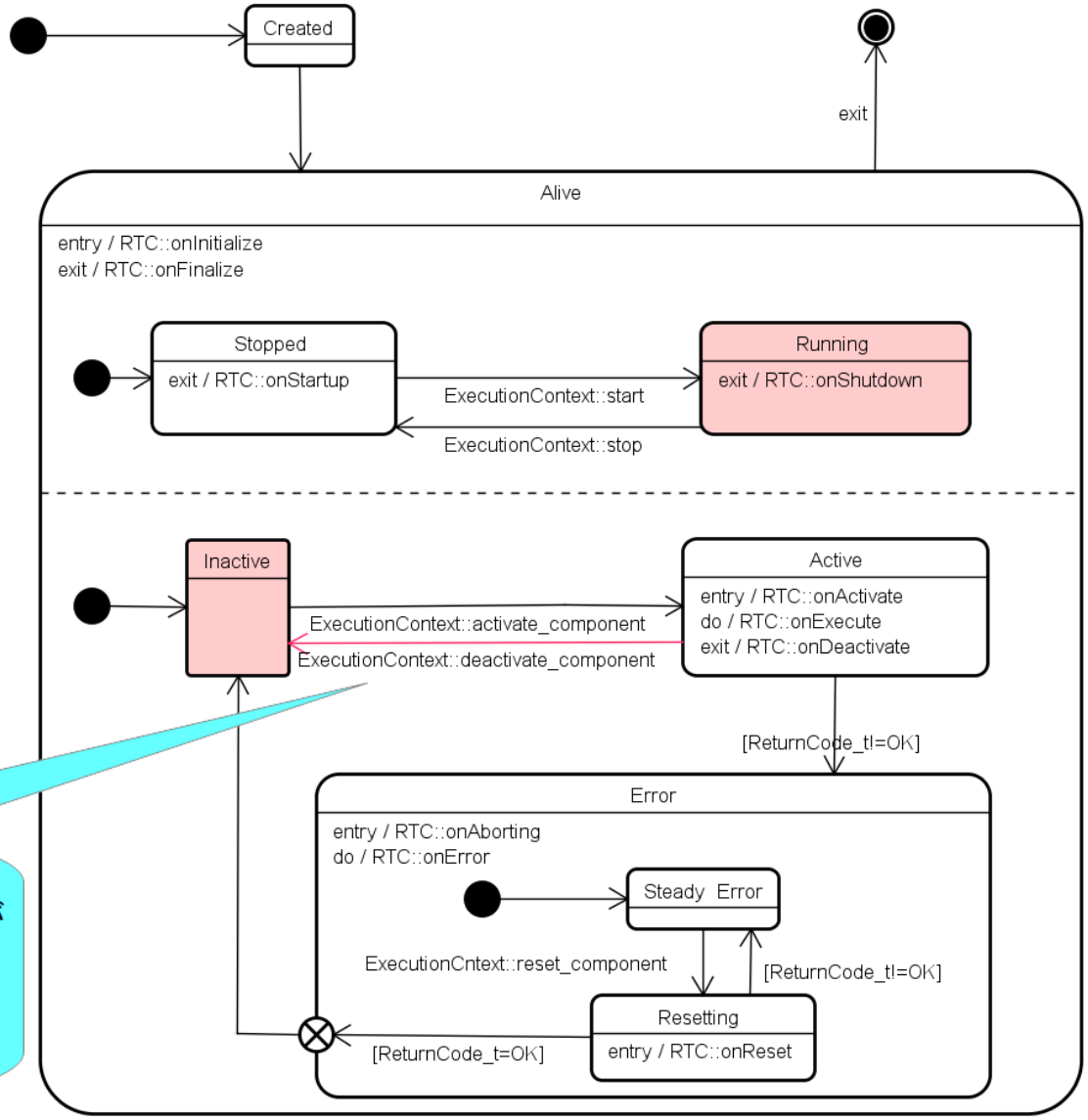
非アクティブ化、終了

- 非アクティブ化

- 終了



RTコンポーネントの状態遷移(非アクティブ化)

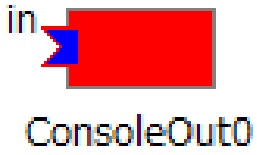


RTシステムエディタの操作によりRTコンポーネントの非アクティブ化を行うとdeactivate_componentメソッドが呼び出される。
deactivate_componentメソッドによりコンポーネントがInactive状態に遷移する。
この時onDeactivatedコールバックが実行される

RTC Builder
補足

リセット

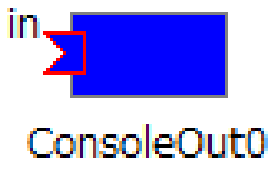
- RTCがエラー状態に遷移した場合にエディタ上には赤く表示される。



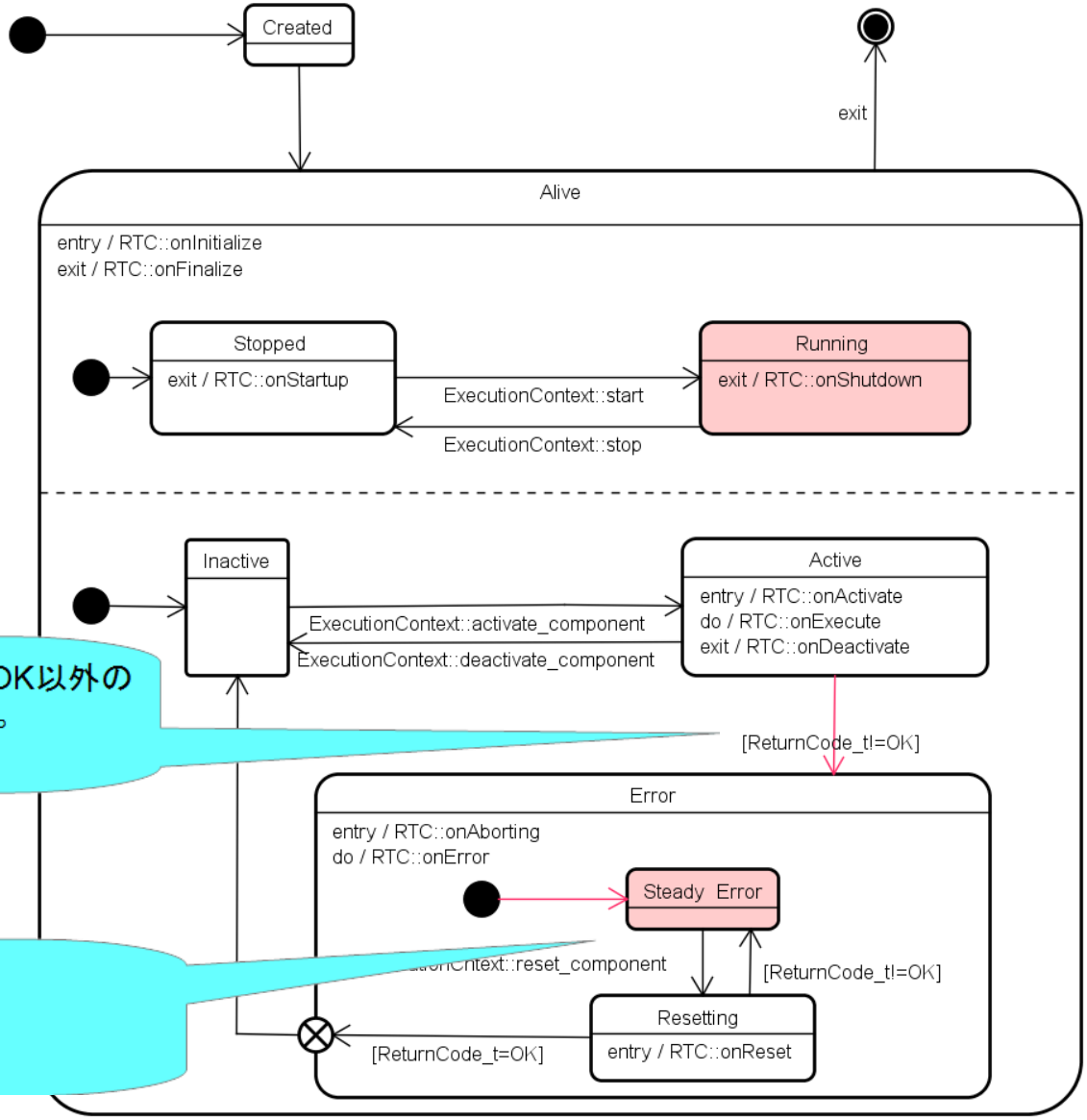

```
RTC::ReturnCode_t Test::onActivated(RTC::UniqueId ec_id)
{
        HANDLE hCom = INVALID_HANDLE_VALUE;
        hCom = CreateFile("COM5", GENERIC_READ | GENERIC_WRITE, 0, NULL, OPEN_EXISTING, 0, NULL);
        if (hCom == INVALID_HANDLE_VALUE)
        {
                return RTC::RTC_ERROR;
    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

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

- 以下の操作で非アクティブ状態に戻す



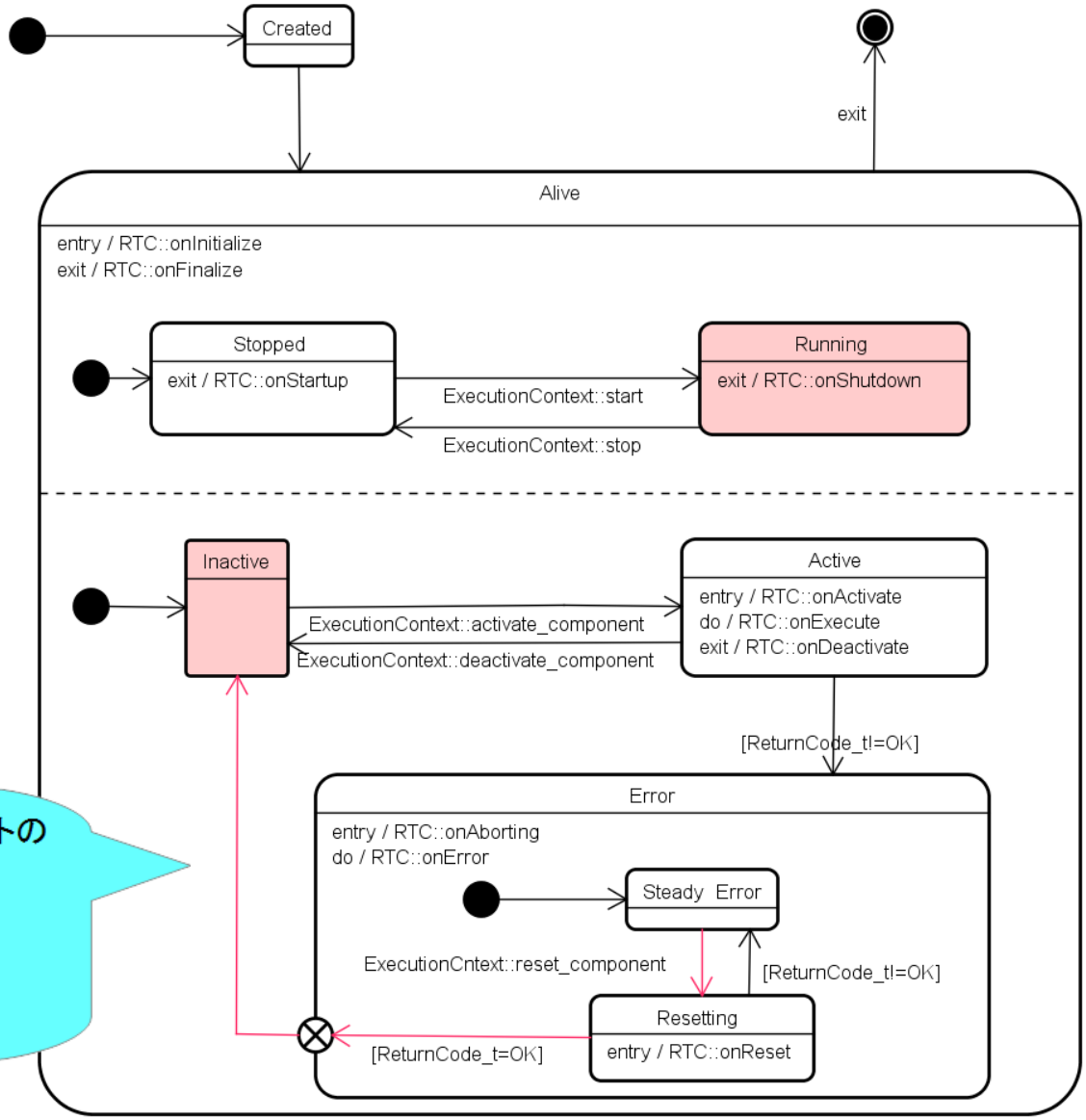
RTコンポーネントの状態遷移(エラー)



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

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

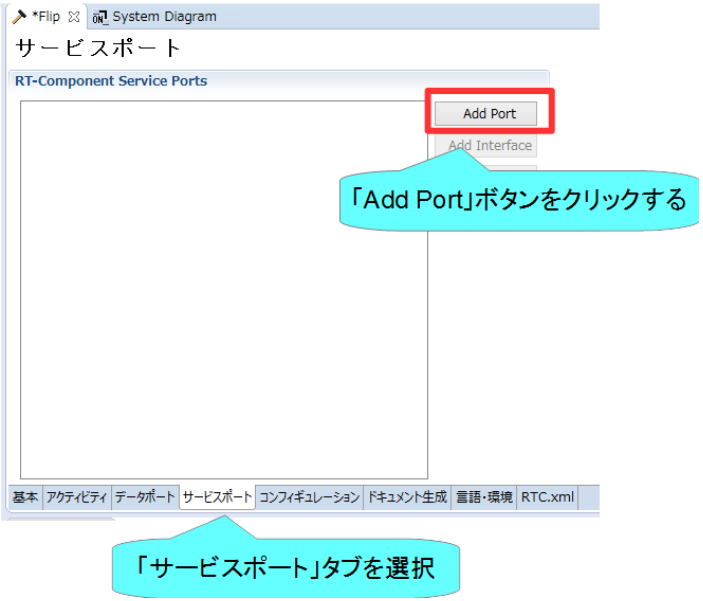
RTコンポーネントの状態遷移(リセット)

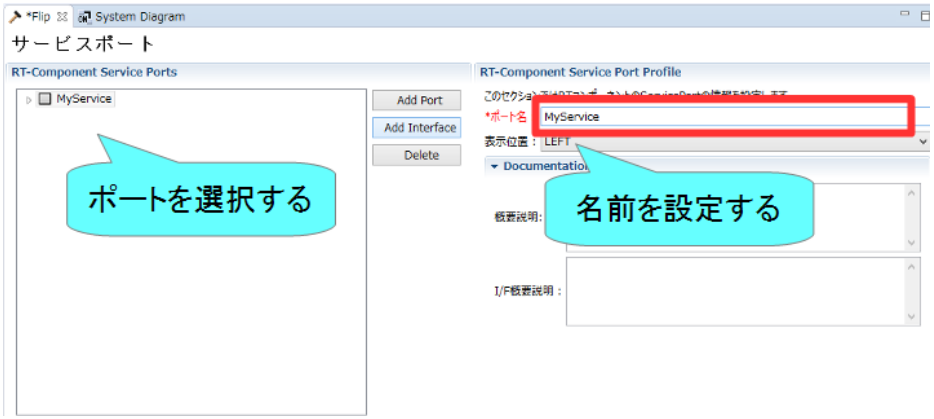


RTシステムエディタの操作によりRTコンポーネントのリセットを行うとreset_componentメソッドが呼び出される。
reset_componentメソッドによりコンポーネントがInactive状態に遷移する。
この時onResetコールバックが実行される

サービスポートの設定

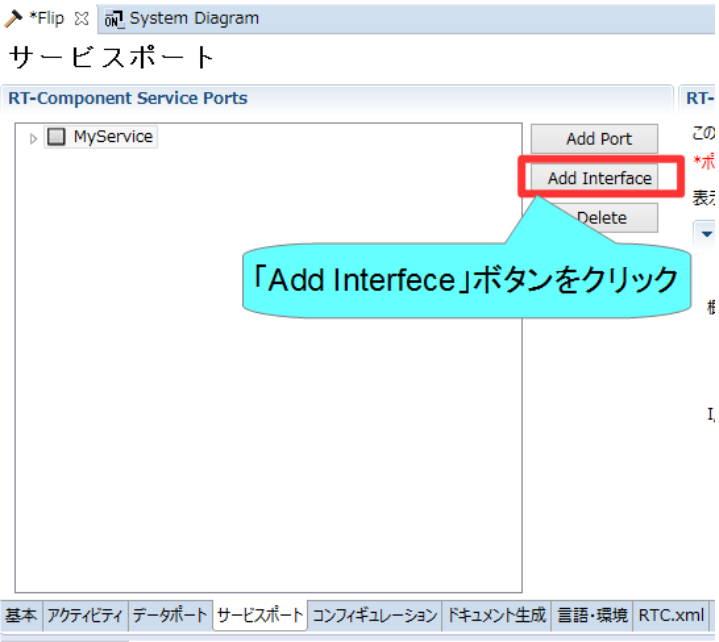
- サービスポートの追加、インターフェースの追加、設定を行う

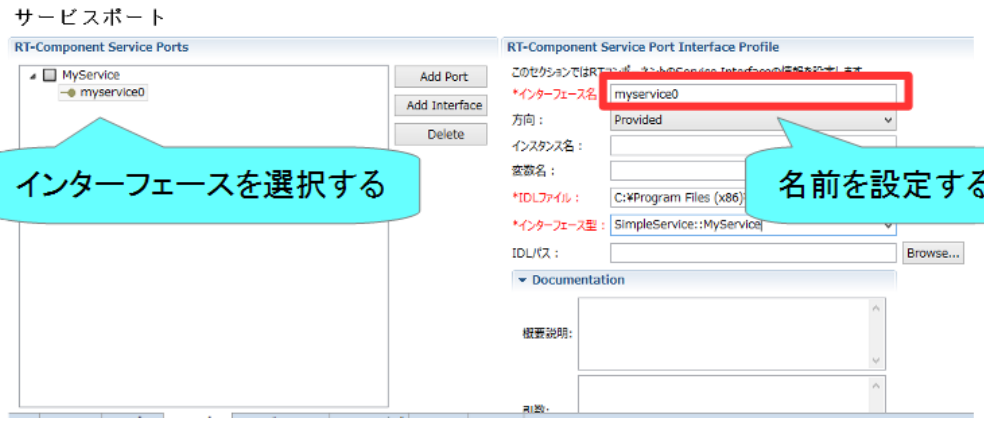




サービスポートの設定

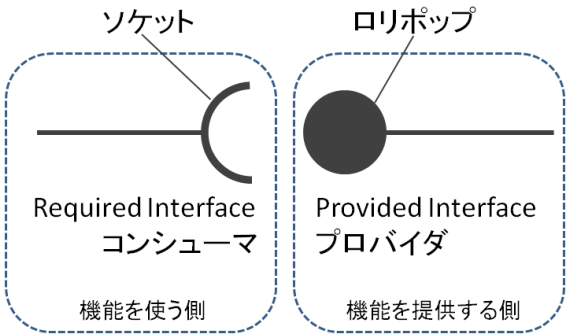
- インターフェースを追加する



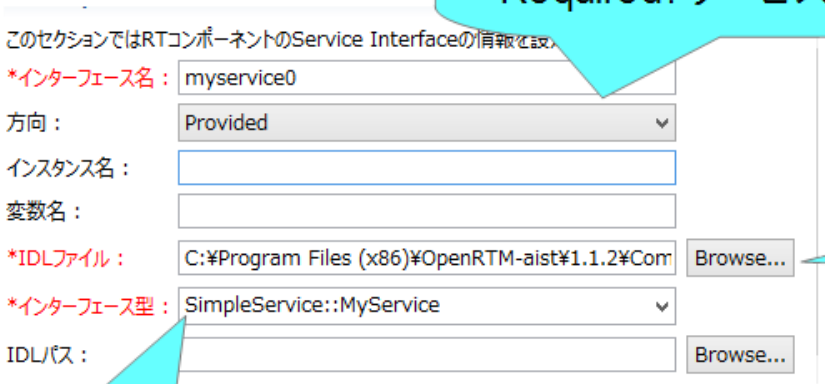


サービスポートの設定

- インターフェースの設定を行う



「Provided」・「Required」から選択
Provided：サービスを提供する側
Required：サービスを利用する側



「Browse...」をクリックしてIDLファイルを選択

インターフェース型を選択

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

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

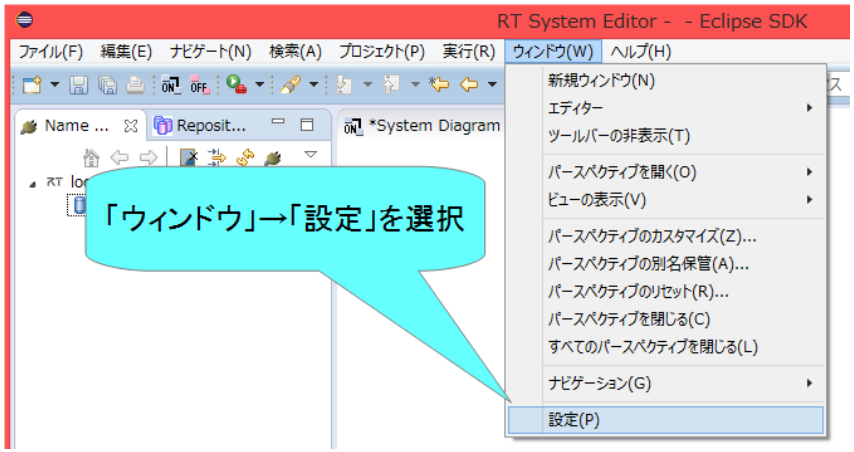
サービスポートの設定

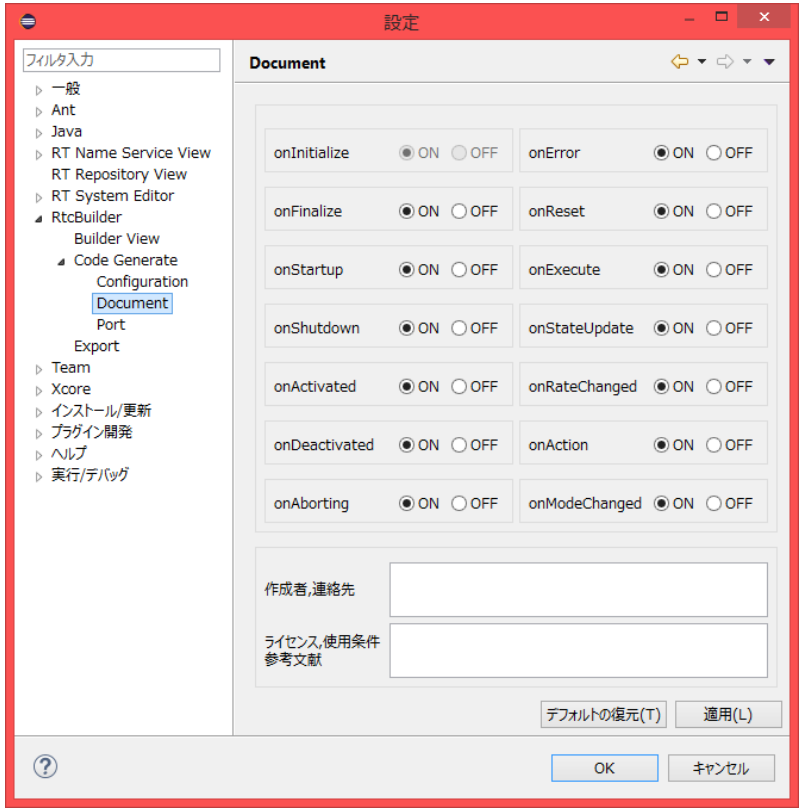
- IDLファイルについて
  - プログラミング言語に非依存のインターフェース定義言語

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

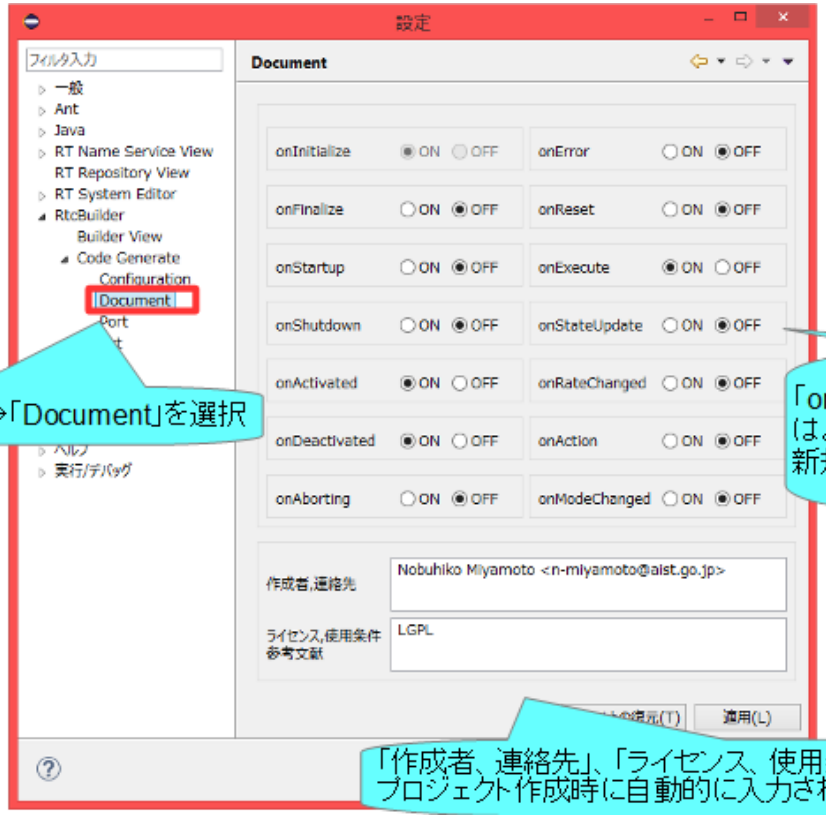
- コンシュマー側でプロバイダ側のecho、get_valueなどのオペレーションを呼び出す

RTC Builderに関する設定





RTC Builderに関する設定



「RtcBuilder」→「Code Generate」→「Document」を選択

「onActivated」、「onDeactivated」、「onExecute」はよく使うので「ON」にしておくとプロジェクトを新規作成したときに自動的にONになるので作業が減る。

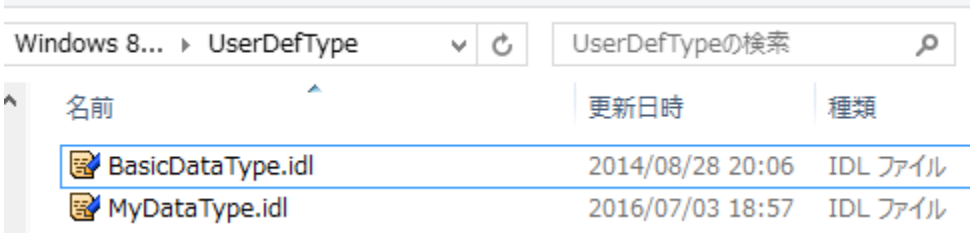
「作成者、連絡先」、「ライセンス、使用条件、参考文献」を入力しておくとプロジェクト作成時に自動的に入力されるので便利。

独自のデータ型の利用

- 独自のデータ型でデータポートの通信を行う手順
  - IDLファイルを作成する
    - MyDataType.idlを任意のフォルダ(ここではC:¥UserDefType)作成

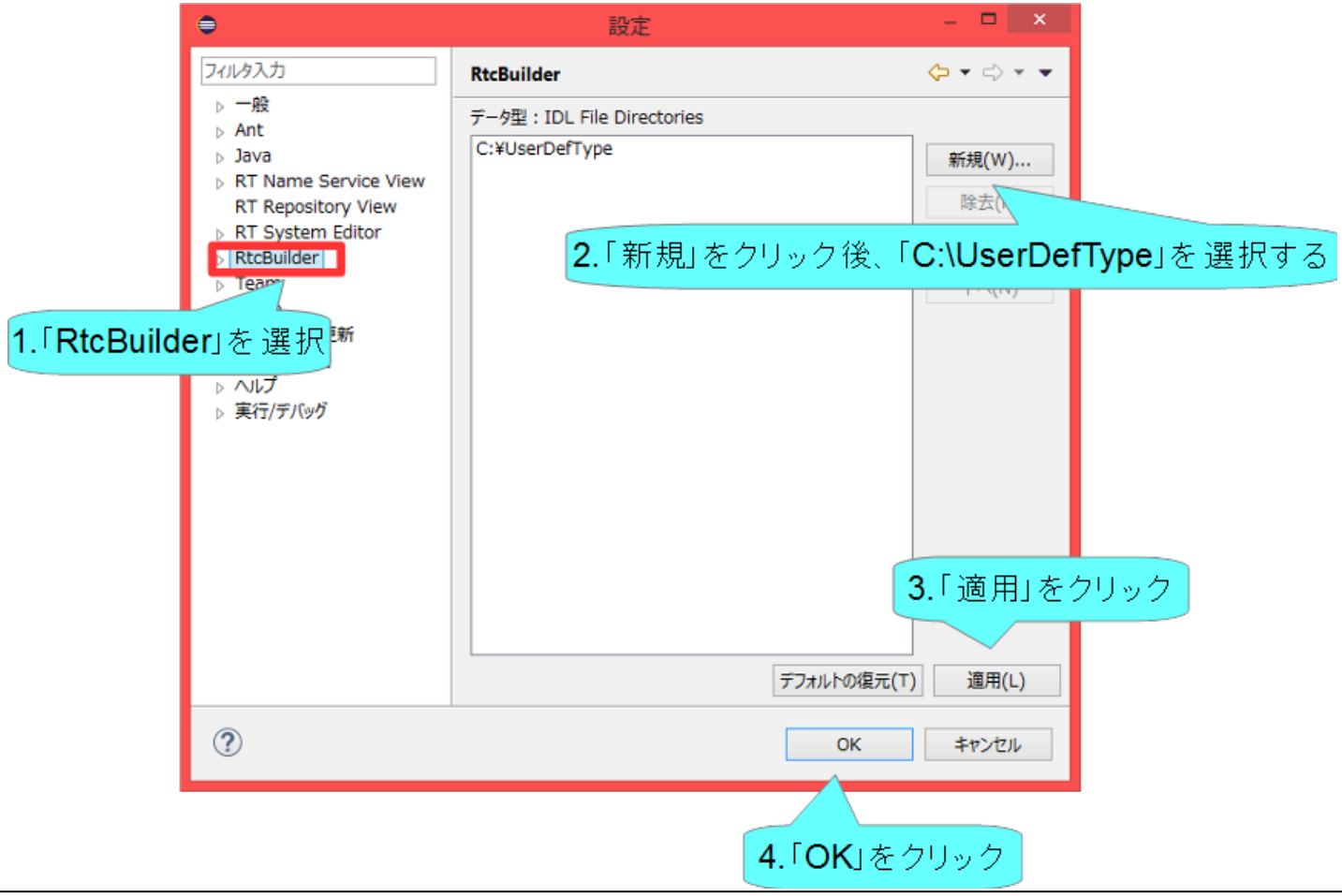
```
1  // @file MyDataType.idl
2  #include "BasicDataType.idl"
3
4  struct MyData
5  {
6    RTC::Time tm;
7    short shortVariable;
8    long longVariable;
9    sequence<double> data;
10 };[EOF]
```

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



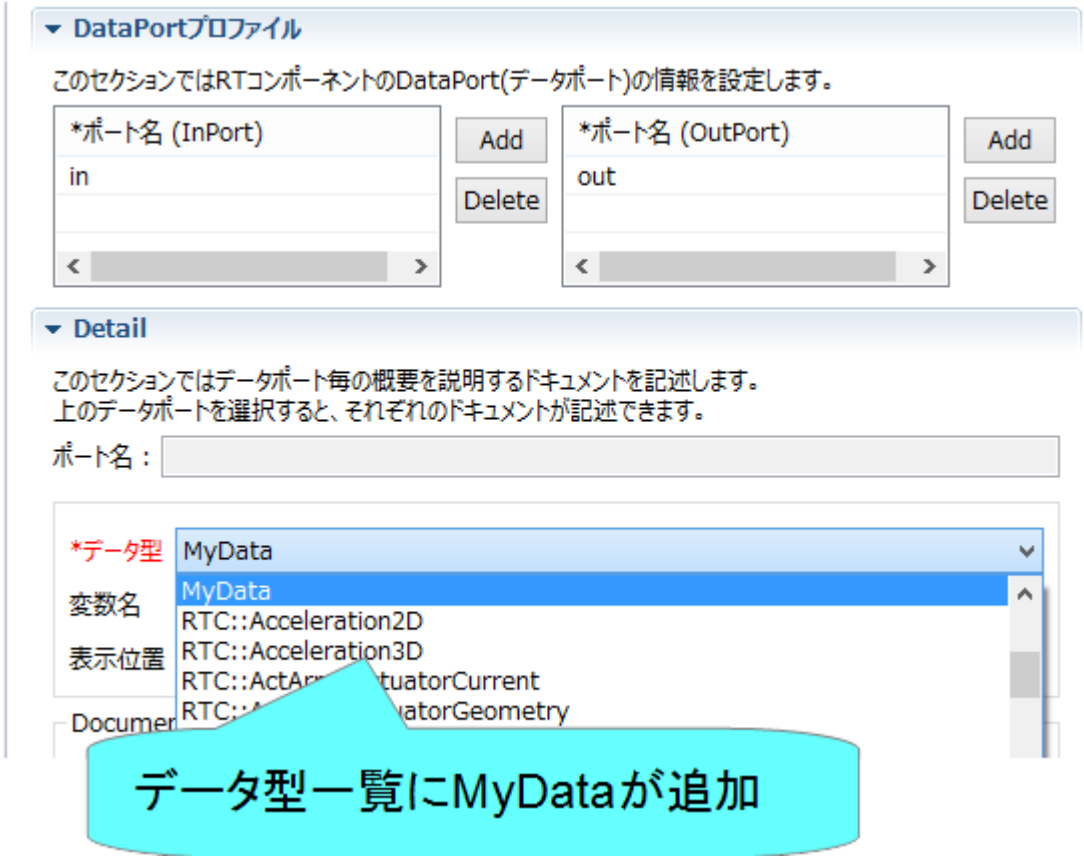
独自のデータ型の利用

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



独自のデータ型の利用

- 独自のデータ型でデータポートの通信を行う手順



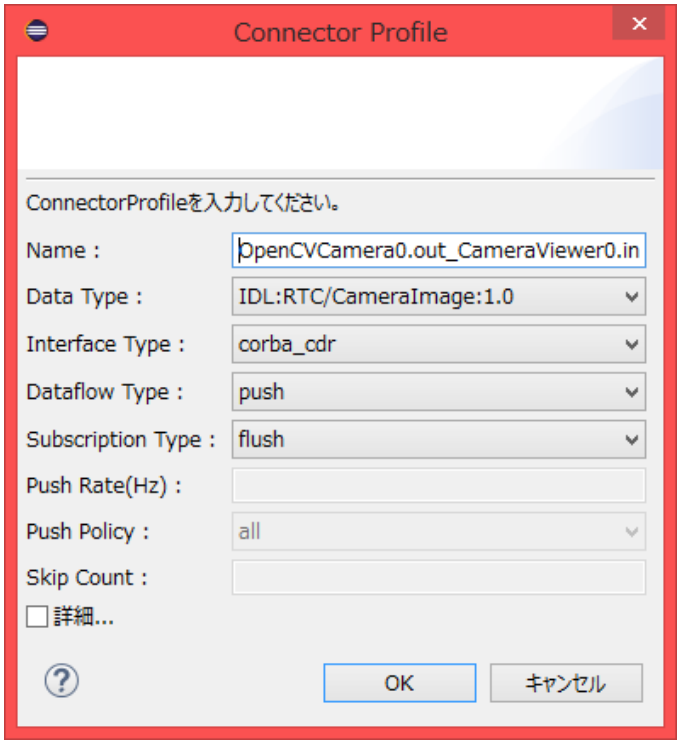
RT System Editor
補足

コネクタプロファイルの設定

項目	設定内容
Name	接続の名称
DataType	ポート間で送受信するデータの型. ex)TimedOctet,TimedShortなど
InterfaceType	データを送信方法. ex)corba_cdrなど
DataFlowType	データの送信手順. ex)push, pullなど
SubscriptionType	データ送信タイミング. 送信方法がPushの場合有効. New, Periodic, Flushから選択
Push Rate	データ送信周期(単位:Hz). SubscriptionTypeがPeriodicの場合のみ有効
Push Policy	データ送信ポリシー. SubscriptionTypeがNew, Periodicの場合のみ有効. all, fifo, skip, newから選択
Skip Count	送信データスキップ数. Push PolicyがSkipの場合のみ有効

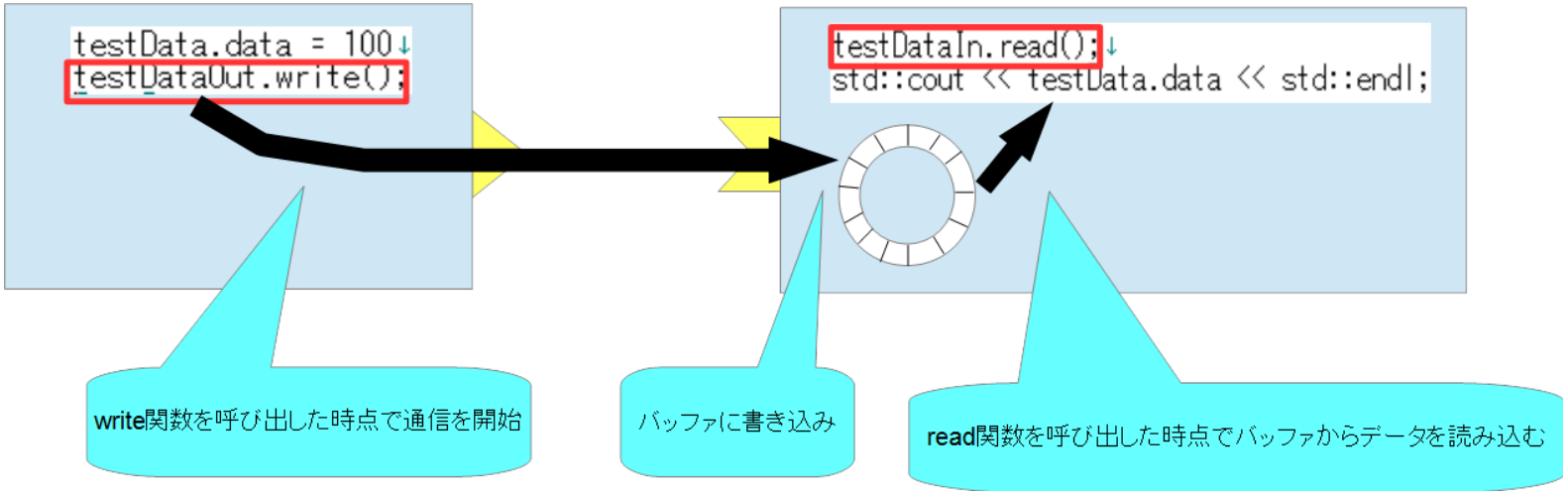
コネクタプロファイルの設定

- InterfaceTye
  - データの送信方法
  - 1.1.2ではcorba_cdr(CORBAによる通信)のみ選択可能
  - 1.2.0では以下の通信方法も選択可能になる予定
    - direct(同一プロセスで起動したRTC間でデータを直接変数に渡す)
    - shared_memory(共有メモリによる通信)
- DataFlowType
  - データの送信手順
    - Push
      - OutPortがInPortにデータを送る
    - Pull
      - InPortがOutPortに問い合わせてデータを受け取る
- SubscriptionType
  - データ送信タイミング(DataFlowTypeがPush型のみ有効)
    - flush(同期)
      - バッファを介さず即座に同期的に送信
    - new(非同期)
      - バッファ内に新規データが格納されたタイミングで送信
    - periodic(非同期)
      - 一定周期で定期的にデータを送信
- Push Policy(SubscriptionTypeがnew、periodicのみ有効)
  - データ送信ポリシー
    - all
      - バッファ内のデータを一括送信
    - fifo
      - バッファ内のデータをFIFOで1個ずつ送信
    - skip
      - バッファ内のデータを間引いて送信
    - new
      - バッファ内のデータの最新値を送信(古い値は捨てられる)

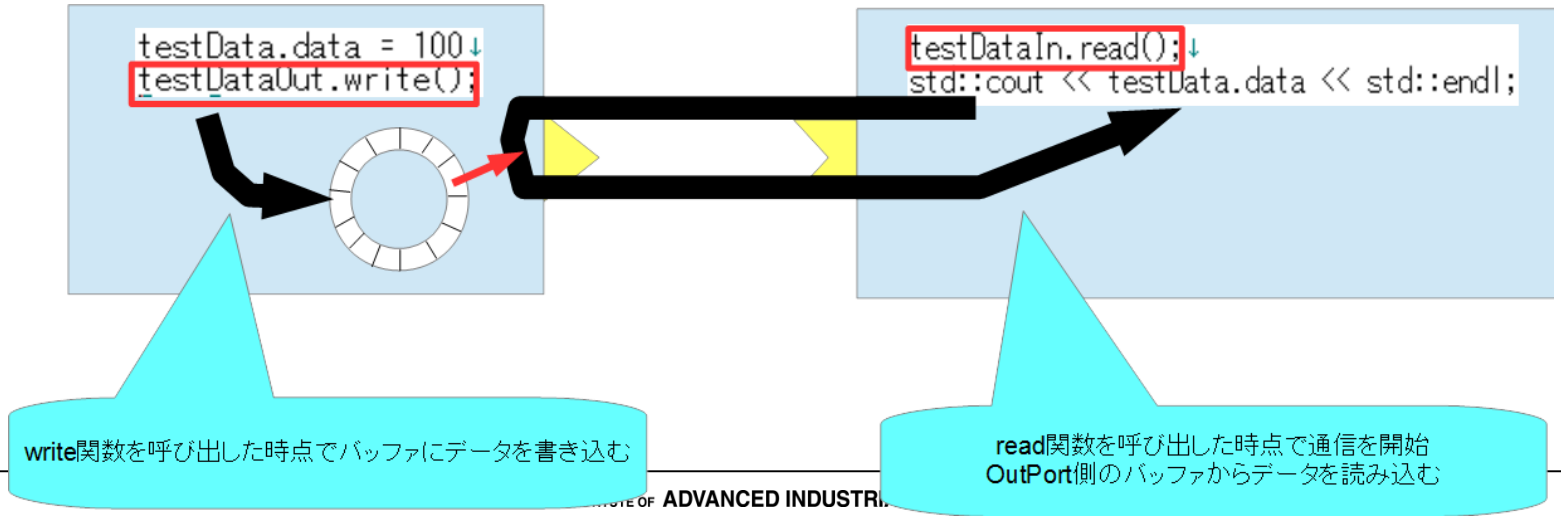


コネクタプロファイルの設定

- DataFlowType
  - Push

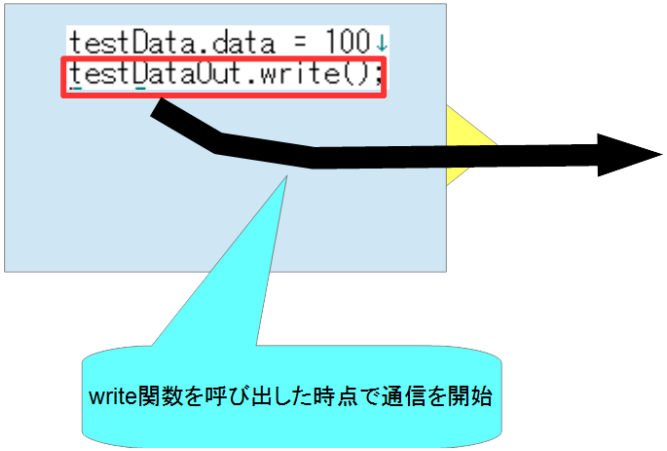


  - Pull

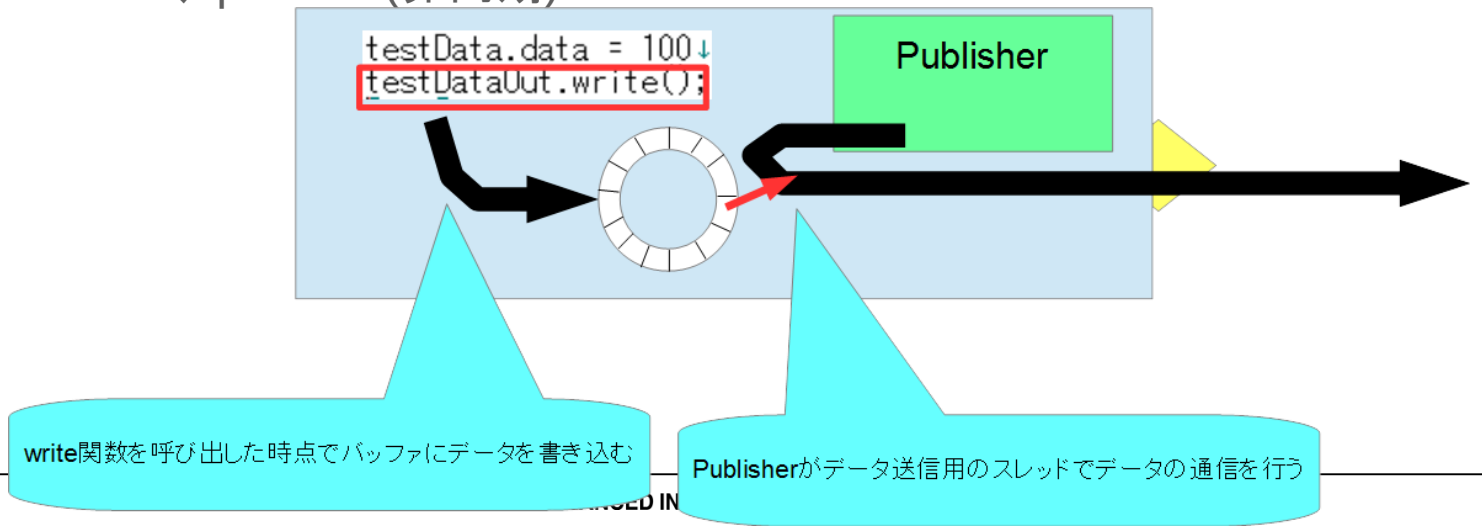


コネクタプロファイルの設定

- SubscriptionType
  - flush(同期)

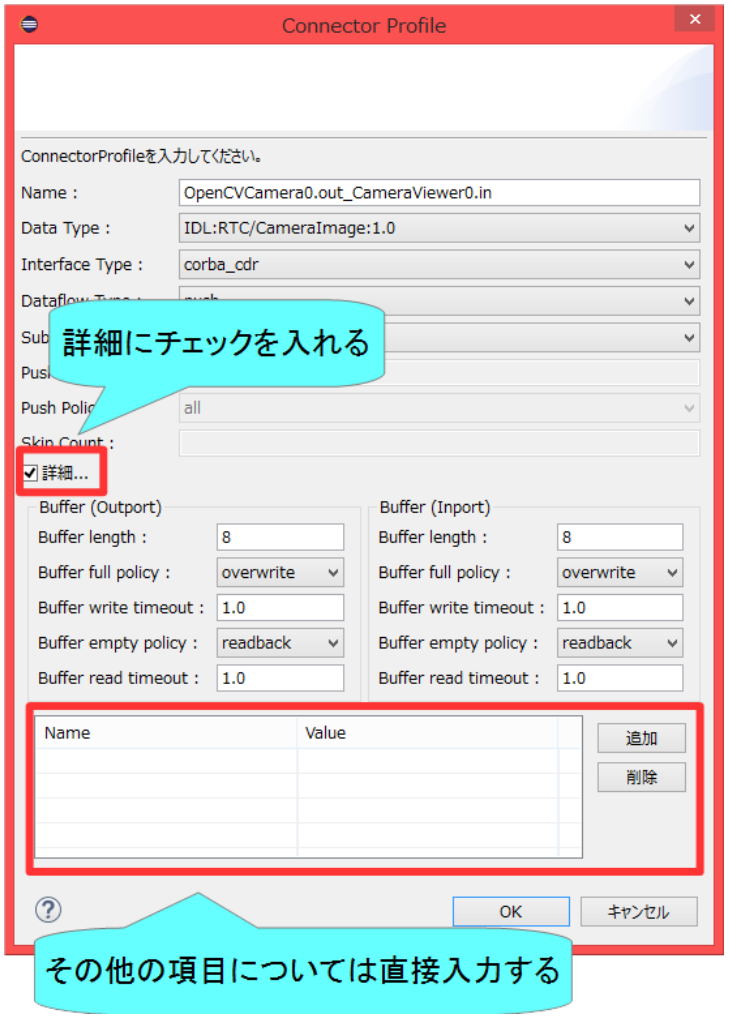


  - new、periodic(非同期)



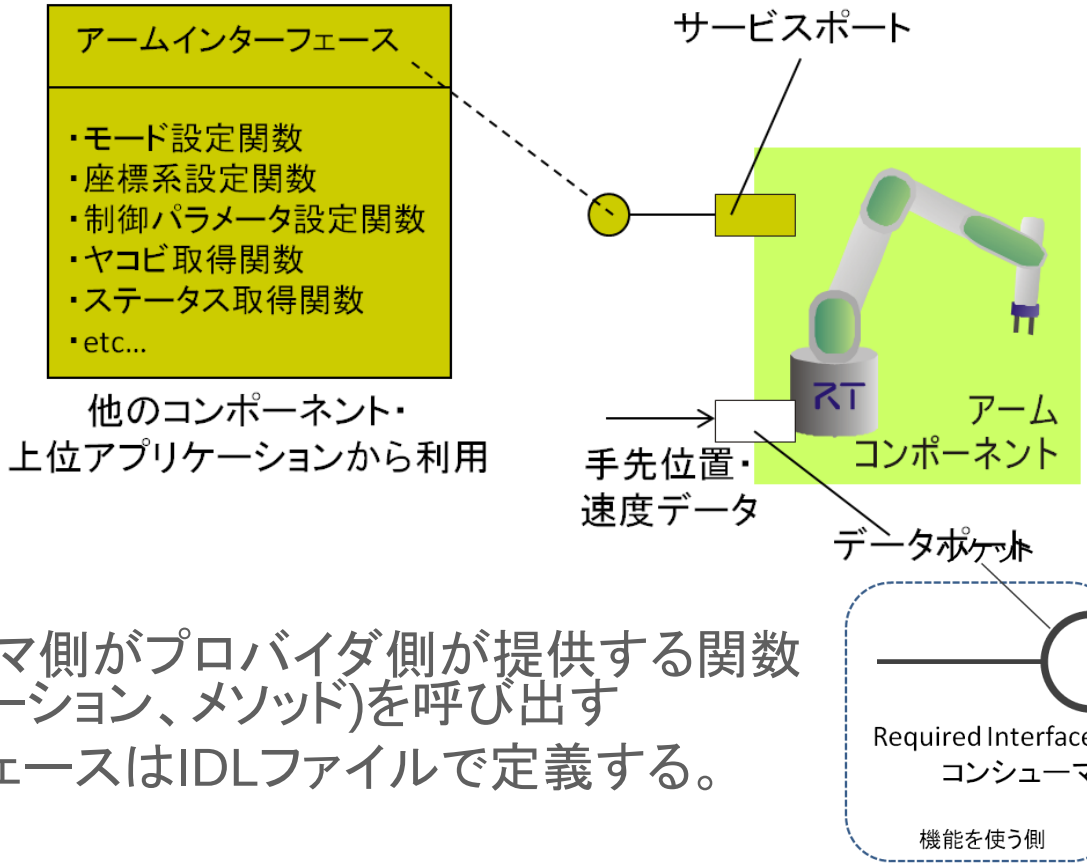
コネクタプロファイルの設定

項目	設定内容
Buffer length	バッファの大きさ
Buffer full policy	データ書き込み時に，バッファフルだった場合の処理．overwrite，do_nothing，blockから選択
Buffer write timeout	データ書き込み時に，タイムアウトイベントを発生させるまでの時間(単位:秒)
Buffer empty policy	データ読み出し時に，バッファが空だった場合の処理．readback，do_nothing，blockから選択
Buffer read timeout	データ読み出し時に，タイムアウトイベントを発生させるまでの時間(単位:秒)



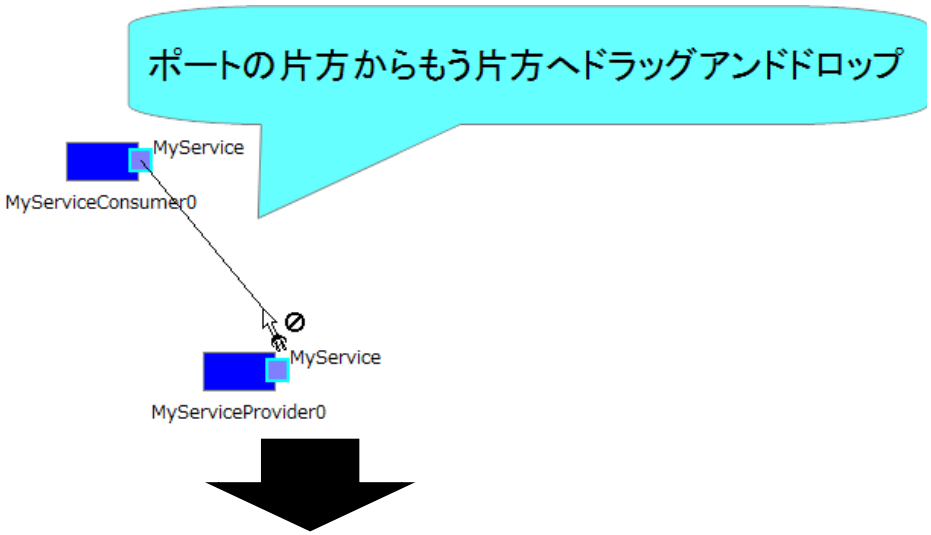
サービスポートについて

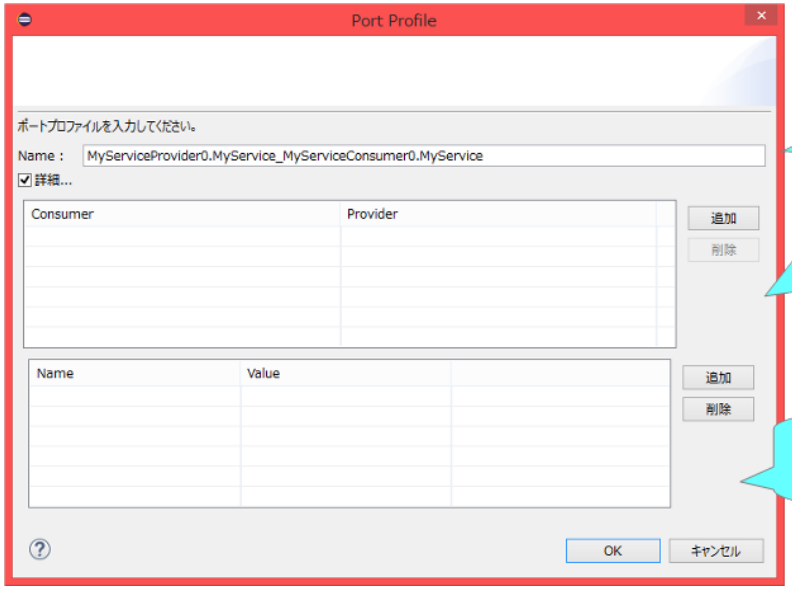
- コマンドレベルのやり取りを行うための仕組み
  - 任意のタイミングで操作を行いたい時などに使用
    - 例えばロボットアームのサーボを停止させる、ハンドを閉じる等



- コンシューマ側がプロバイダ側が提供する関数群(オペレーション、メソッド)を呼び出す
- インターフェースはIDLファイルで定義する。

サービスポートの接続





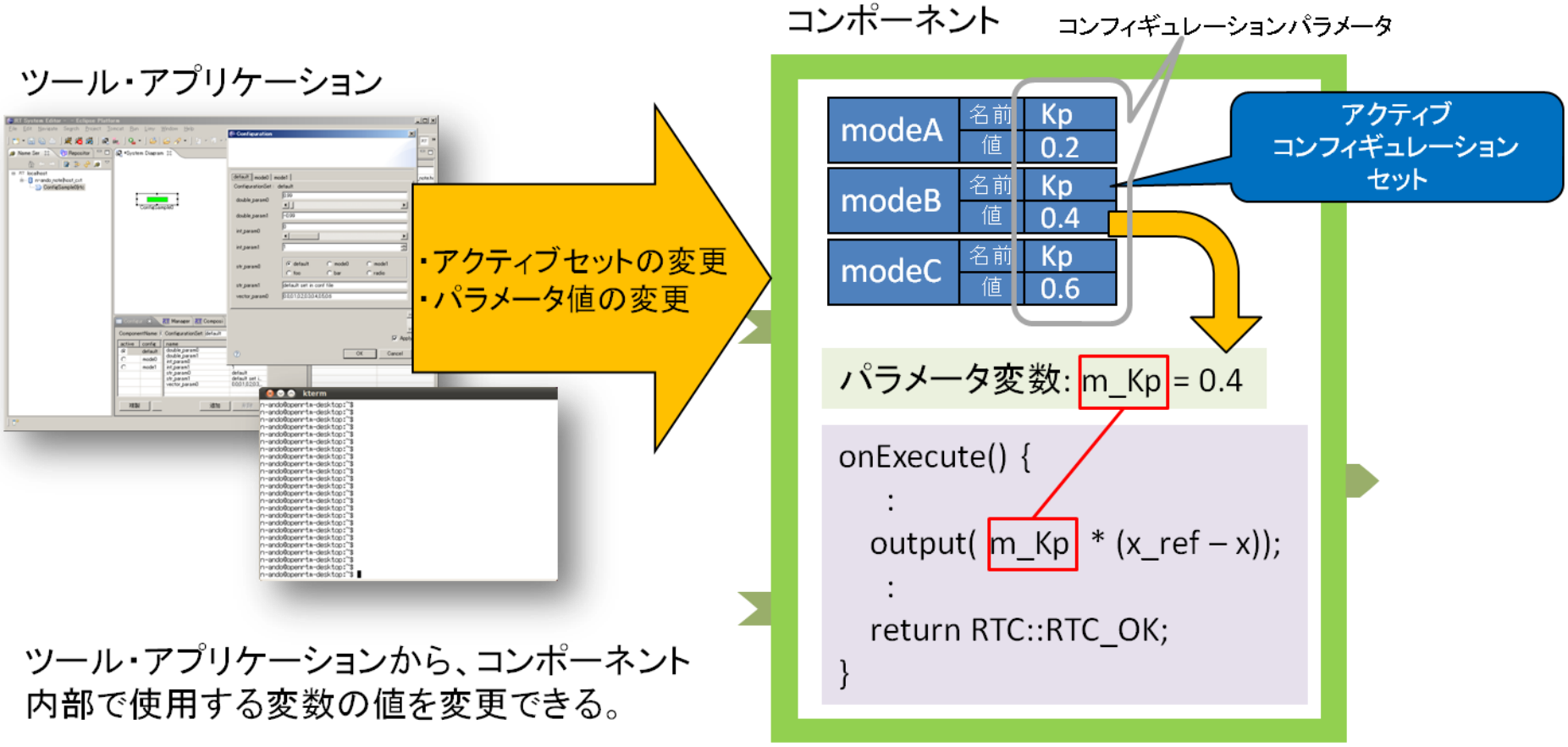
名前の設定

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

その他の設定を直接入力

コンフィギュレーションパラメータについて

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

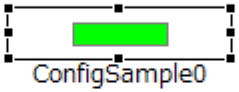


ツール・アプリケーションから、コンポーネント内部で使用する変数の値を変更できる。

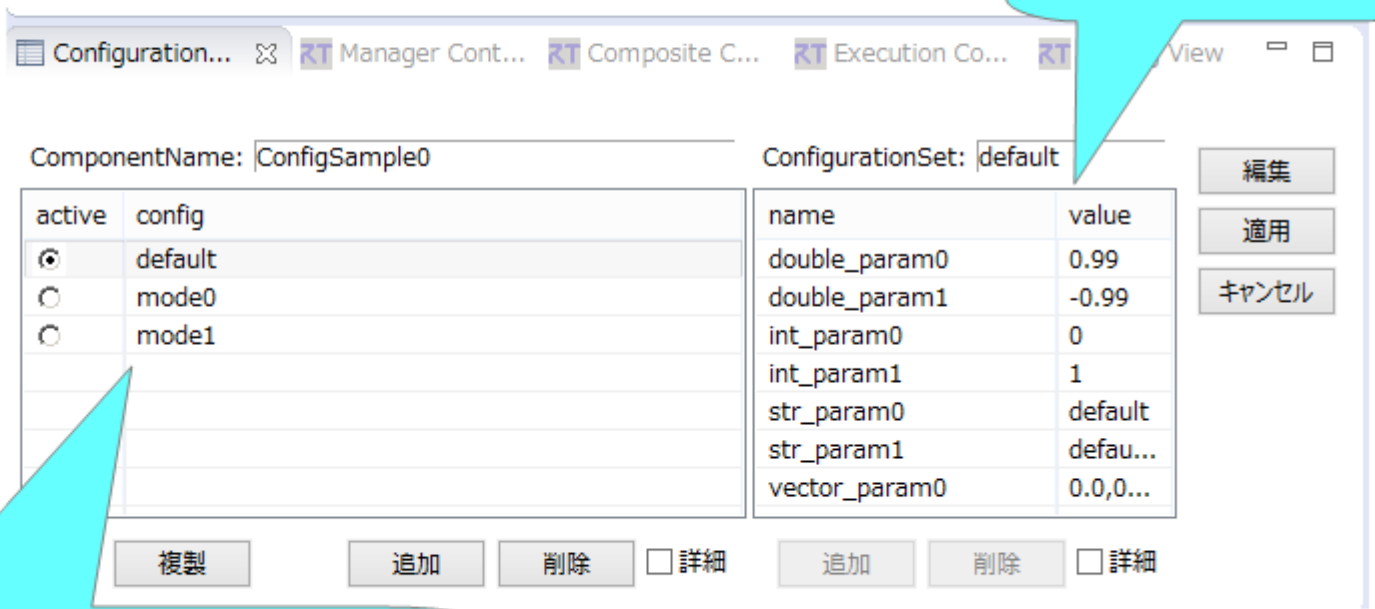
コンフィギュレーションパラメータの設定

対象のRTCをクリックすると表示

「Configuration View」タブを選択



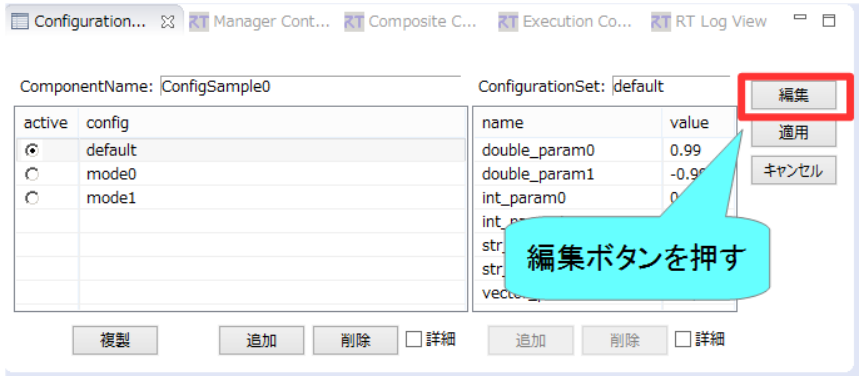
パラメーター一覧

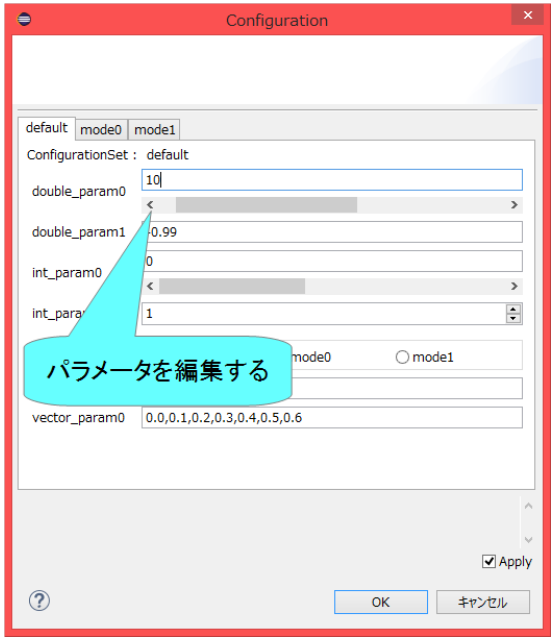


コンフィギュレーションセット一覧

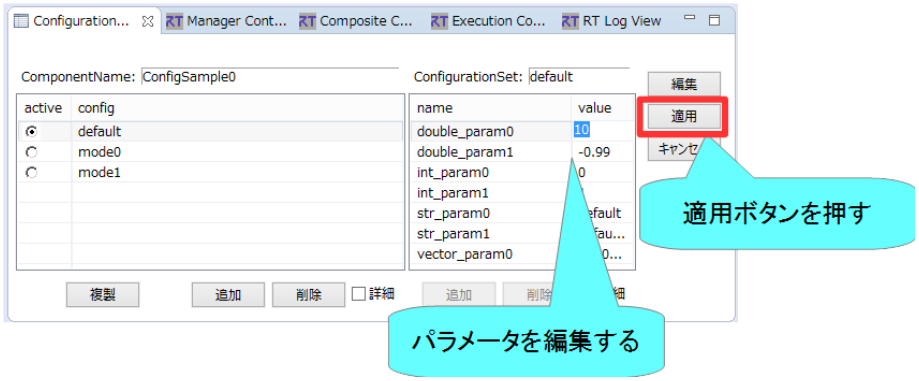
コンフィギュレーションパラメータの設定

- 方法1



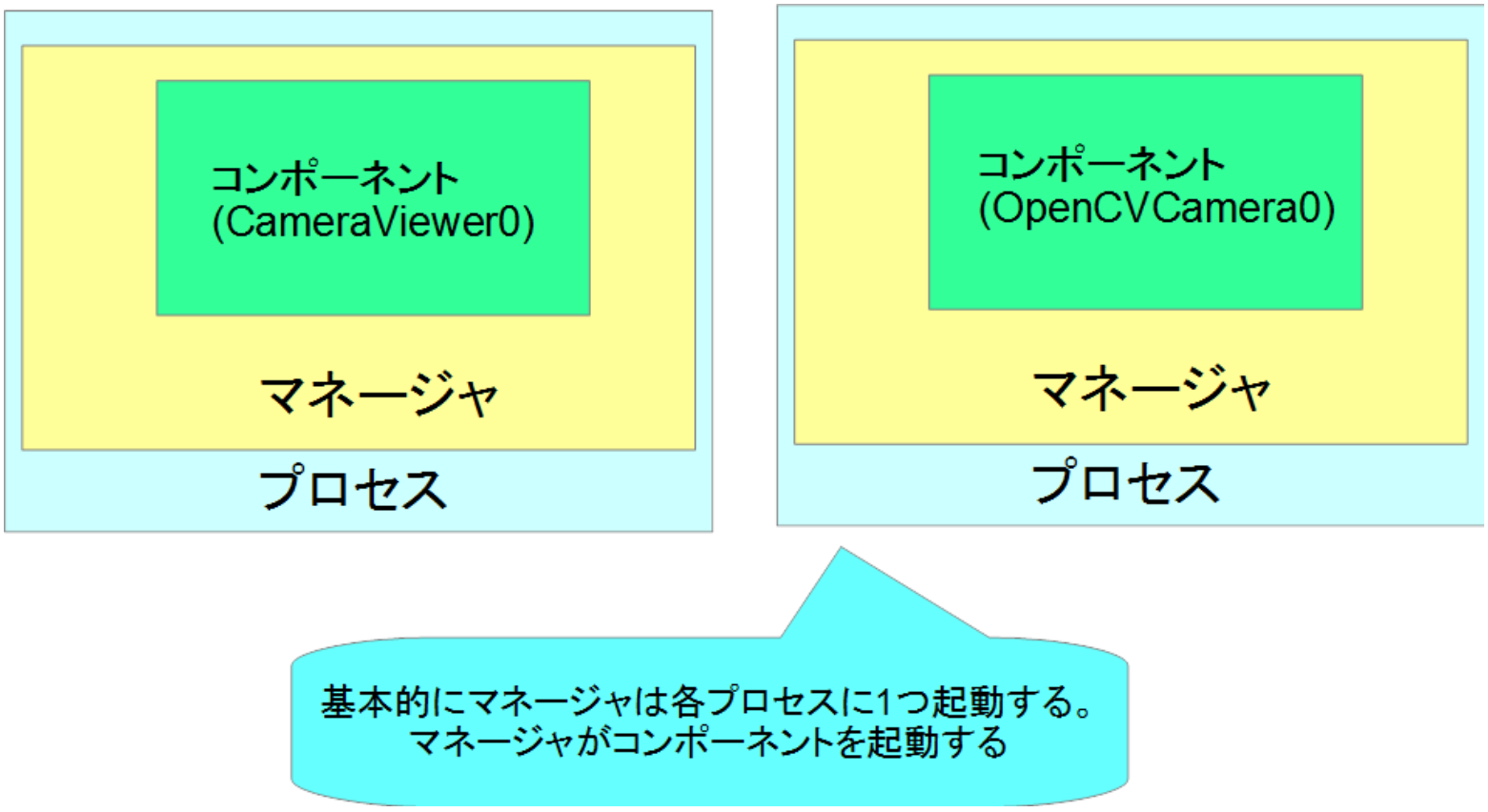


- 方法2

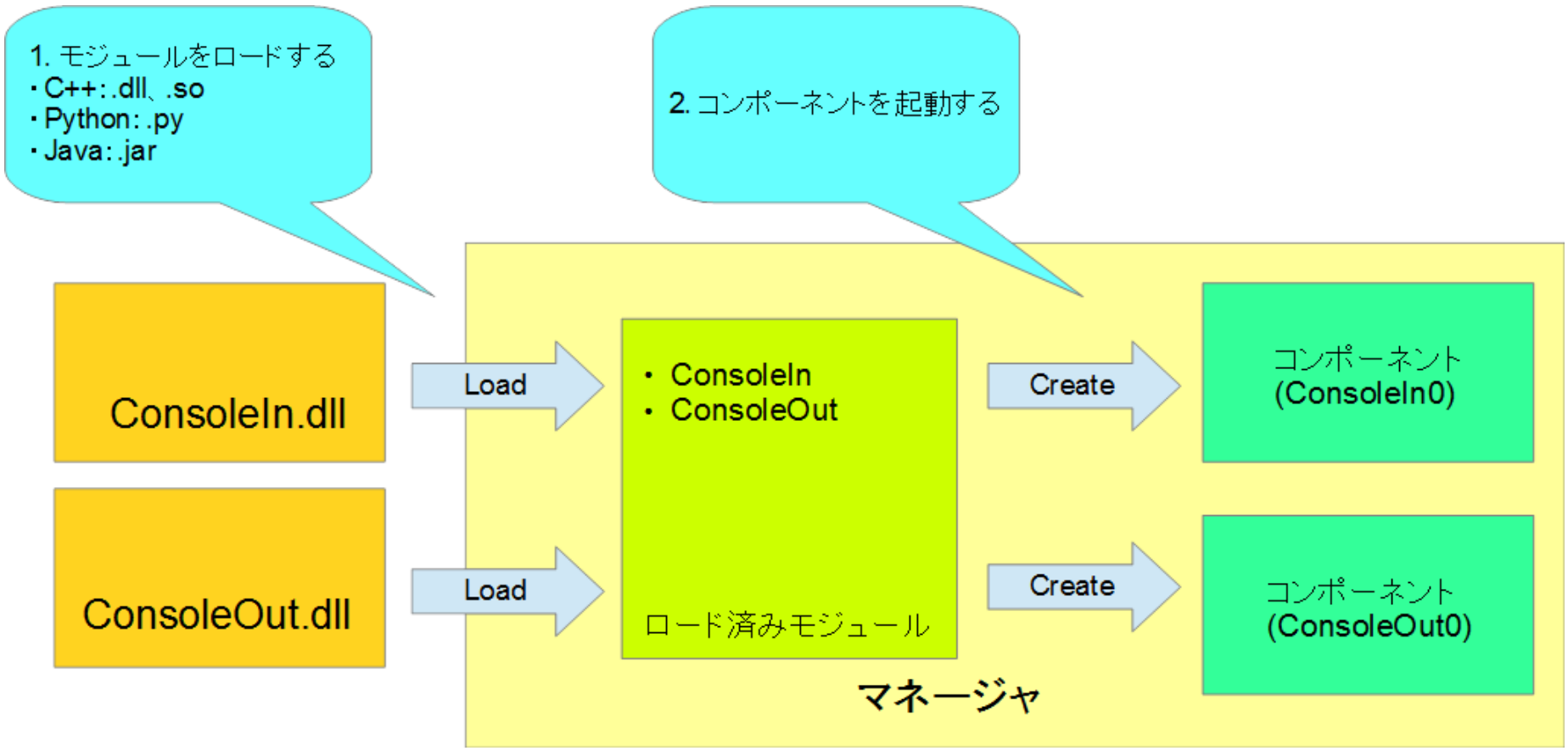


マネージャの操作

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

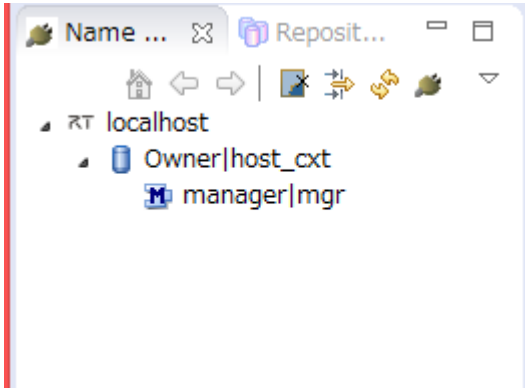


マネージャの操作


1. モジュールをロードする
・C++: .dll、.so
・Python: .py
・Java: .jar
2. コンポーネントを起動する
ConsoleIn.dll
ConsoleOut.dll
Load
Load
・ConsoleIn
・ConsoleOut
ロード済みモジュール
Create
Create
コンポーネント
(ConsoleIn0)
コンポーネント
(ConsoleOut0)
マネージャ

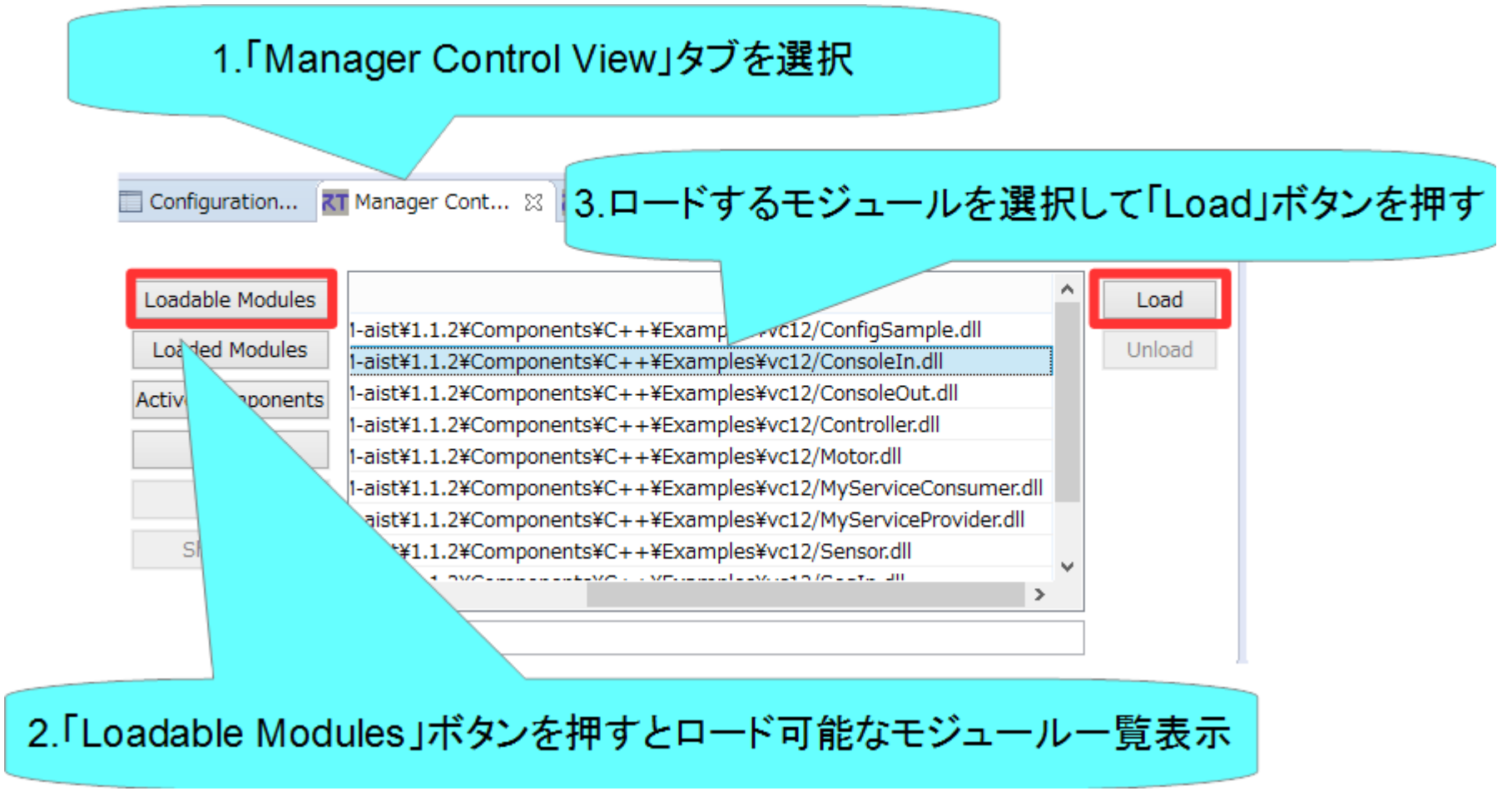
マネージャの操作

- マスターマネージャの起動、RT System Editorからの操作によるRTCの生成までの手順を説明する
  - rtc.confの設定
    - 「manager.is_master」を「YES」に設定して起動するマネージャをマスターに設定する
      - manager.is_master: YES
    - モジュール探索パスの設定
      - manager.modules.load_path: ., C:¥¥Program Files (x86)¥¥OpenRTM-aist¥¥1.1.2¥¥Components¥¥C++¥¥Examples¥¥vc12
  - 作成したrtc.confを設定ファイルの指定してrtcd.exeを起動する
    - rtcdはコマンドプロンプトからrtcd.exeを入力するか、OpenRTM-aistをインストールしたフォルダからコピーして使用する
    - rtcdはマネージャの起動のみを行う
      - ～Comp.exeは起動時に特定のコンポーネントの起動も行う
    - RT Syetem Editorのネームサービスビューにマネージャが表示される



マネージャの操作

- モジュールのロード

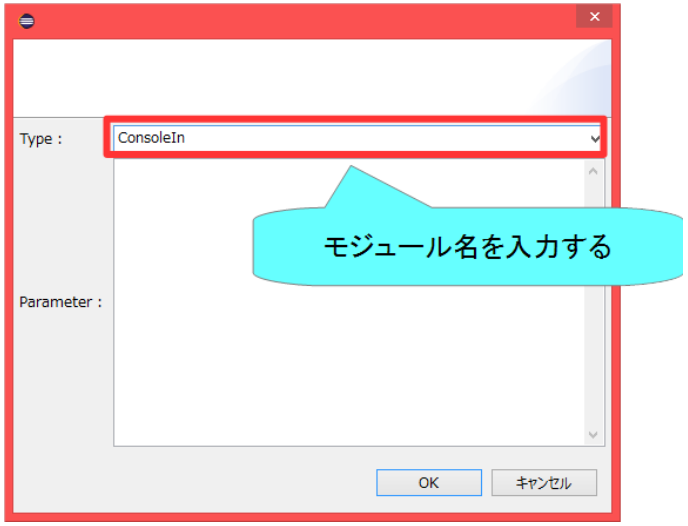


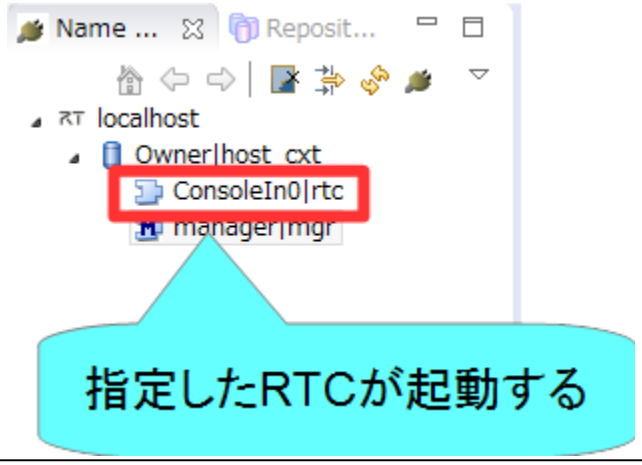
マネージャの操作

- モジュールのロード

マネージャの操作

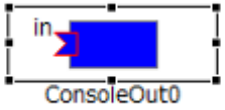
- RTCの生成





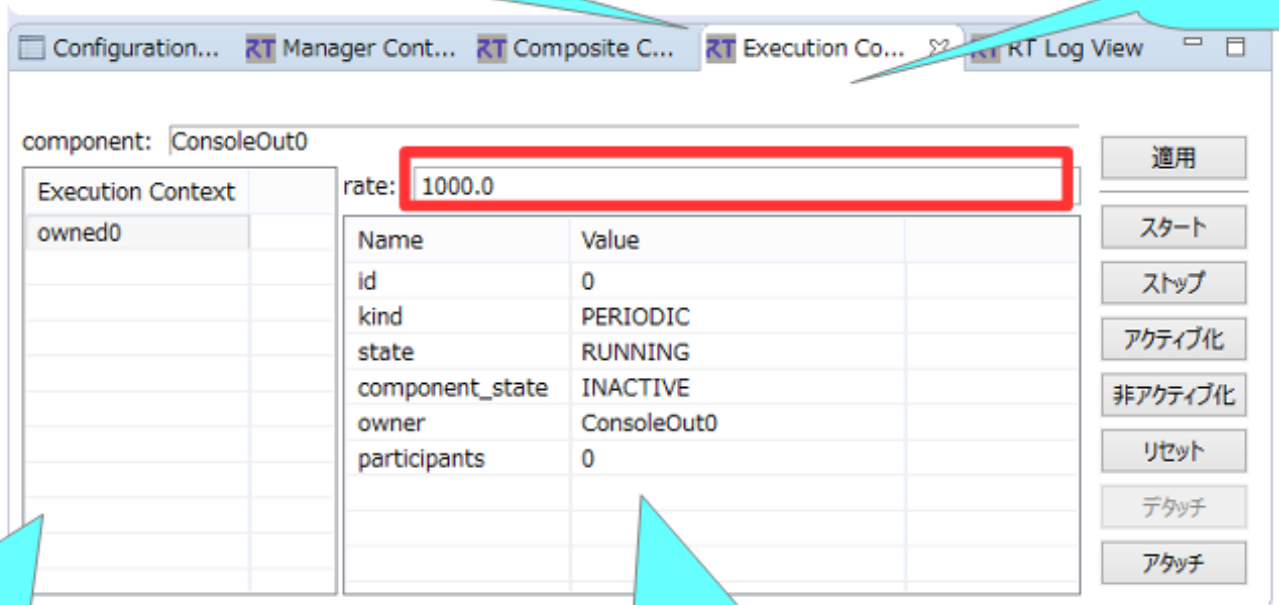
実行コンテキストの操作

RTCをクリック



「Execution Context View」タブを選択

実行周期

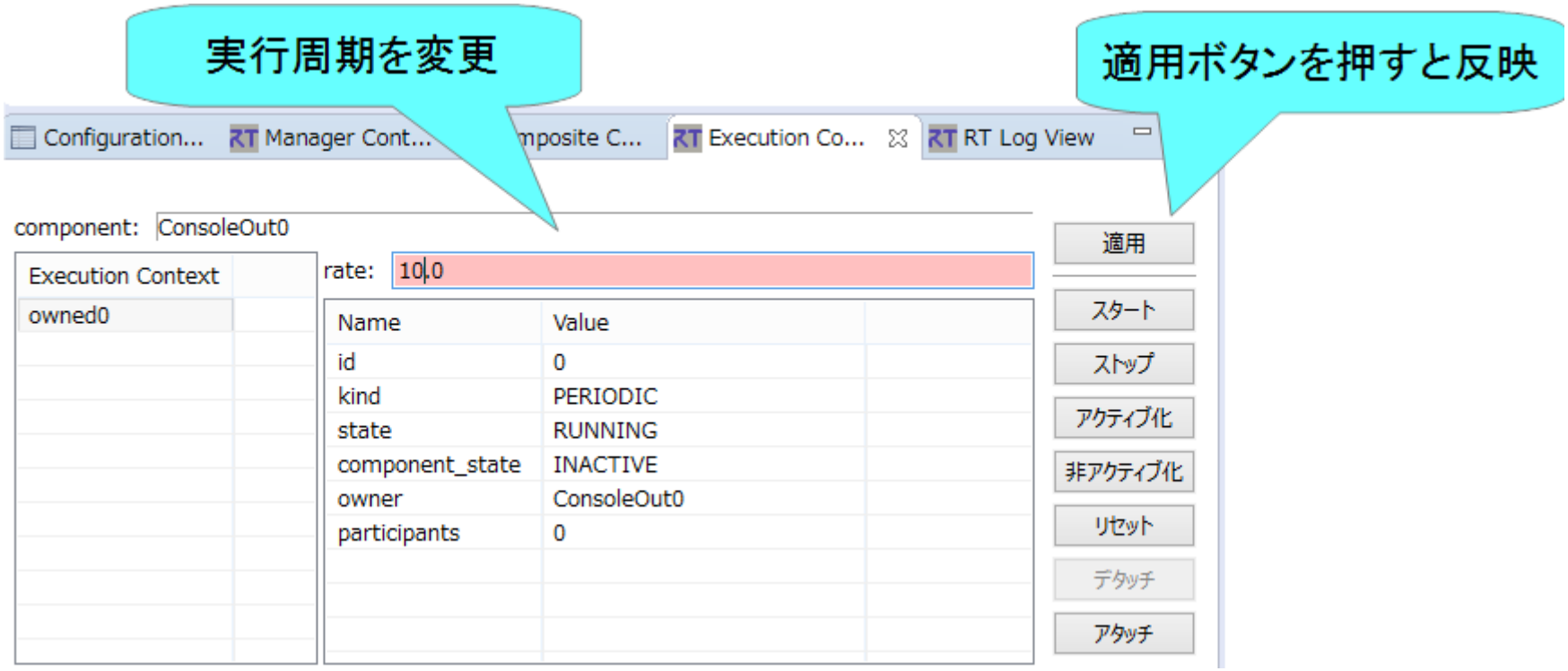


関連付けている実行コンテキスト一覧

実行コンテキストの情報

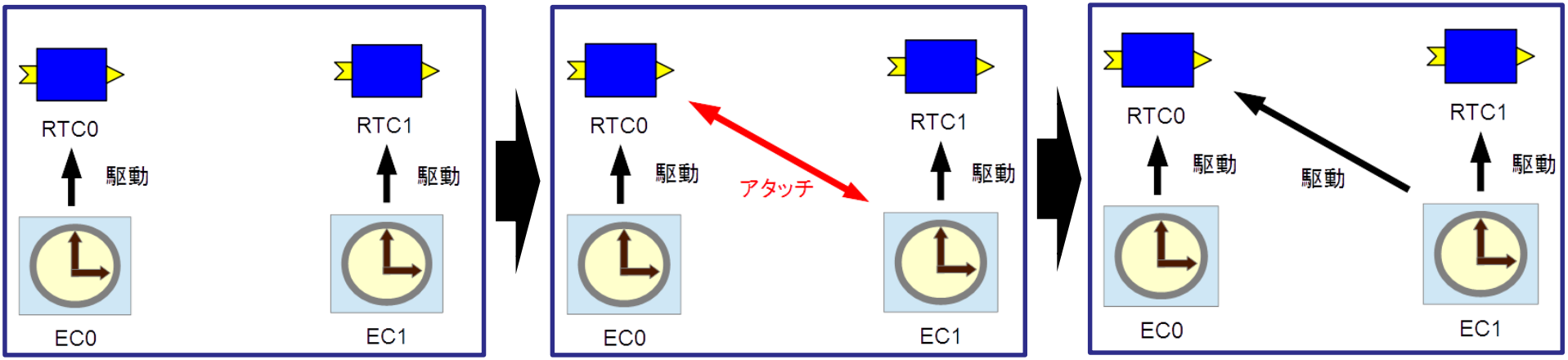
実行コンテキストの操作

- 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



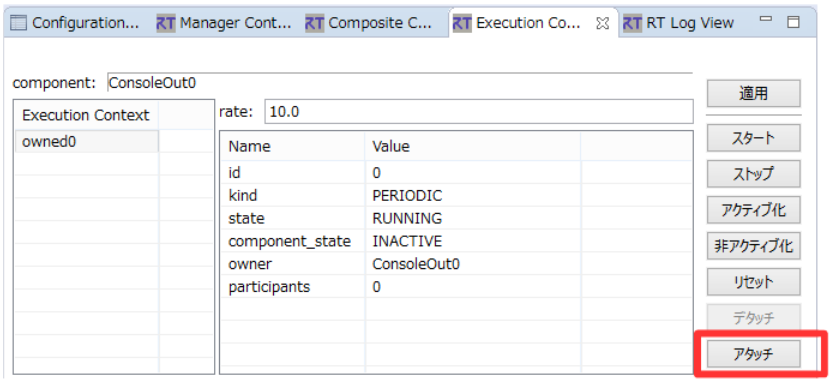
RTC0とRTC1は別々の実行コンテキストで駆動

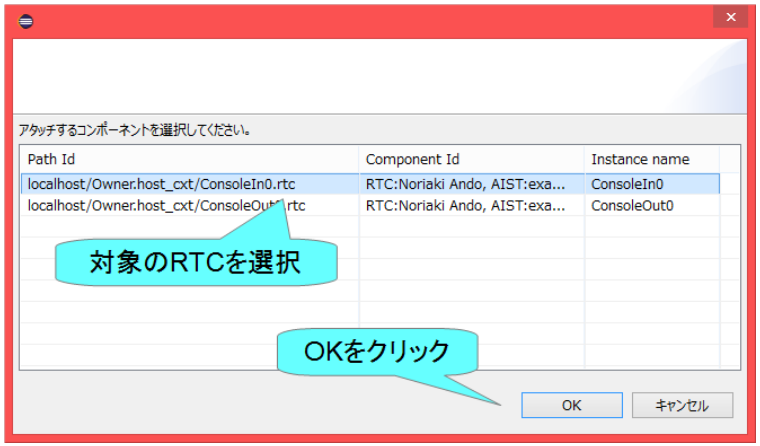
RTC0とEC1を関連付ける

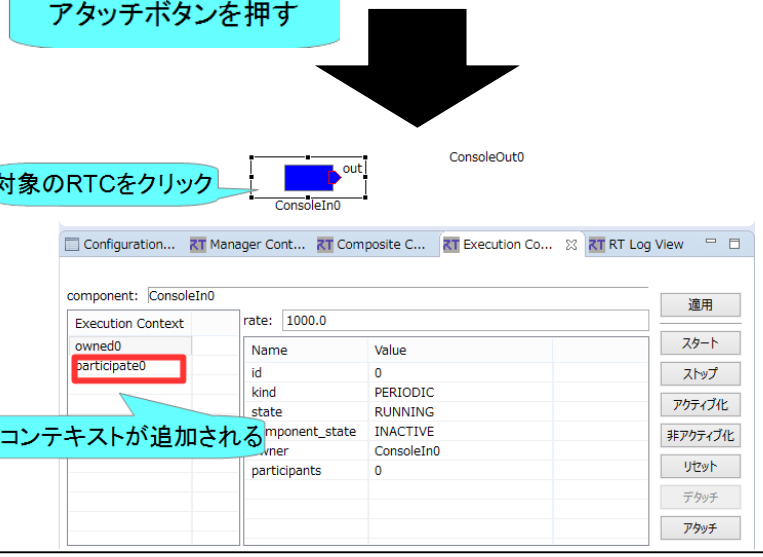
EC1がRTC0も駆動する

実行コンテキストの操作

- 実行コンテキストの関連付け

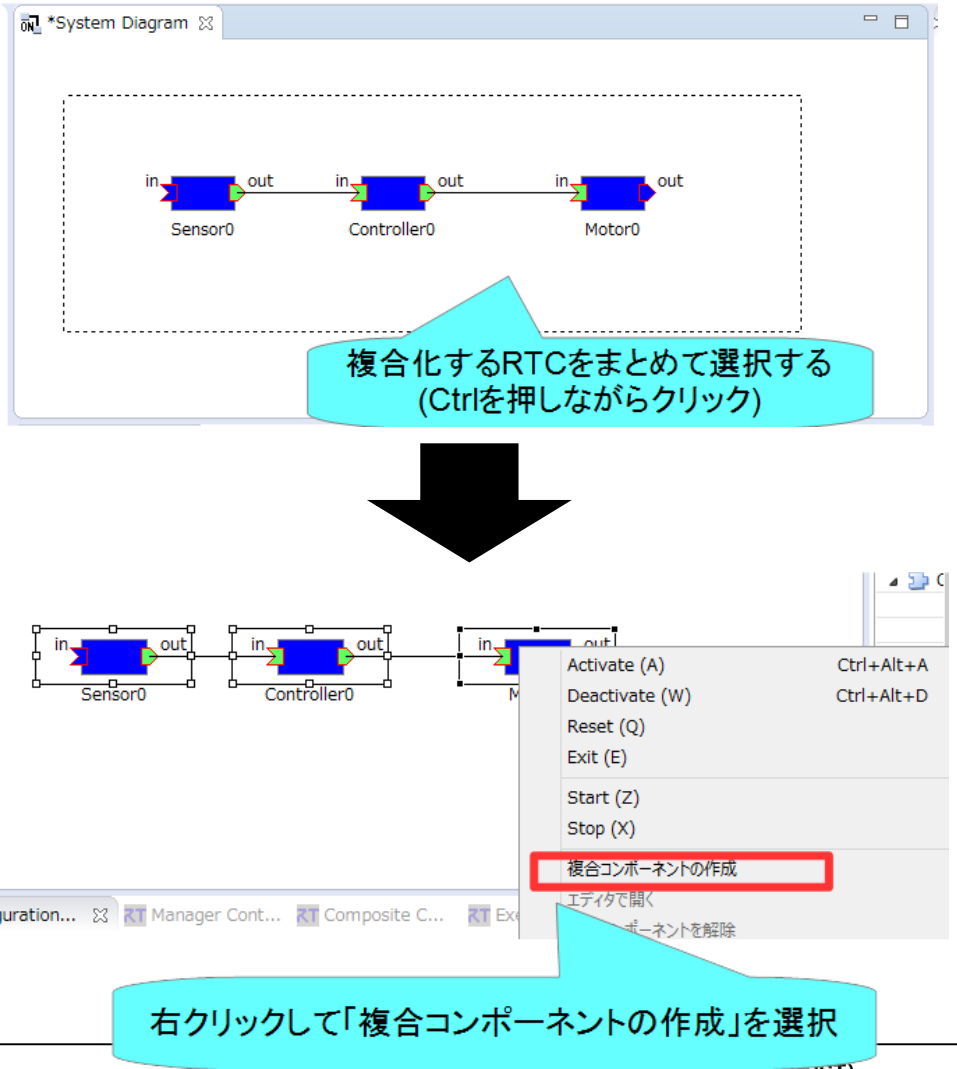






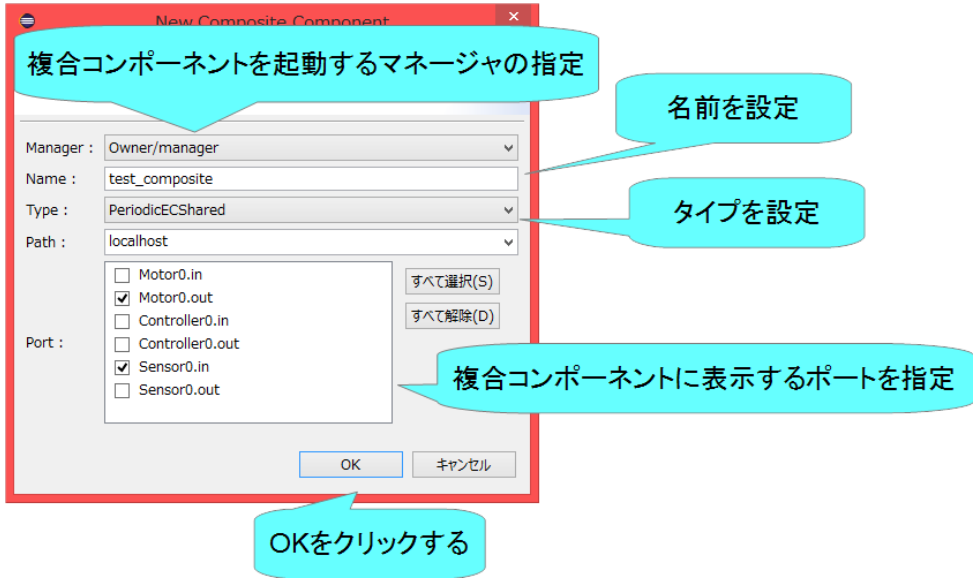
複合コンポーネントの操作

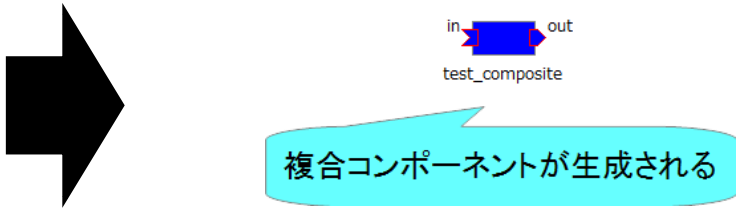
- 複合コンポーネントの生成



複合コンポーネントの操作

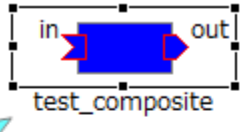
- 複合コンポーネントの生成





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

複合コンポーネントの操作



複合コンポーネントをクリック

「Composite Component View」タブを選択

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

component: test_composite type: PeriodicECShared

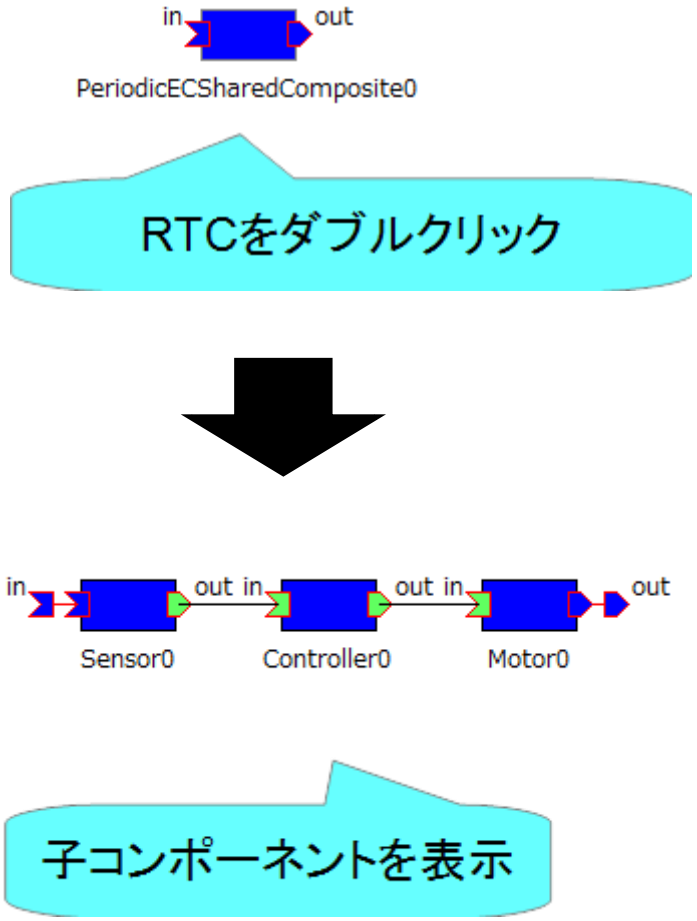
	component	port
☐	Controller0	Controller0.in
☐	Controller0	Controller0.out
☑	Sensor0	Sensor0.in
☐	Sensor0	Sensor0.out
☐	Motor0	Motor0.in
☑	Motor0	Motor0.out

適用 キャンセル

適用ボタンを押すと変更を反映

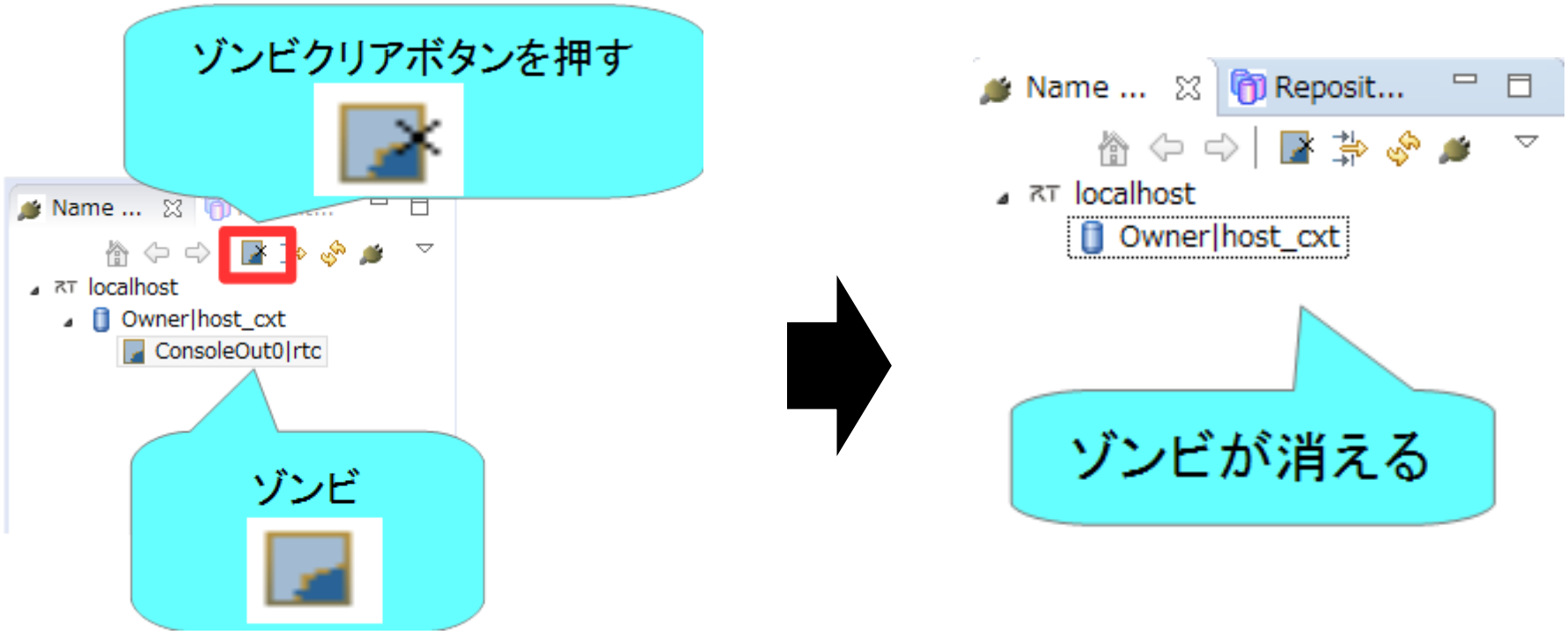
表示するポートの選択

複合コンポーネントの操作



ゾンビの削除

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



RT System Editorに関する設定