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)

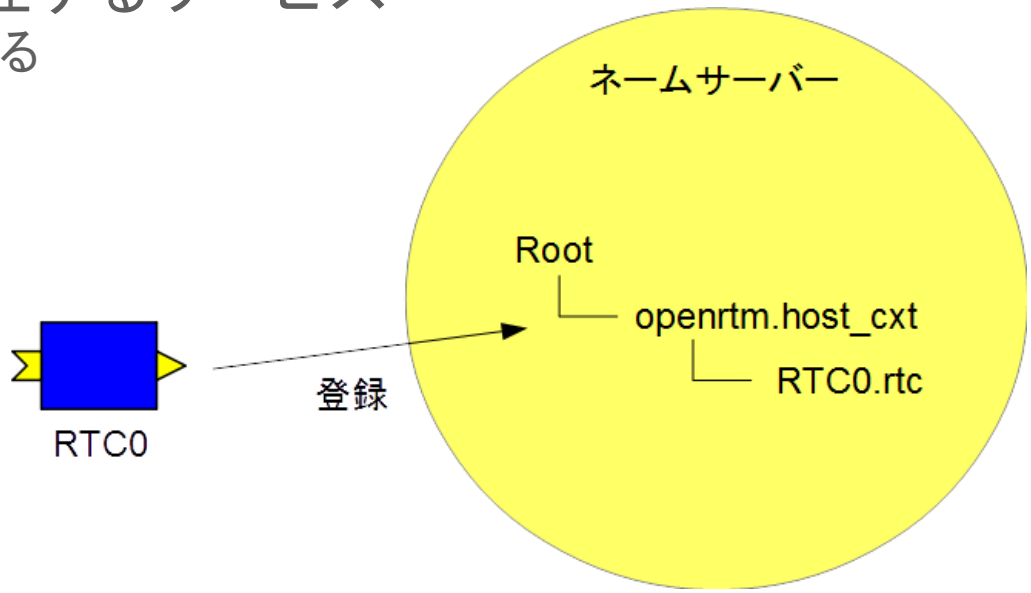
- OpenRTM-aist
  - OpenRTM-aist-1.1.2-RELEASE_x86.msi
  - インストール後に再起動する
  - Visual Studio 2013以外(2010、2012、2015)を使用する場合は環境変数を変更
    - 「RTM_VC_VERSION」をvc10、vc11、vc13
- Python
  - python-2.7.10.msi
    - 2.7.11は府外が発生するため非推奨
  - ※OpenRTM-aistの32bit版をインストールする場合Pythonも32bit版をインストールする。
    OpenRTM-aistの64bit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

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



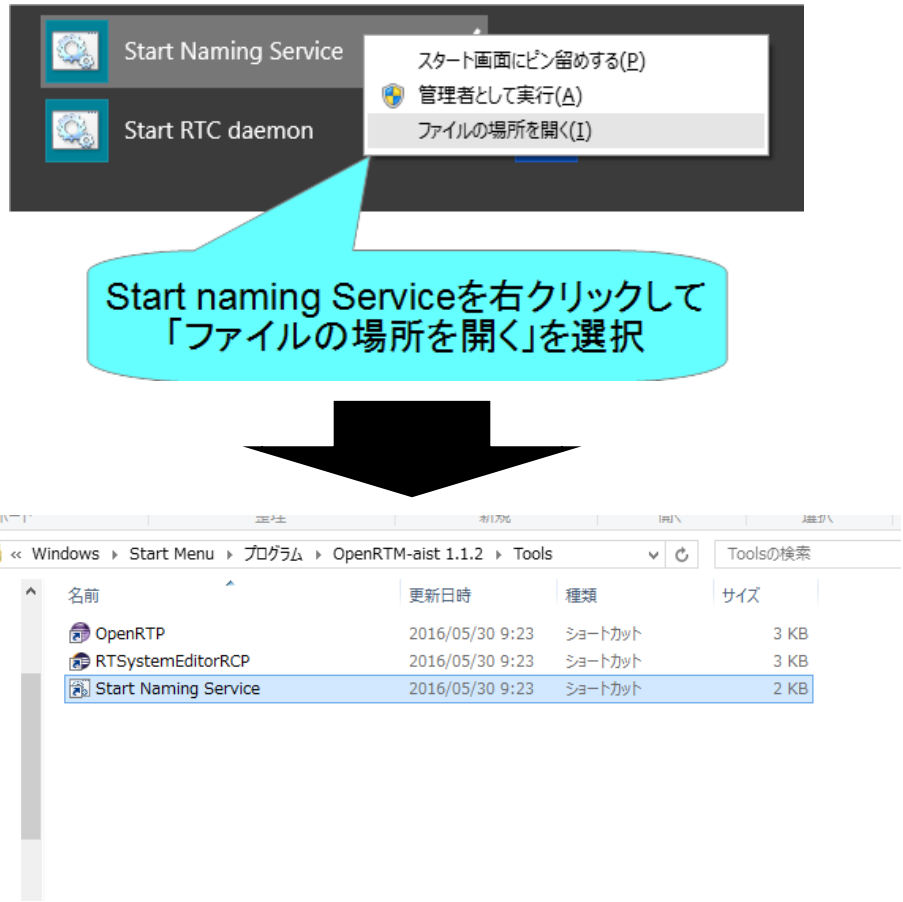
- 起動する手順
  - Windows 7
    - 「スタート」→「すべてのプログラム」→「OpenRTM-aist 1.1.2」→「Tools」→「Start Naming Service」
  - Windows 8.1
    - 「スタート」→「アプリビュー(右下矢印)」→「OpenRTM-aist 1.1.2」→「Start Naming Service」
  - Ubuntu
    - $ rtm-naming

ネームサーバーの起動

- Windows 8.1

ネームサーバーの起動

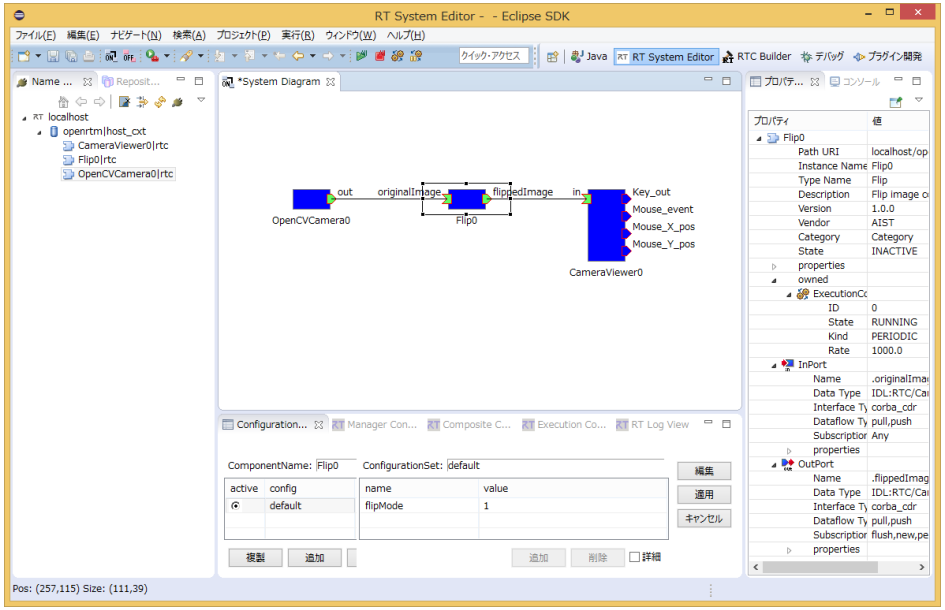
- いちいちアプリビューから起動するのは非常に手間がかかるため、以下の作業をしてスタートメニューのフォルダを開いておくことをお勧めします。



システム構築支援ツール
RT System Editorについて

RT System Editor

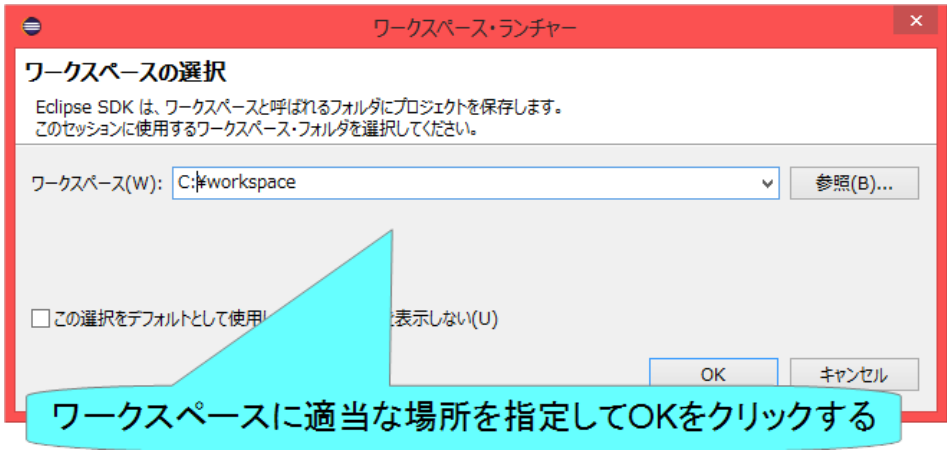
- RTCをGUIで操作するためのツール
  - データポート、サービスポートの接続
  - アクティブ化、非アクティブ化、リセット、終了
  - コンフィギュレーションパラメータの操作
  - 実行コンテキストの操作
    - 実行周期変更
    - 実行コンテキストの関連付け
  - 複合化
  - マネージャからRTCを起動
  - 作成したRTシステムの保存、復元



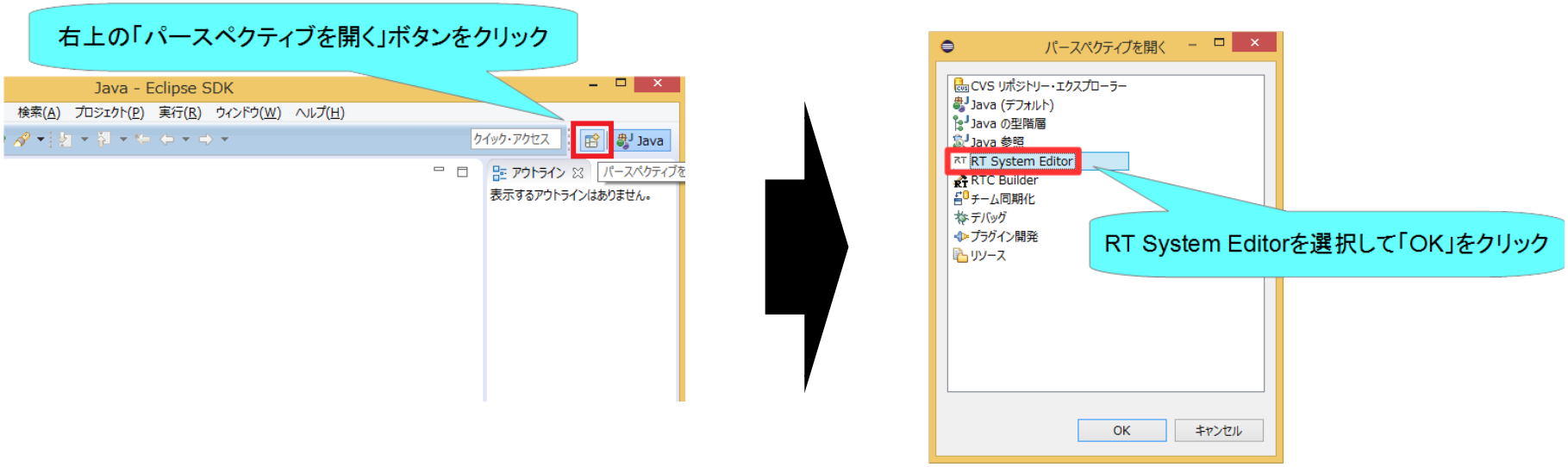
RT System Editorの起動

- 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」
    - ※先ほどの作業でスタートメニューのフォルダを開いている場合はOpenRTPのショートカットをダブルクリックしても開けます
    - ※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

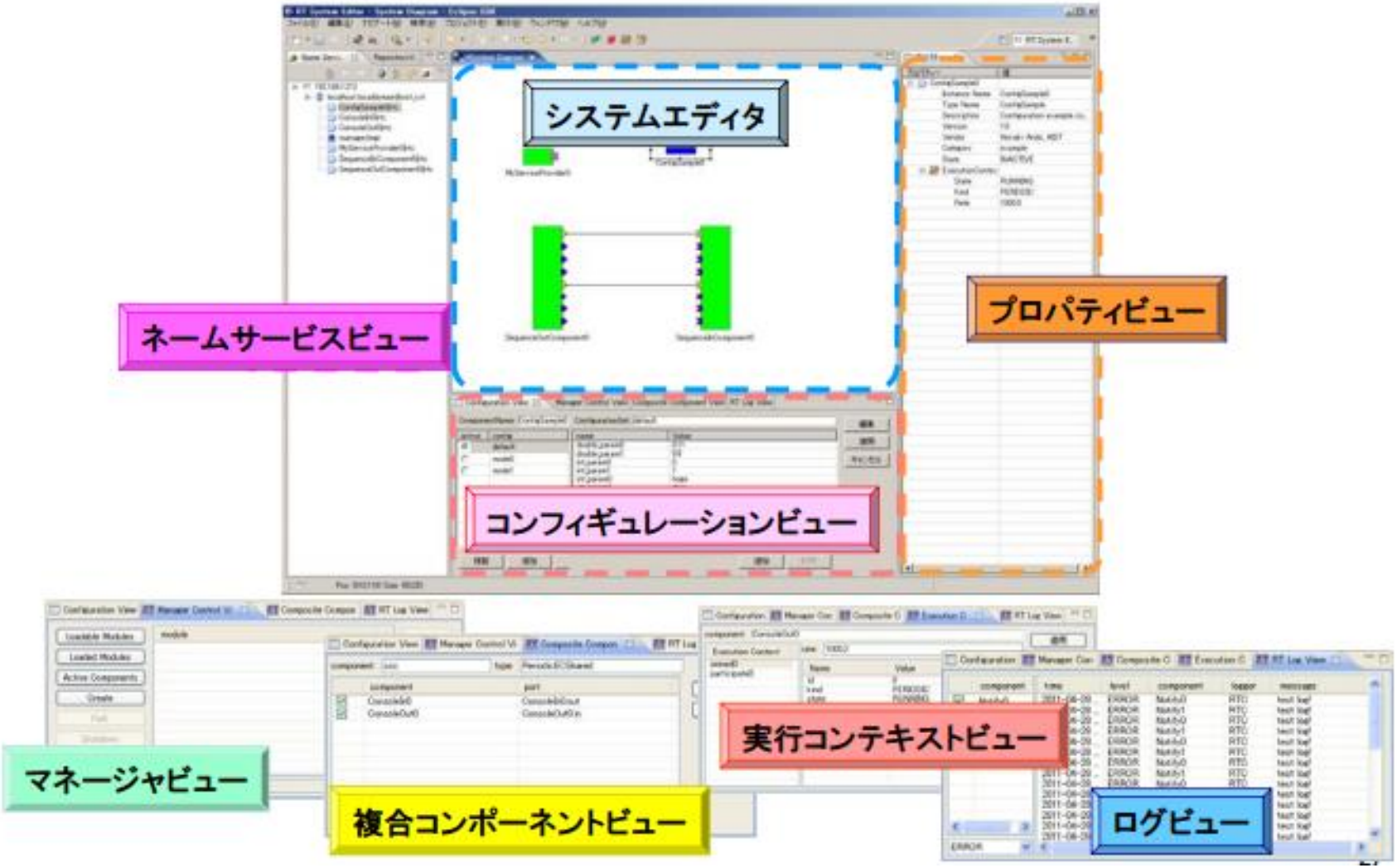
RT System Editorの起動



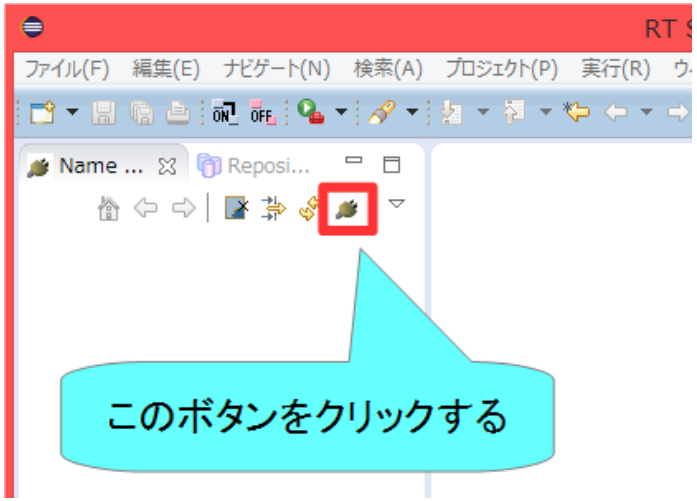
RT System Editorの起動

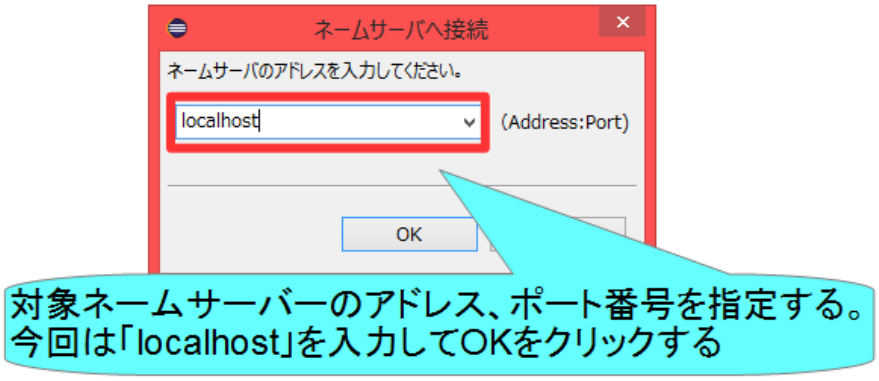


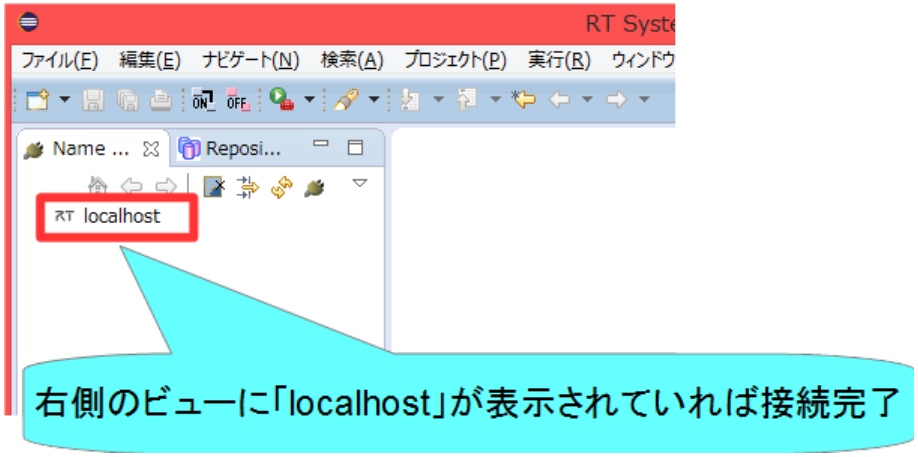
RT System Editorの画面構成



ネームサーバーへ接続

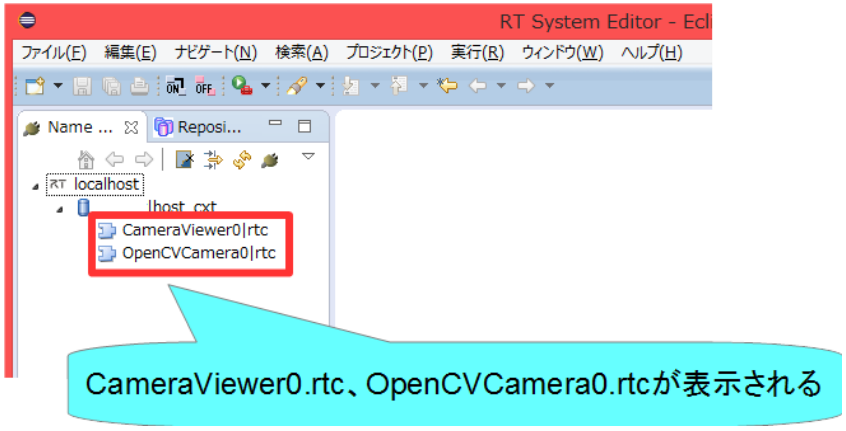

ファイル(F) 編集(E) ナビゲート(N) 検索(A) プロジェクト(P) 実行(R)
Name ...
Reposi...
このボタンをクリックする


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

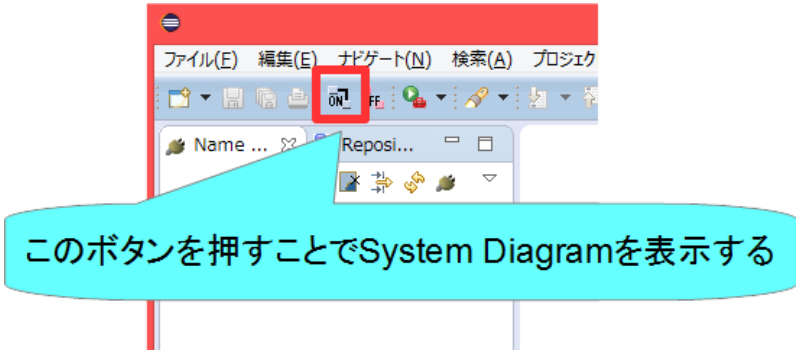

ファイル(F) 編集(E) ナビゲート(N) 検索(A) プロジェクト(P) 実行(R) ウィンドウ
Name ...
Reposi...
localhost
右側のビューに「localhost」が表示されていれば接続完了

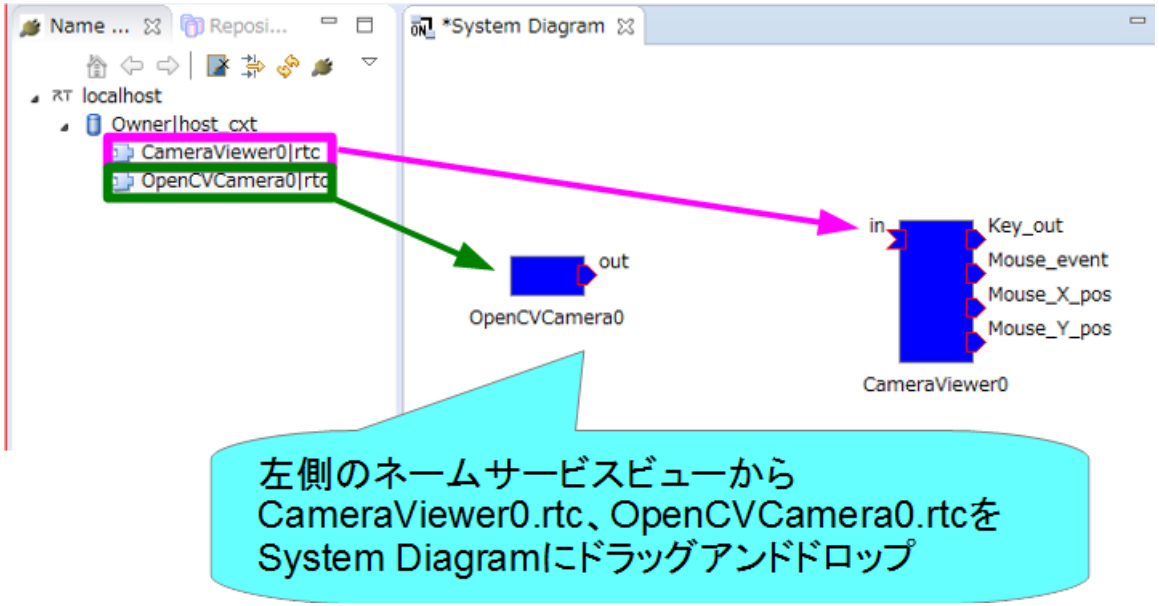
動作確認

- サンプルのRTCを起動して簡単な動作確認を行う
  - ポートの接続
  - アクティブ化
- サンプルのRTCの起動手順
  - CameraViewerコンポーネント、OpenCVCameraコンポーネントを起動
    - Windows 7
      - 「スタート」→「すべてのプログラム」→「OpenRTM-aist 1.1.2」→「Components」→「OpenCV-Examples」→「OpenCVCameraComp.exe」、「CameraViewerComp.exe」
    - Windows 8.1
      - 「スタート」→「アプリビュー(右下矢印)」→「OpenRTM-aist 1.1.2」→「OpenCVCameraComp.exe」、「CameraViewerComp.exe」
    - Ubuntu
      - $ /usr/share/openrtm-1.1/components/c++/opencv-rtcs/CameraViewerComp
      - $ /usr/share/openrtm-1.1/components/c++/opencv-rtcs/OpenCVCameraComp



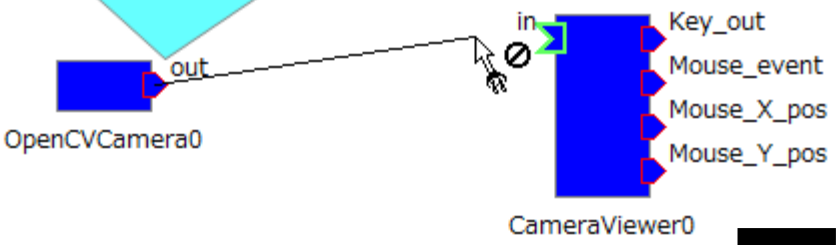
データポートの接続

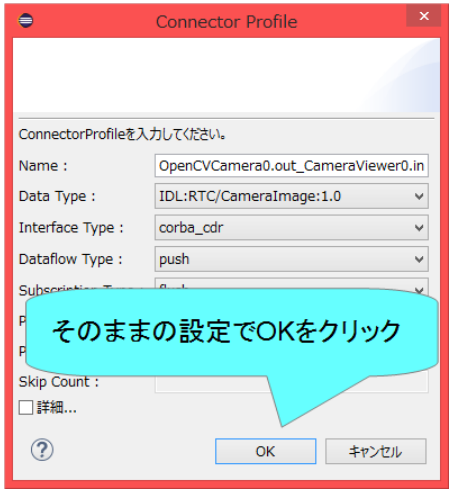

ファイル(F) 編集(E) ナビゲート(N) 検索(A) プロジェク
Name ...
Reposi...
このボタンを押すことでSystem Diagramを表示する


Name ...
Reposi...
*System Diagram
localhost
Owner|host_cxt
CameraViewer0|rtc
OpenCVCamera0|rtc
out
OpenCVCamera0
in
Key_out
Mouse_event
Mouse_X_pos
Mouse_Y_pos
CameraViewer0
左側のネームサービスビューから
CameraViewer0.rtc、OpenCVCamera0.rtcを
System Diagramにドラッグアンドドロップ

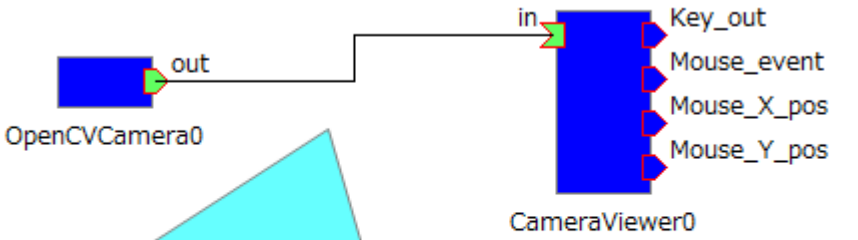
データポートの接続

OpenCVCamera0の「out」を選択して、
CameraViewer0の「in」にドラッグアンドドロップ



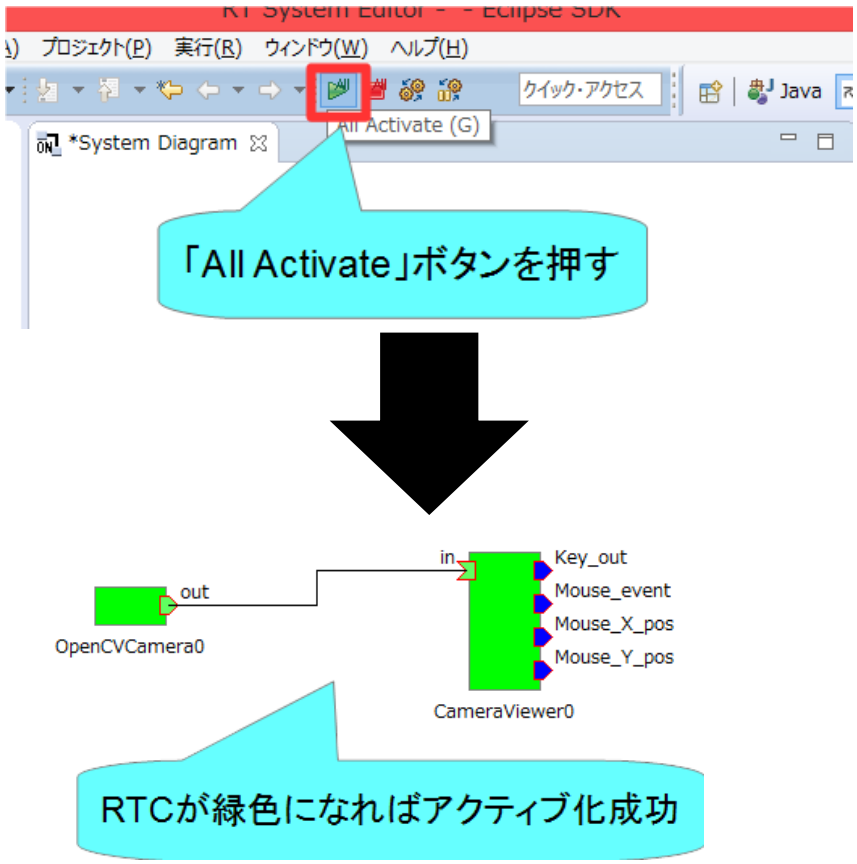


※コネクタプロファイルについては後で説明します



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

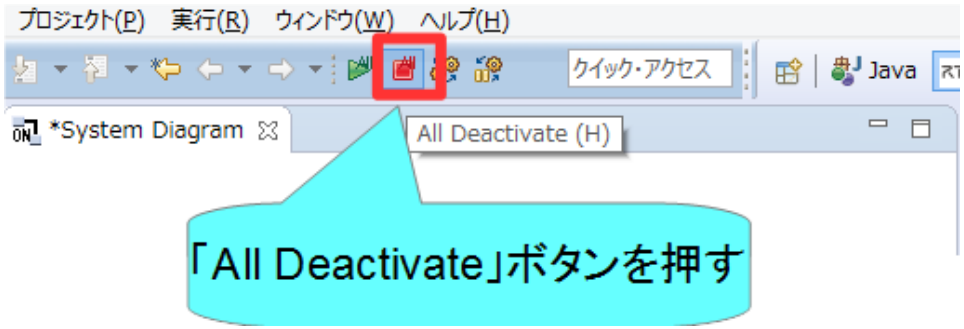
アクティブ化



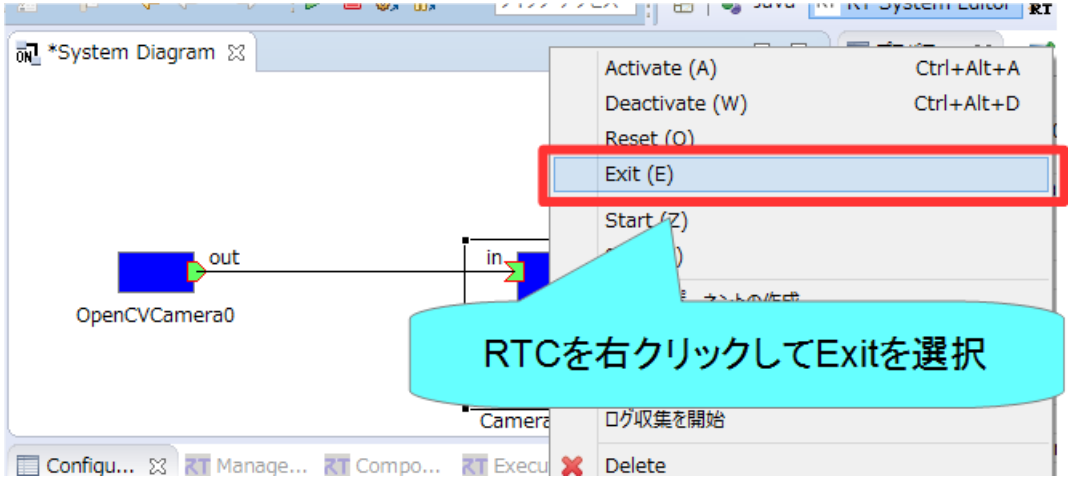
- カメラでキャプチャした画像が表示されるかを確認してください
  - 表示されない場合
    - カメラがPCに接続されていない
    - データポートを接続していない
    - RTCがアクティブになっていない

非アクティブ化、終了

- 非アクティブ化

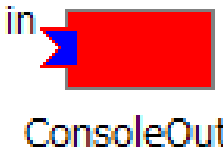


- 終了

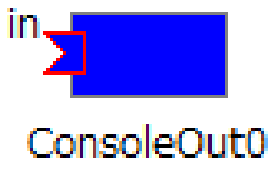


リセット

- 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。



- 以下の操作で非アクティブ状態に戻す



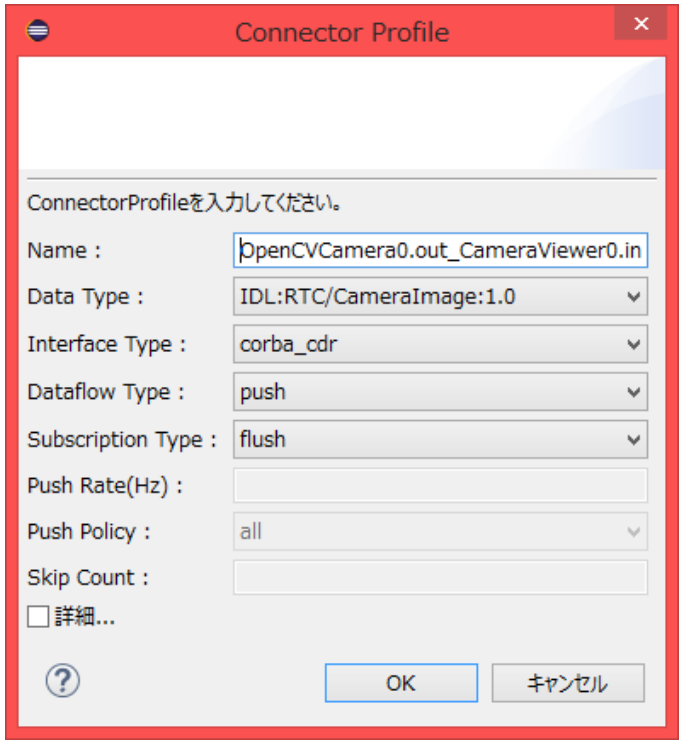
RT System Editor
補足

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

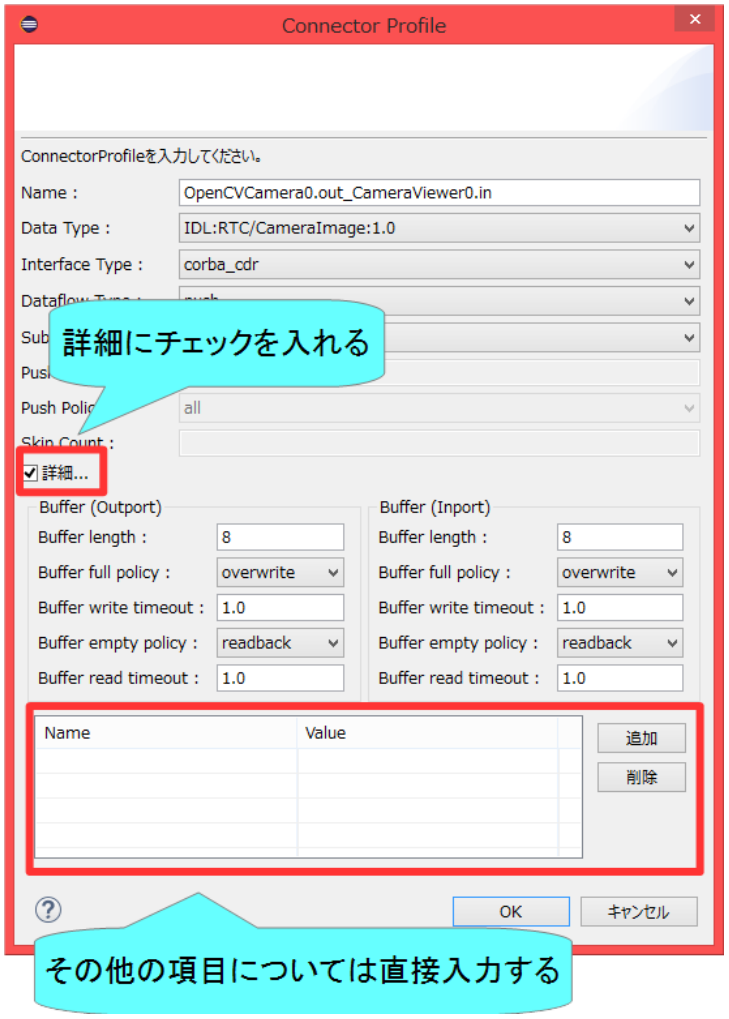
コネクタプロファイルの設定

- 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)

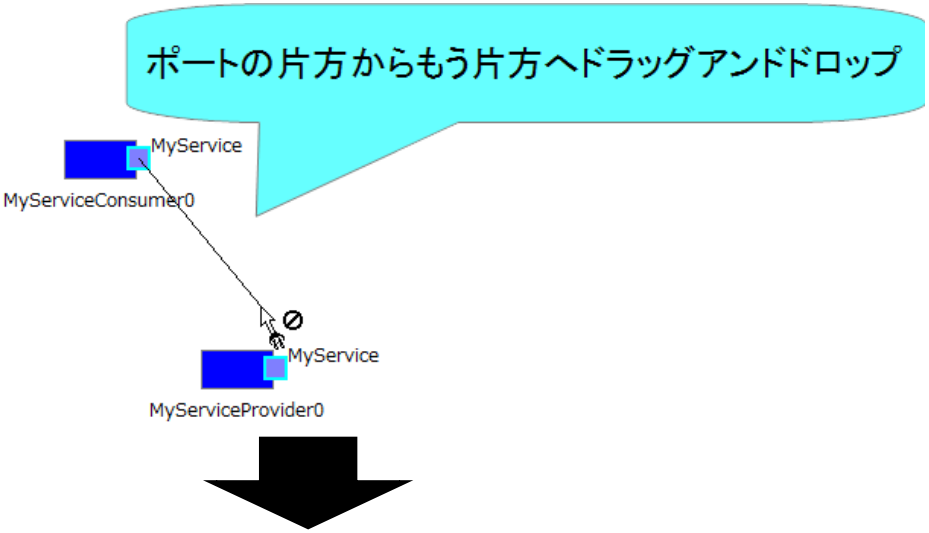


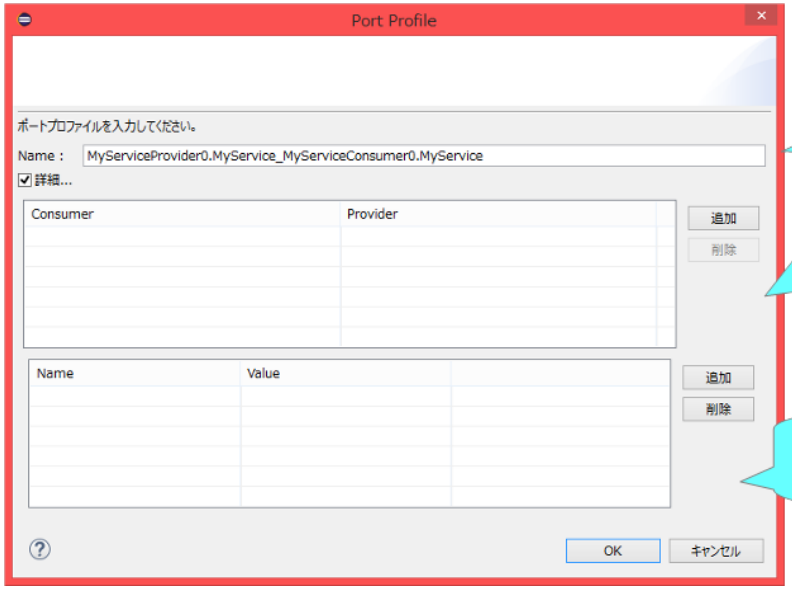
コネクタプロファイルの設定

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

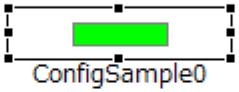
接続するインターフェースの設定
複数のインターフェースが定義されていた場合に、
どのインターフェースに接続するかを設定

その他の設定を直接入力

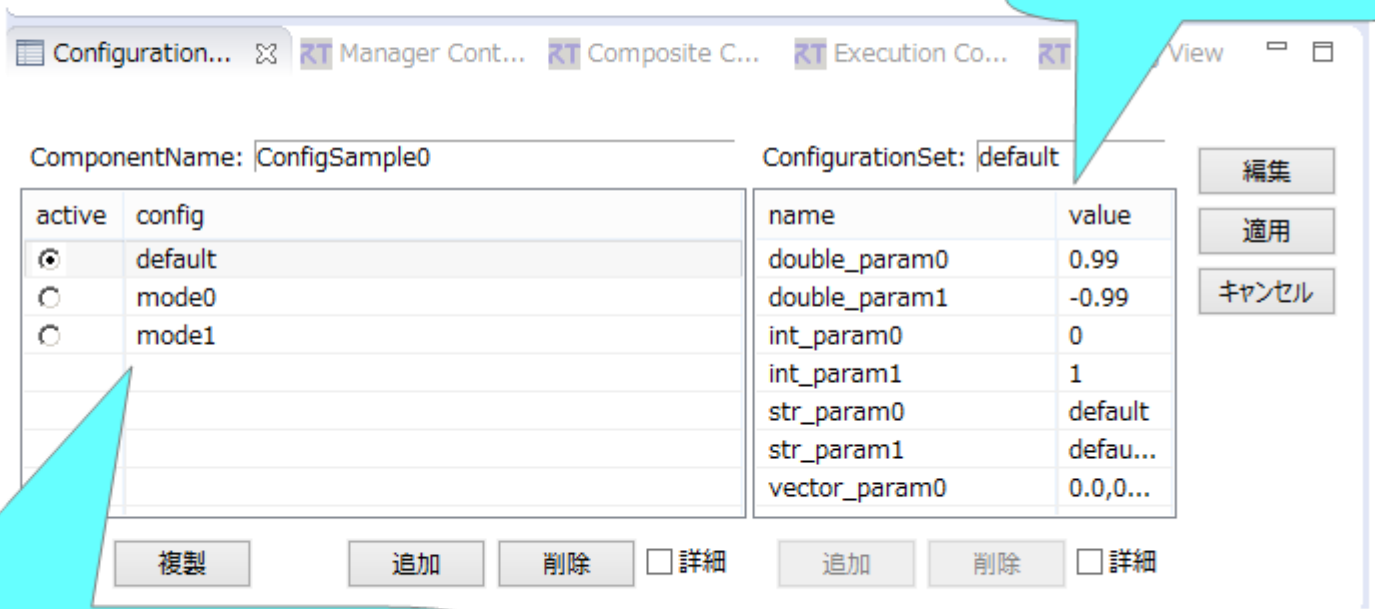
コンフィギュレーションパラメータの設定

対象のRTCをクリックすると表示

「Configuration View」タブを選択

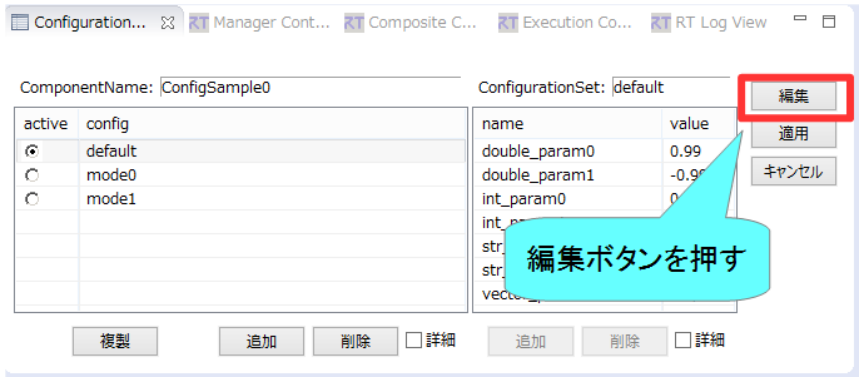


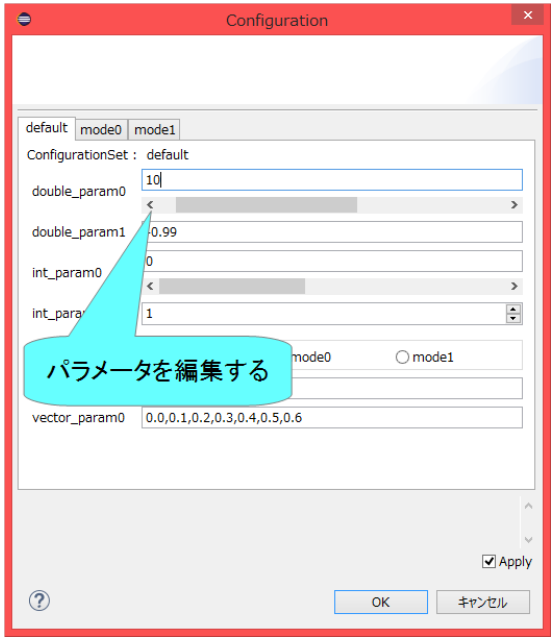
パラメーター一覧

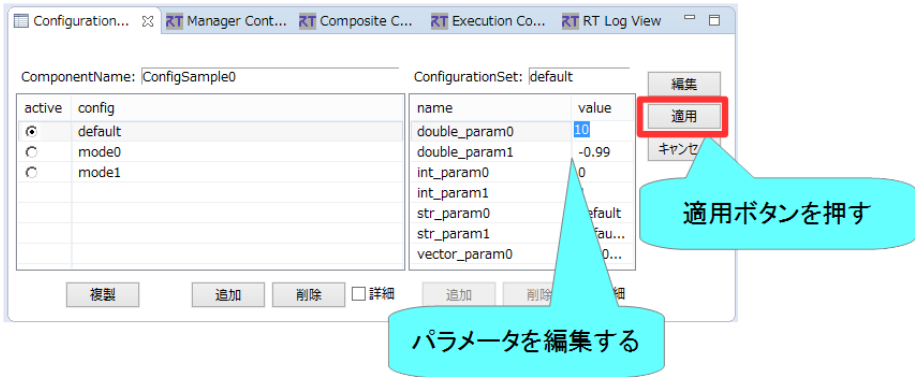


コンフィギュレーションセット一覧

コンフィギュレーションパラメータの設定

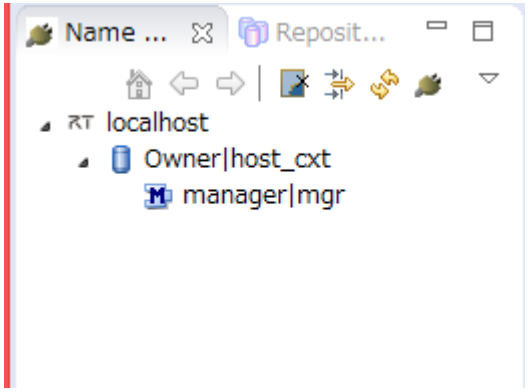






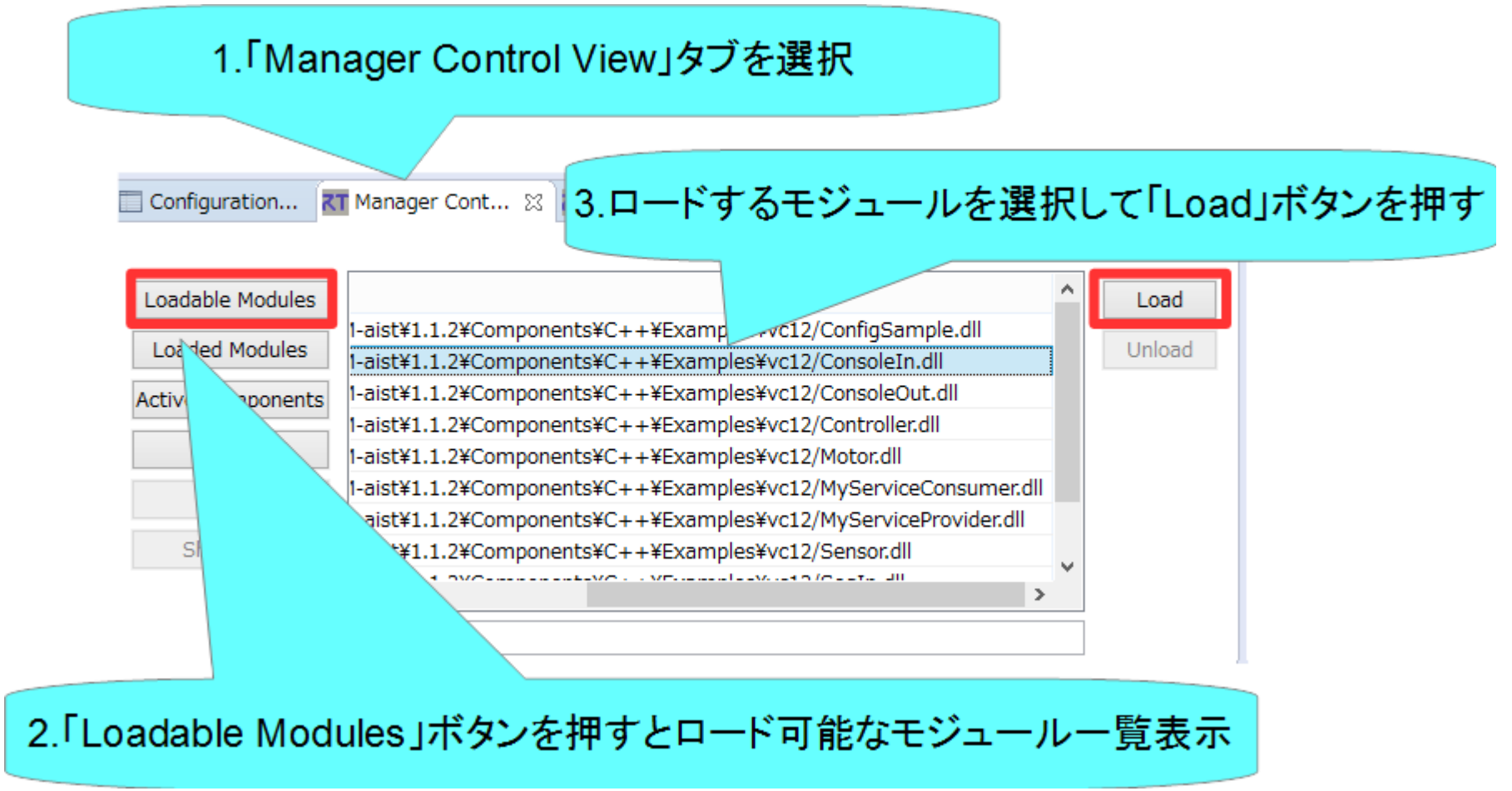
マネージャの操作

- 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
    - 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

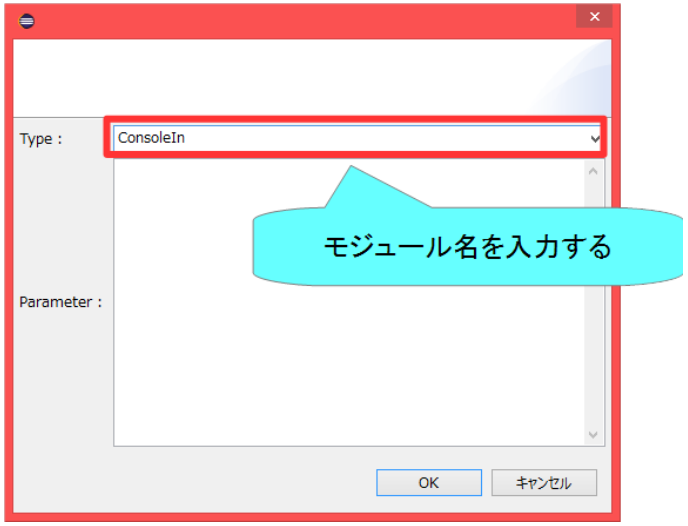


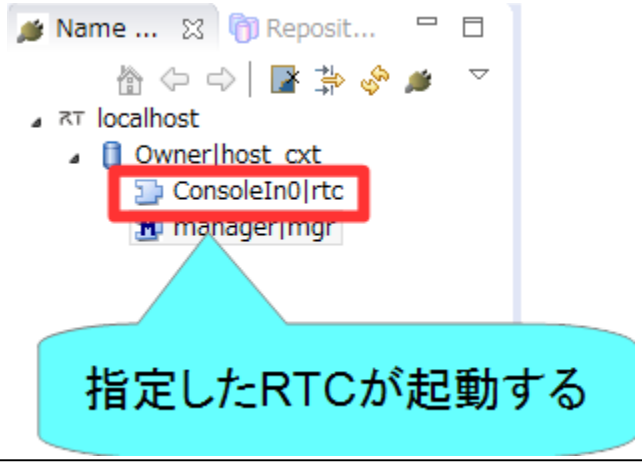
マネージャの操作

- モジュールのロード

マネージャの操作

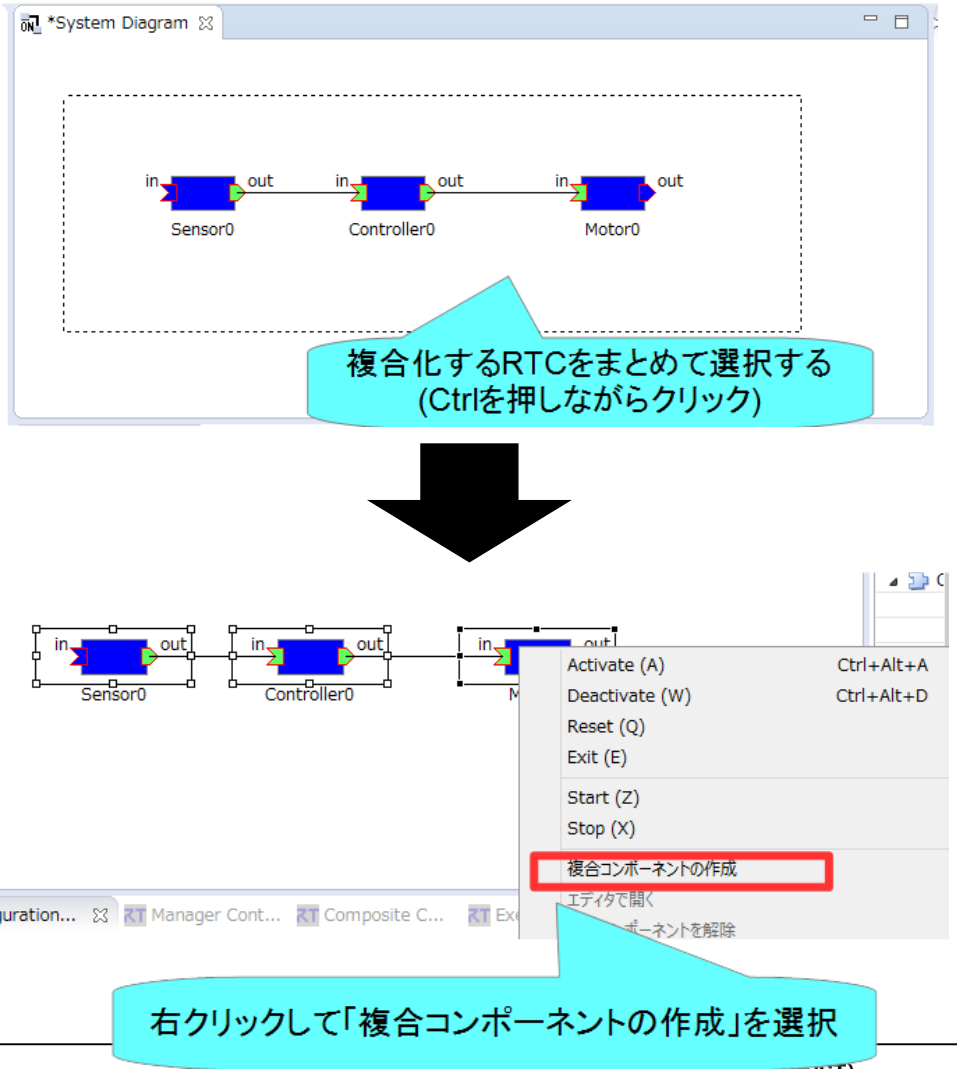
- RTCの生成





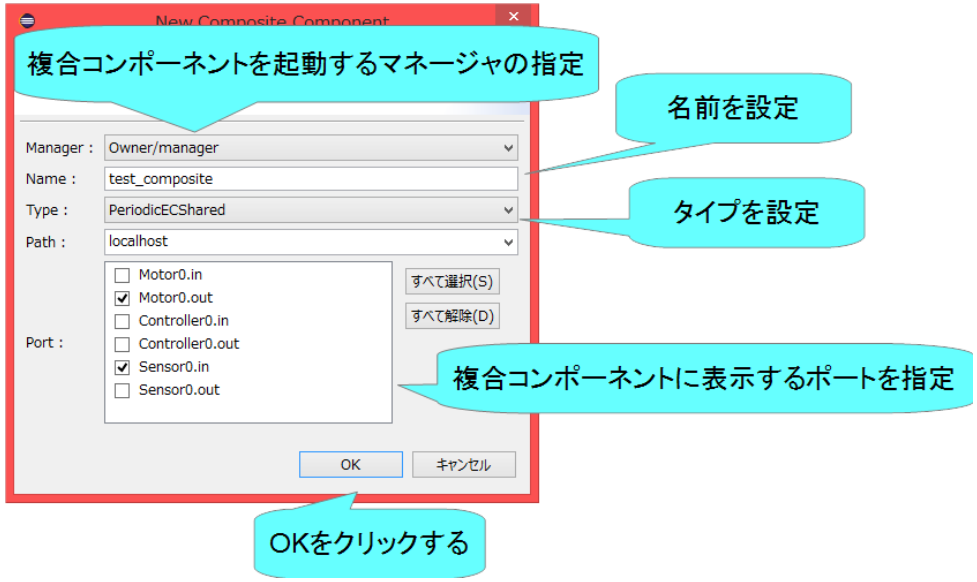
複合コンポーネントの操作

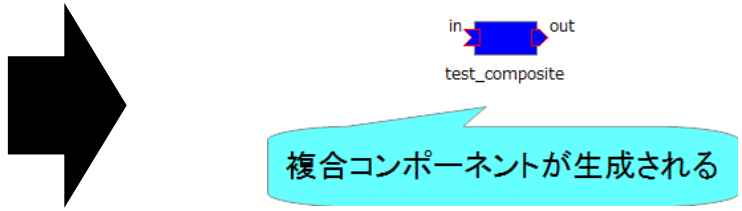
- 複合コンポーネントの生成



複合コンポーネントの操作

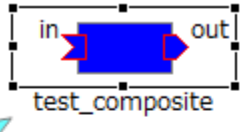
- 複合コンポーネントの生成





- 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

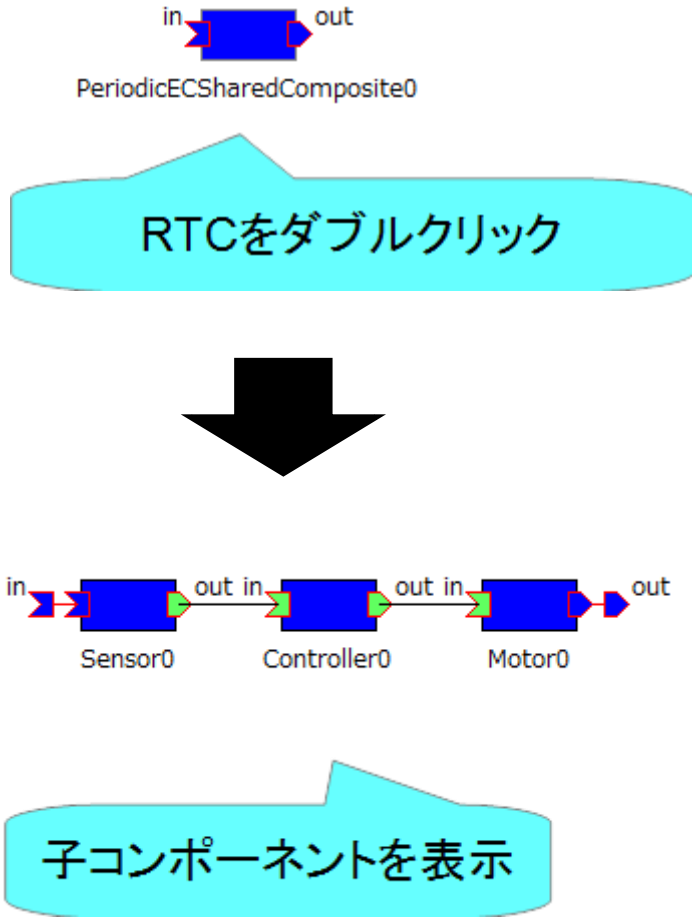
	component	port
☐	Controller0	Controller0.in
☐	Controller0	Controller0.out
☑	Sensor0	Sensor0.in
☐	Sensor0	Sensor0.out
☐	Motor0	Motor0.in
☑	Motor0	Motor0.out

適用 キャンセル

適用ボタンを押すと変更を反映

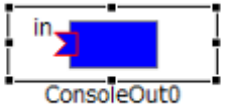
表示するポートの選択

複合コンポーネントの操作



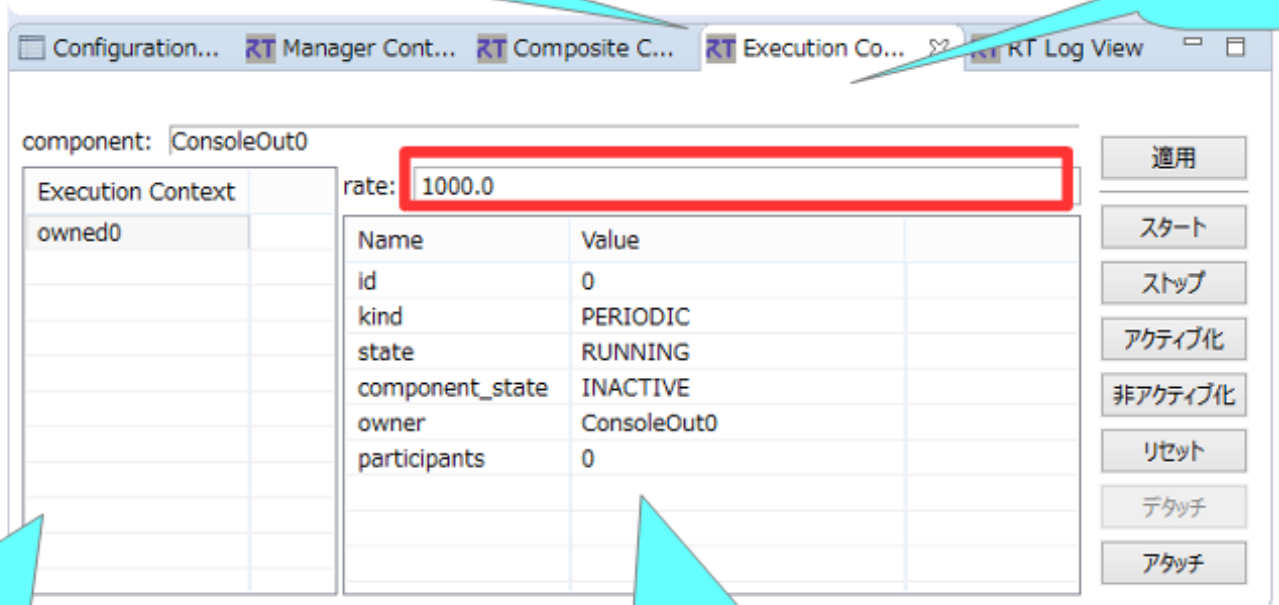
実行コンテキストの操作

RTCをクリック



「Execution Context View」タブを選択

実行周期

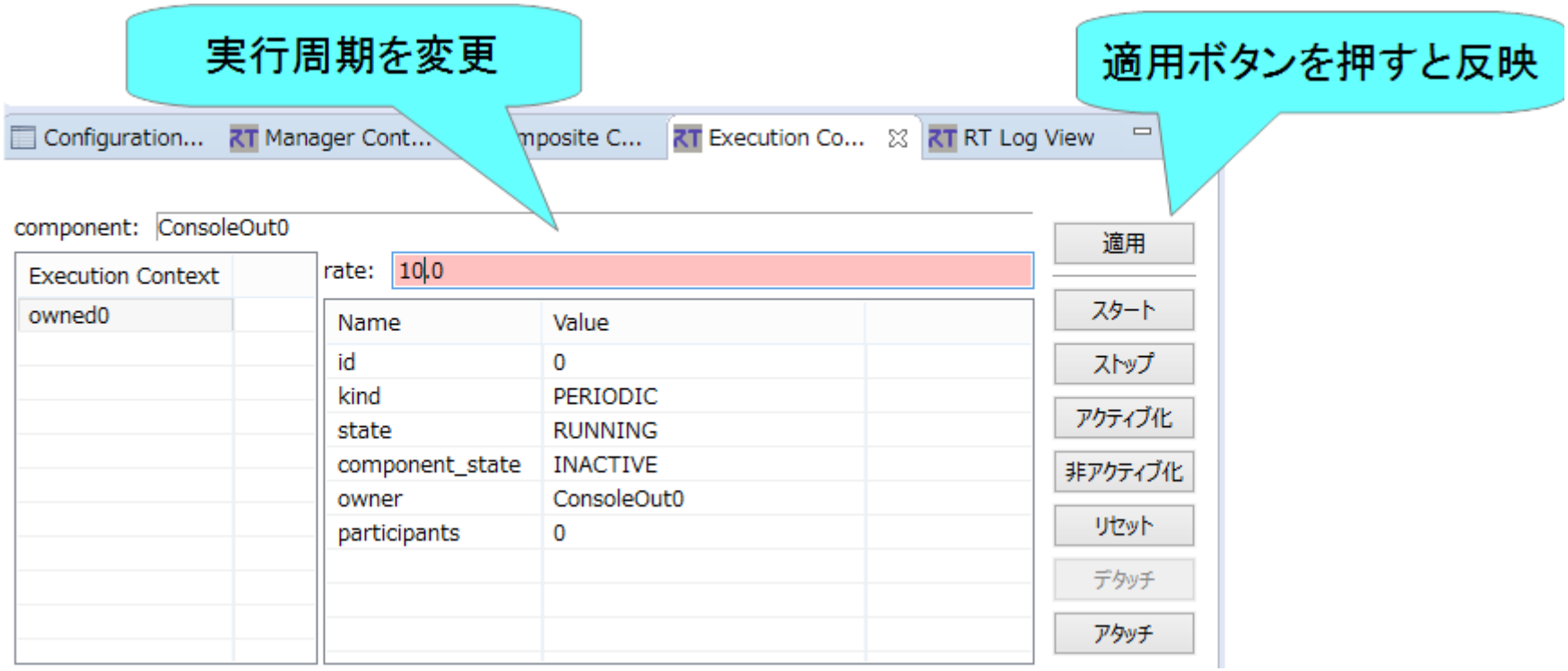


関連付けている実行コンテキスト一覧

実行コンテキストの情報

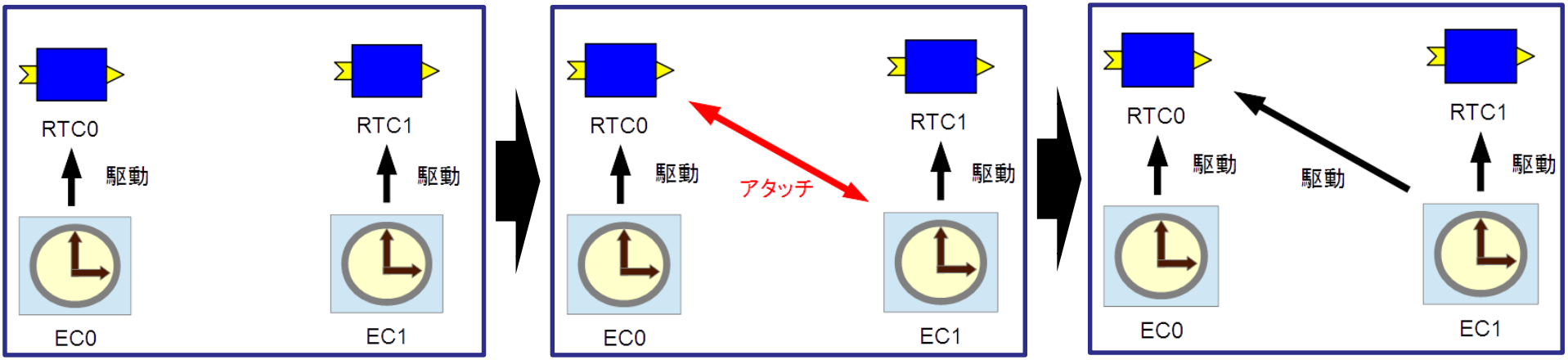
実行コンテキストの操作

- 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



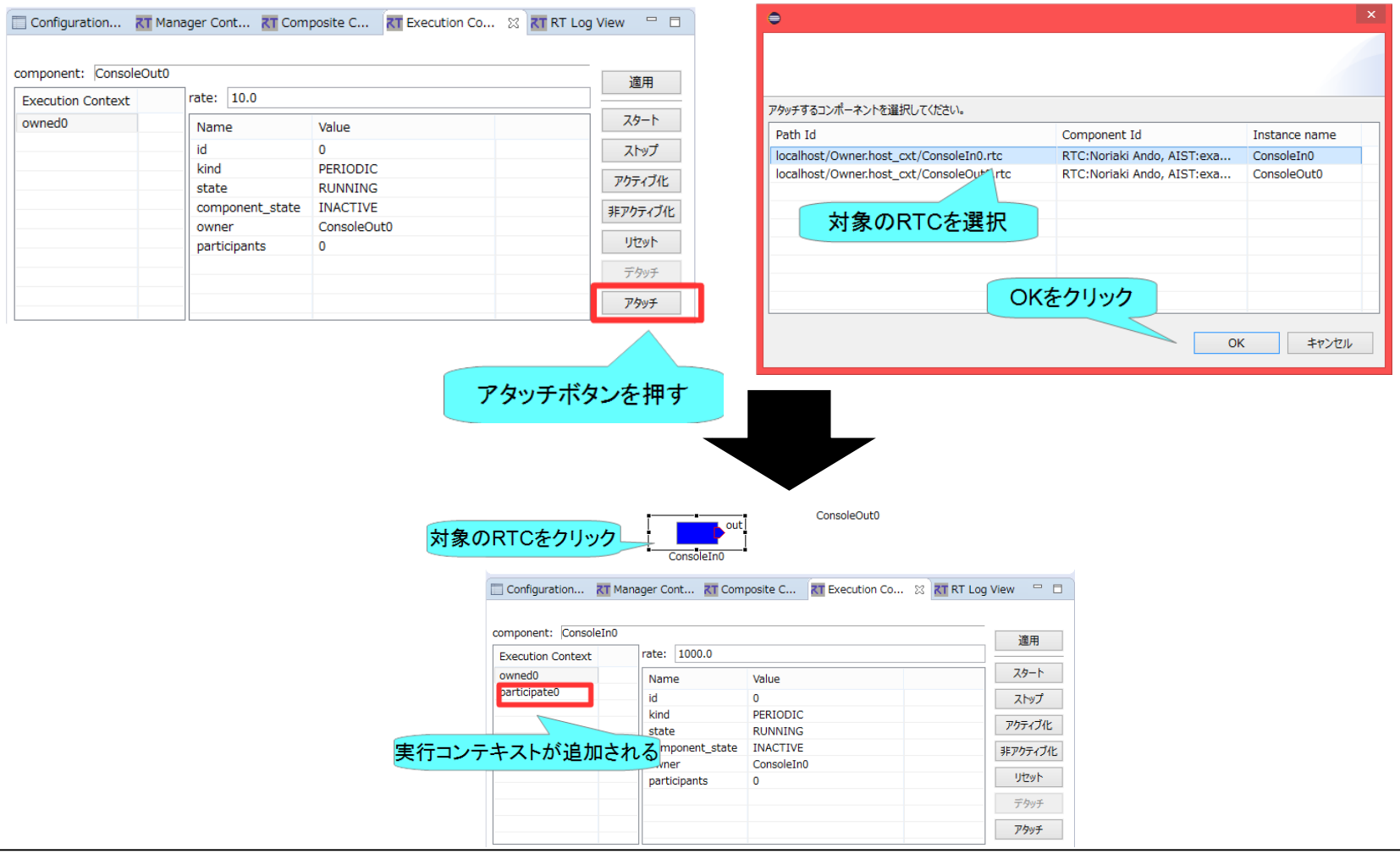
RTC0とRTC1は別々の実行コンテキストで駆動

RTC0とEC1を関連付ける

EC1がRTC0も駆動する

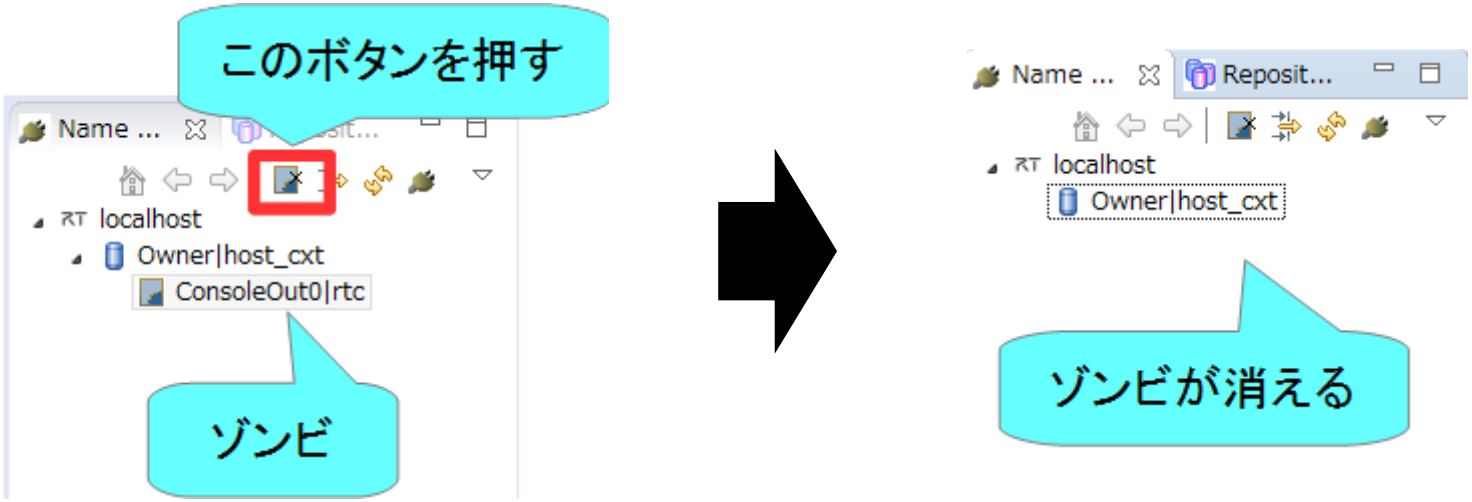
実行コンテキストの操作

- 実行コンテキストの関連付け

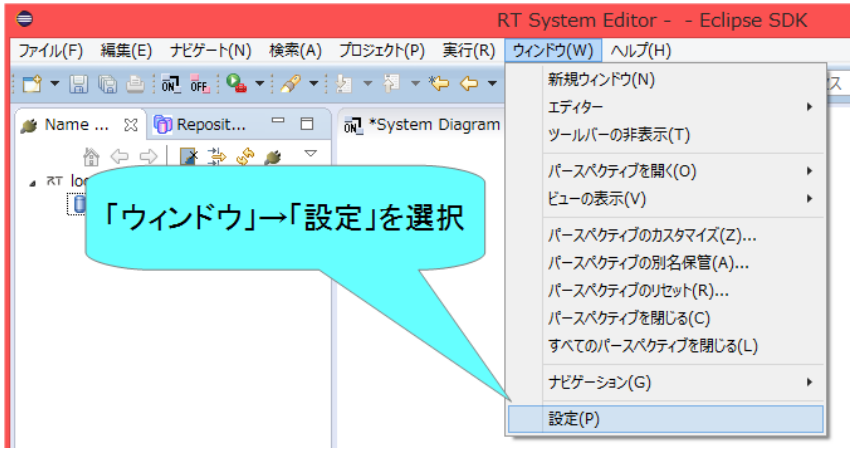


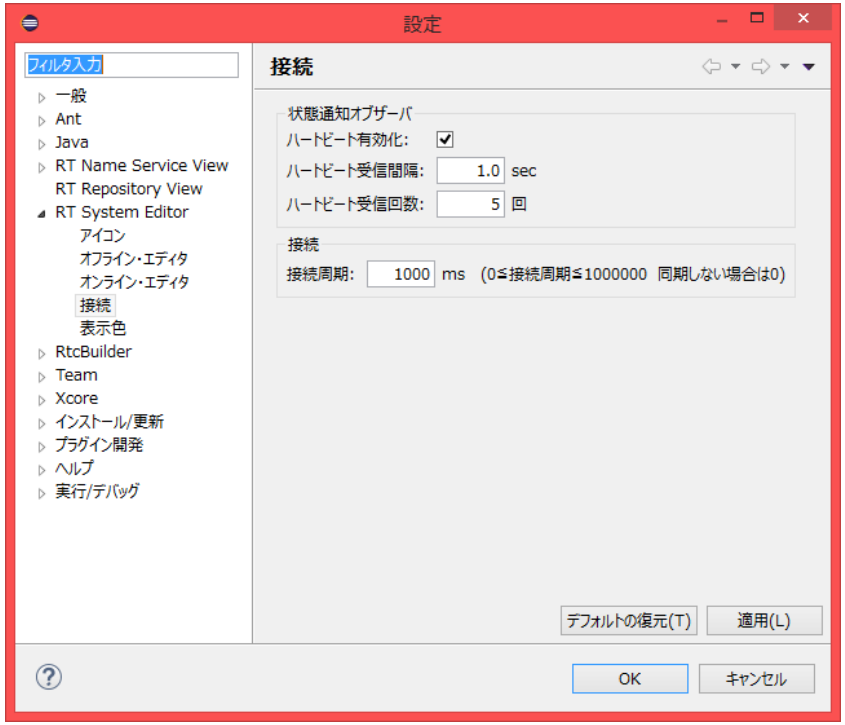
ゾンビの削除

- 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

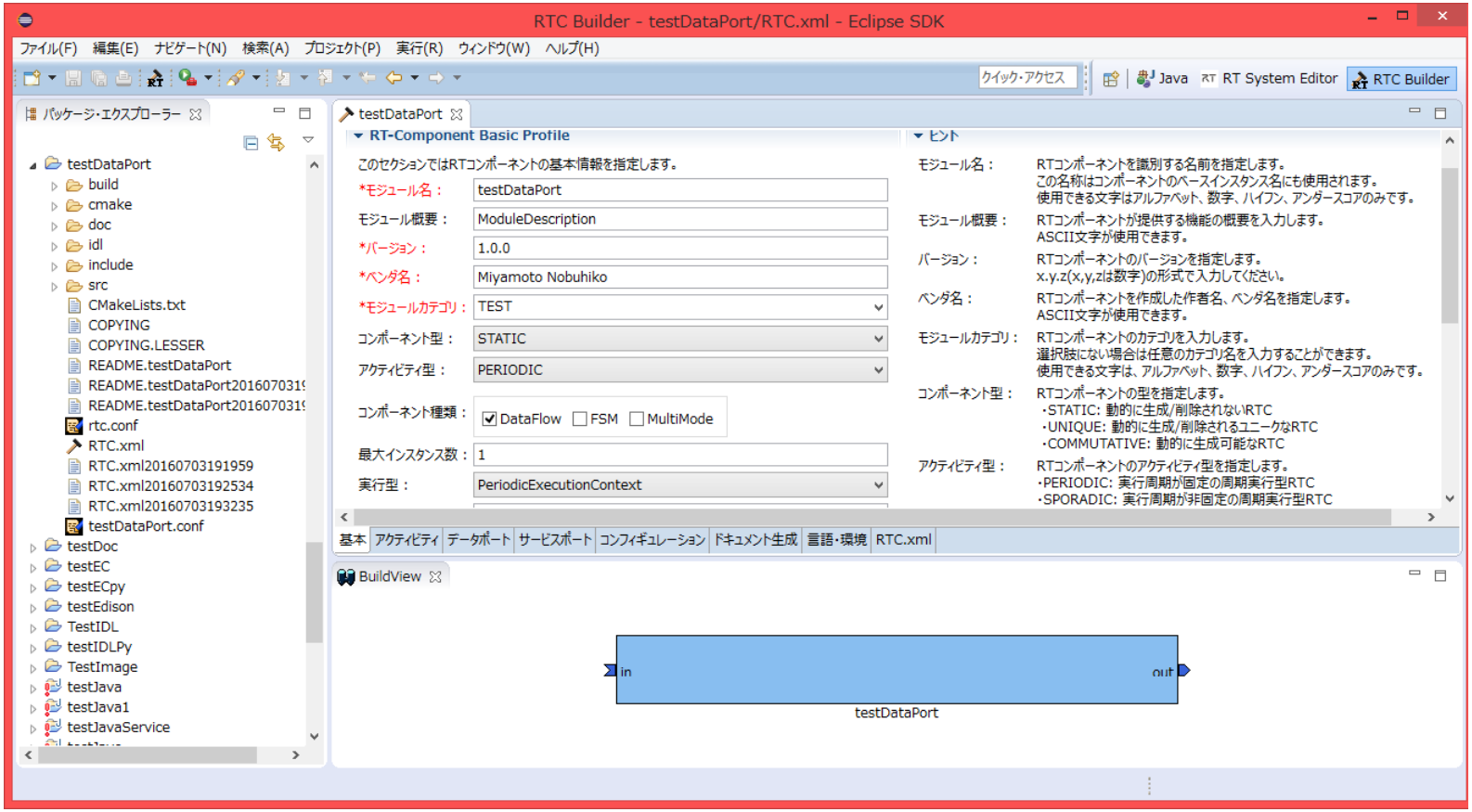




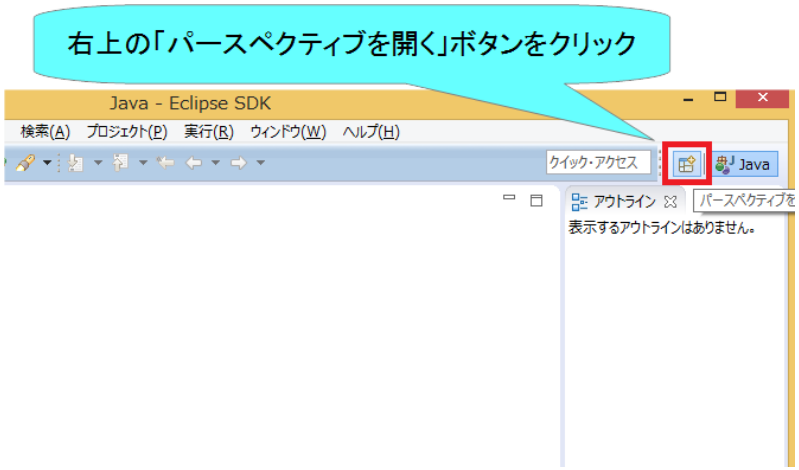
コンポーネント開発ツール RTC Builderについて

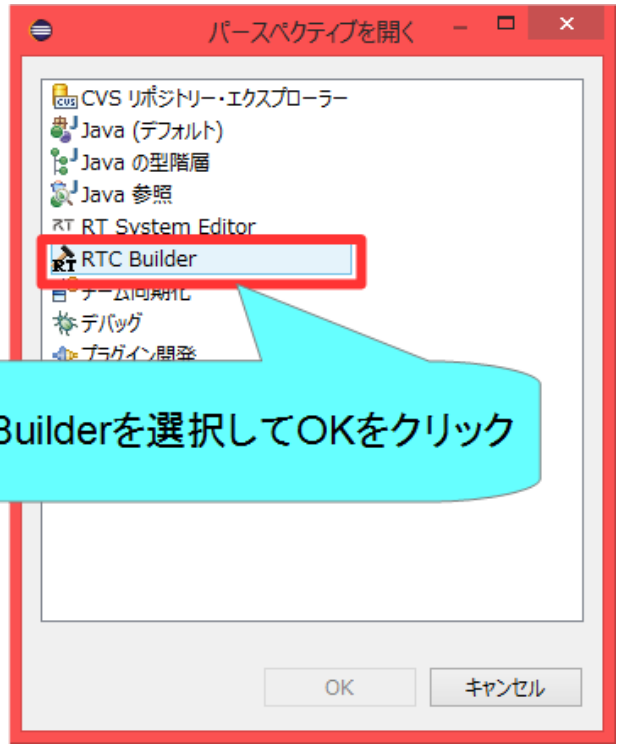
RTC Builder

- 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



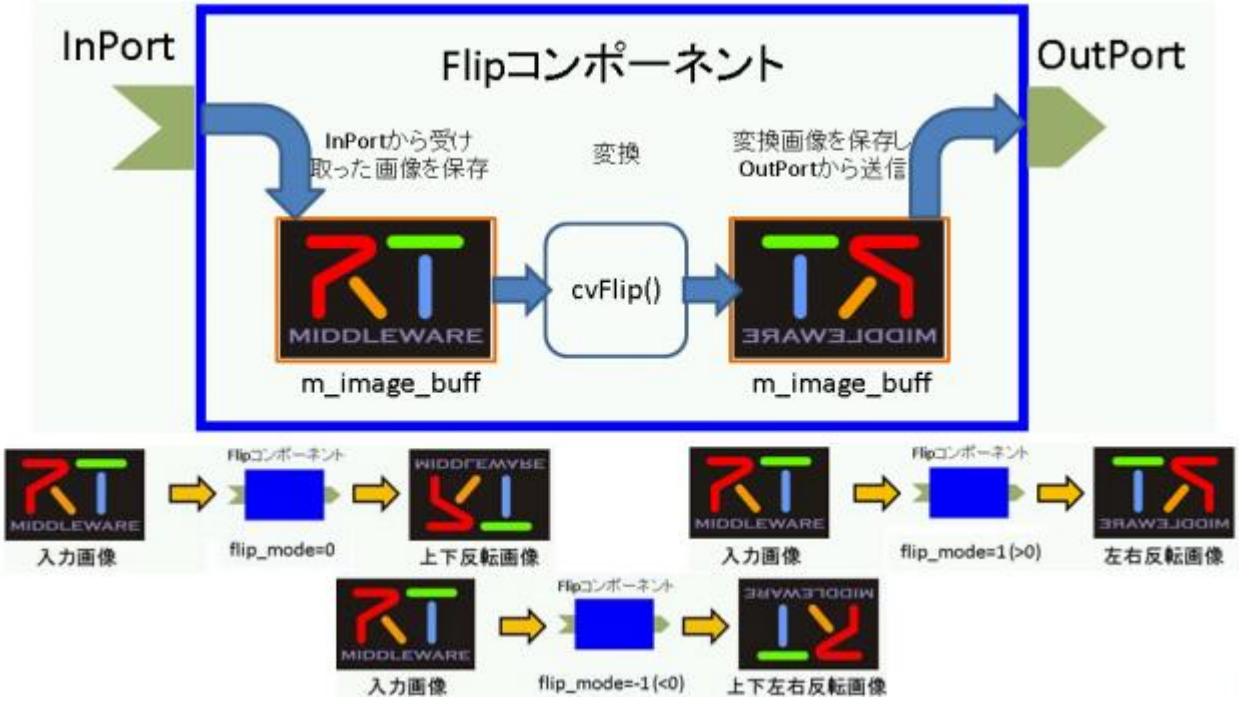
RTC Builderの起動





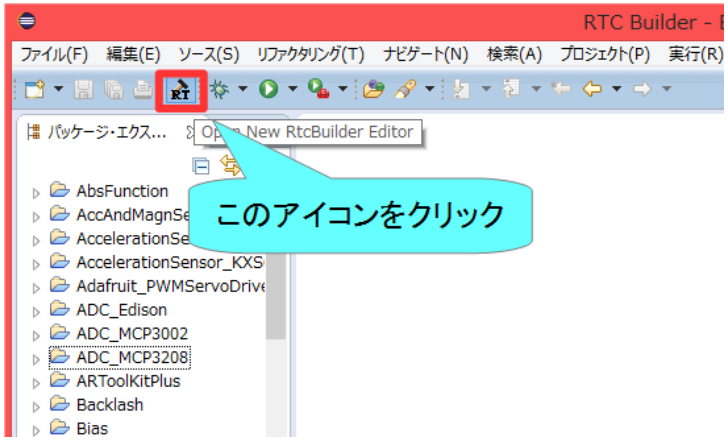
プロジェクト作成

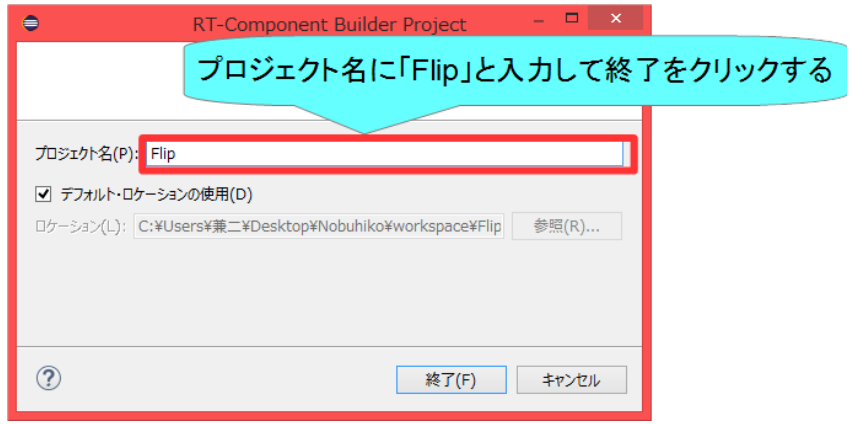
- 第3部プログラミング実習で使用するFlipコンポーネントのスケルトンコードを作成する。
  - 画像の反転を行うコンポーネント
    - InPortで受信した画像データを処理してOutPortから出力
    - コンフィギュレーションパラメータにより反転する方向を設定
    - RT System Editorにより他のRTCと接続、RTCをアクティブ化



第3部プログラミング実習の資料

プロジェクト作成

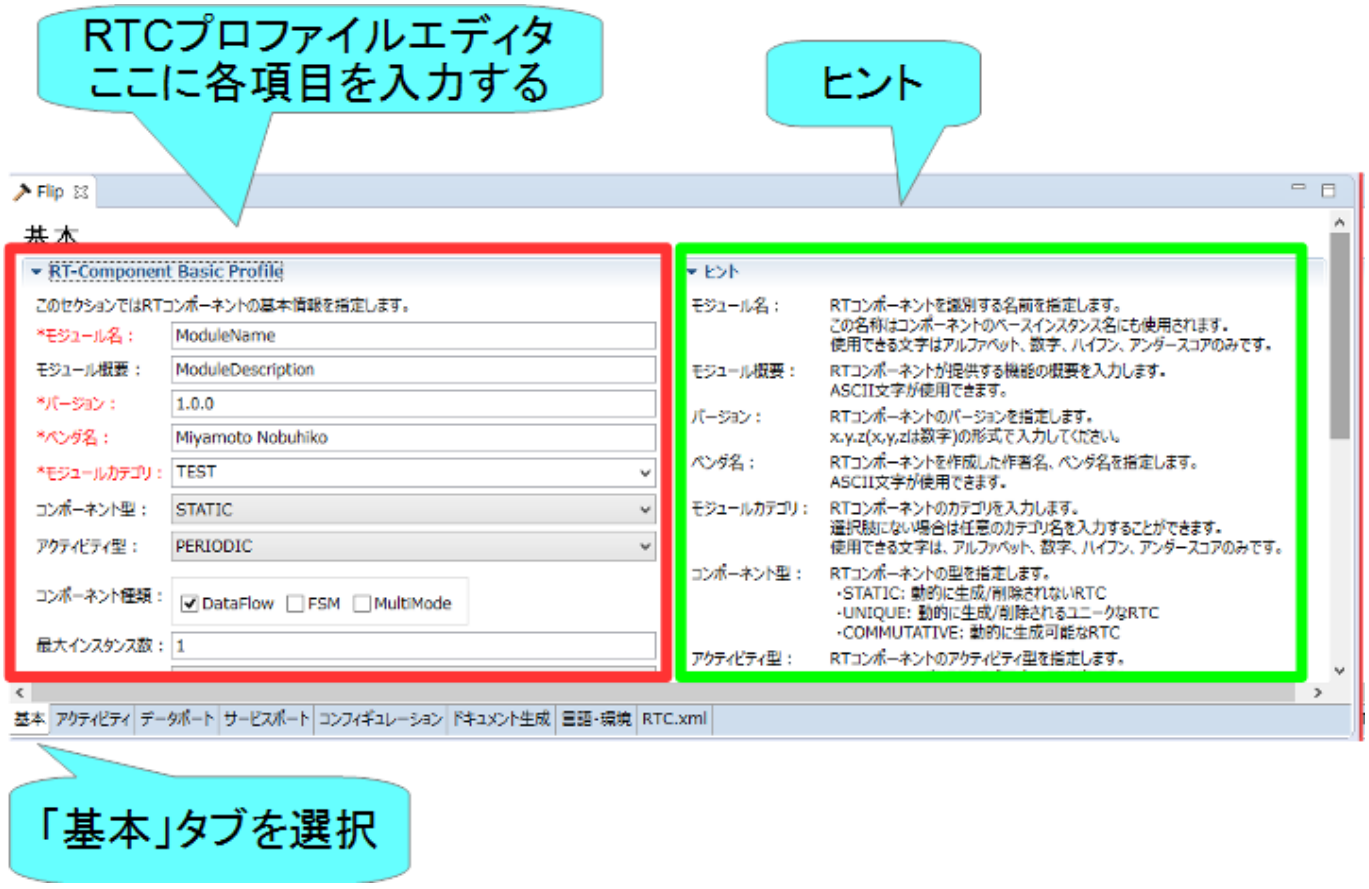




- Eclipse起動時にワークスペースに指定したディレクトリに「Flip」というフォルダが作成される
  - この時点では「RTC.xml」と「.project」のみが生成されている
- 以下の項目が設定する
  - 基本プロファイル
  - アクティビティ・プロファイル
  - データポート・プロファイル
  - サービスポート・プロファイル
  - コンフィギュレーション
  - ドキュメント
  - 言語環境
  - RTC.xml

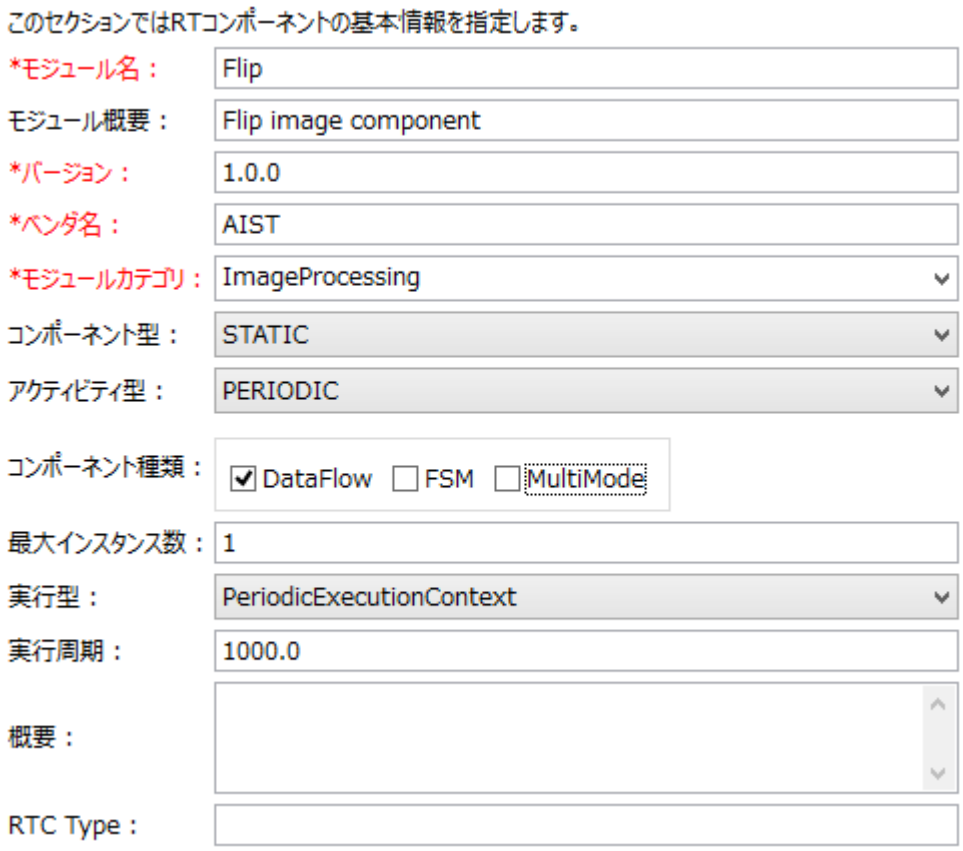
基本プロファイルの入力

- 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



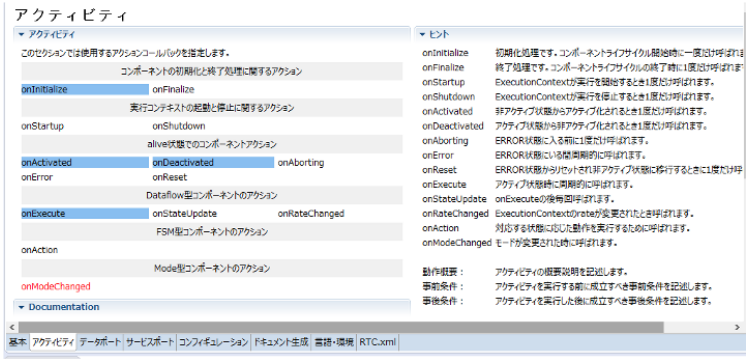
基本プロファイルの入力

- 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



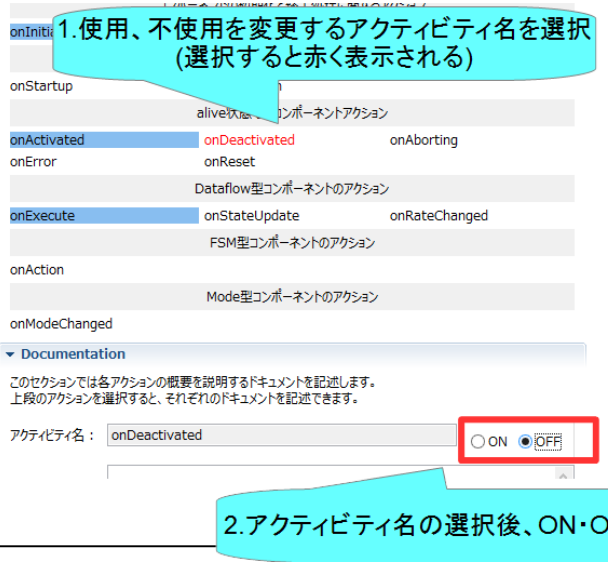
アクティビティの設定

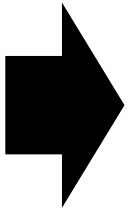
- 使用するアクティビティを設定する

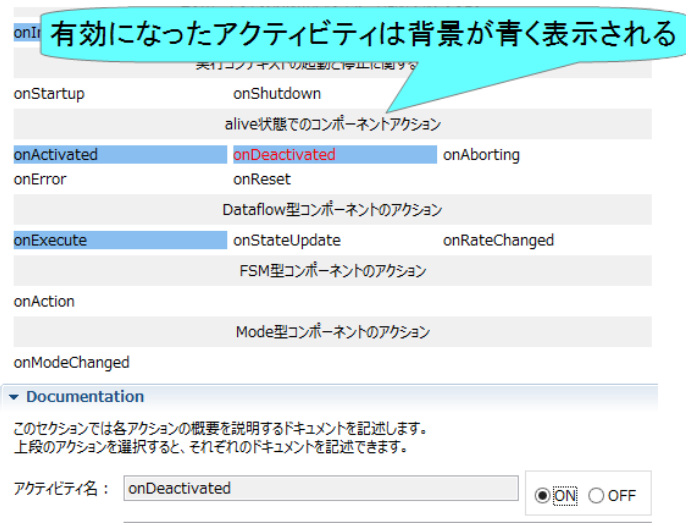


「アクティビティ」タブを選択

- 指定アクティビティを有効にする手順

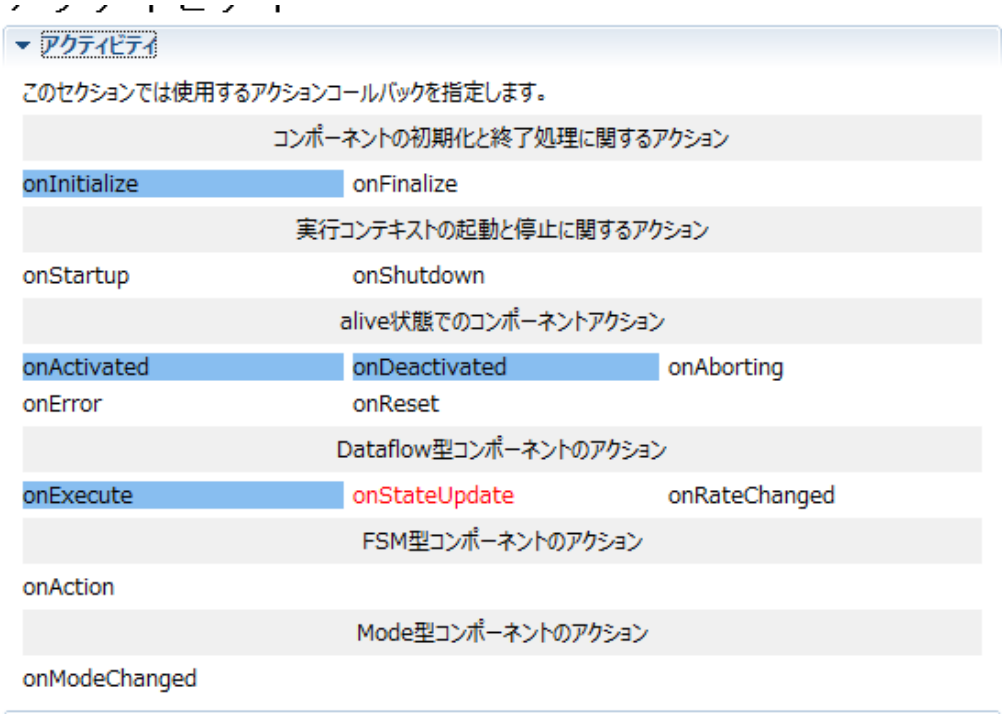






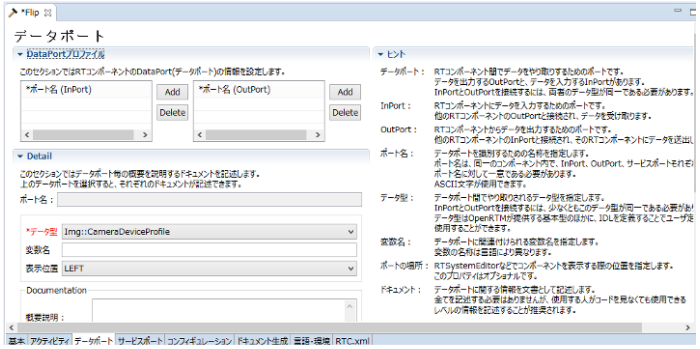
アクティビティの設定

- 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



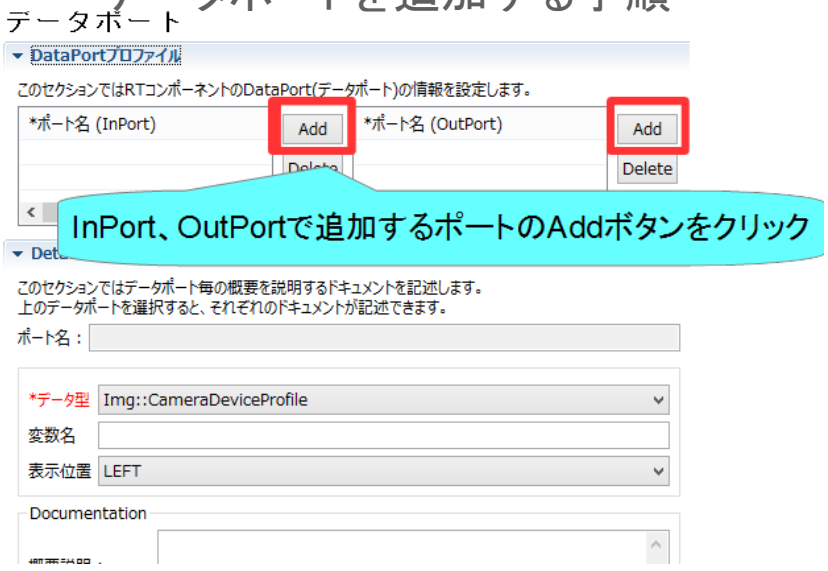
データポートの設定

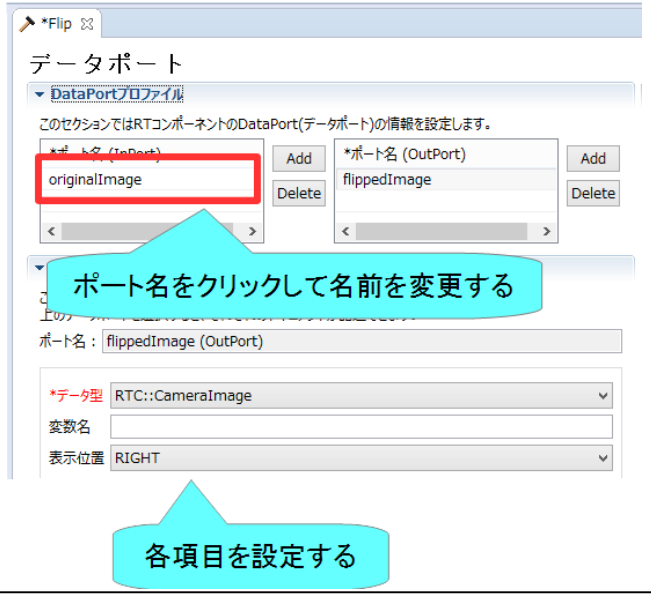
- InPort、OutPortの追加、設定を行う



「データポート」タブを選択

- データポートを追加する手順

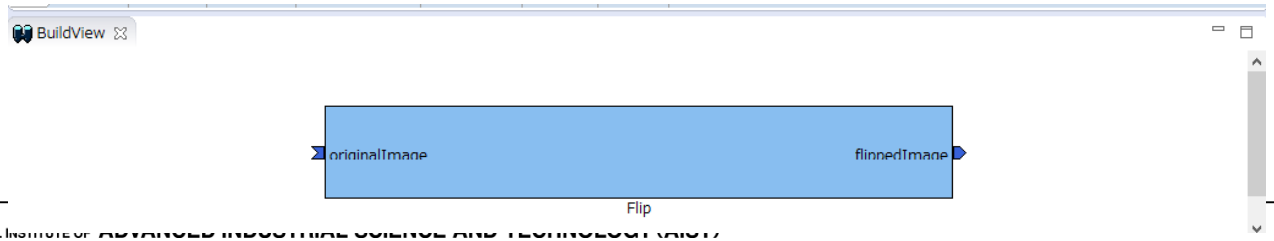




データポートの設定

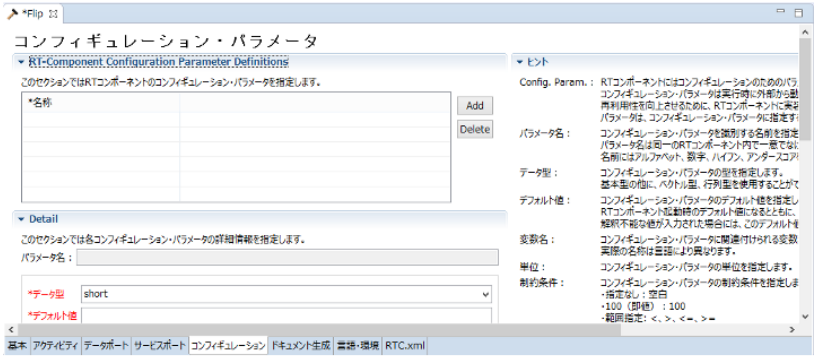
- 以下のInPortを設定する
  - originalImage
    - データ型：RTC::CameraImage
    - 他の項目は任意
- 以下のOutPortを設定する
  - flippedImage
    - データ型：RTC::CameraImage
    - 他の項目は任意

- ※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。



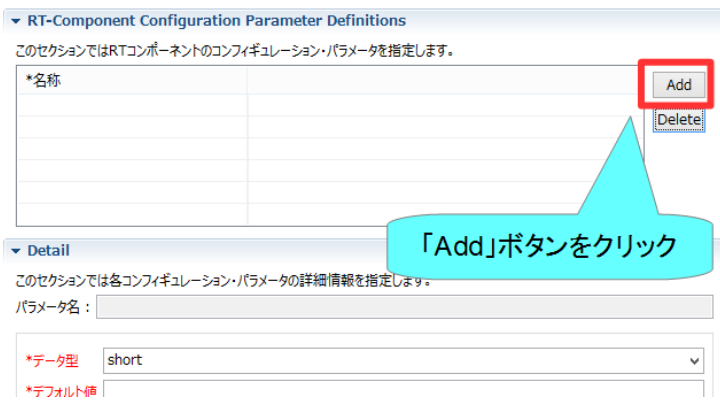
コンフィギュレーションの設定

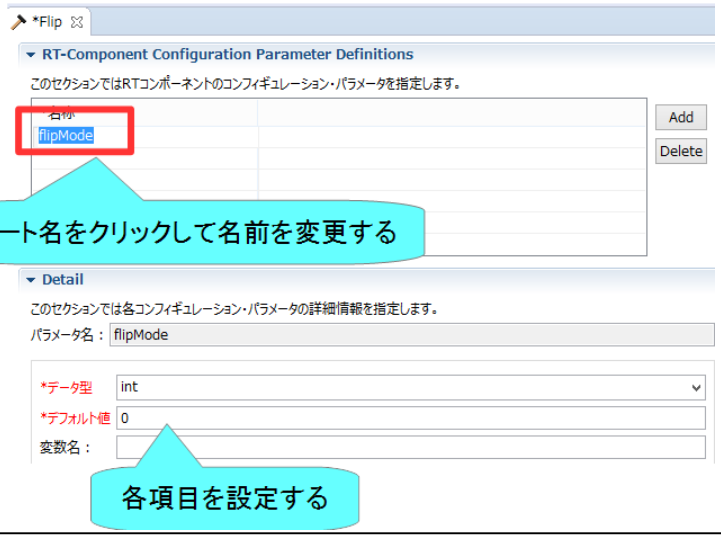
- コンフィギュレーションパラメータの追加、設定を行う



「コンフィギュレーション」タブを選択

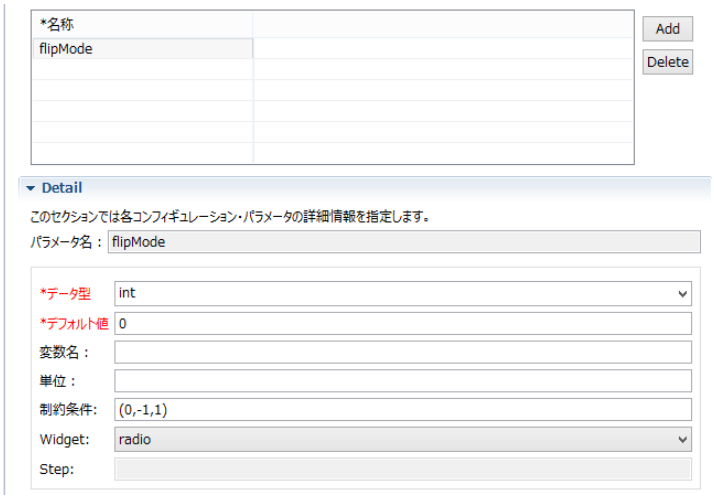
- コンフィギュレーションパラメータを追加する手順

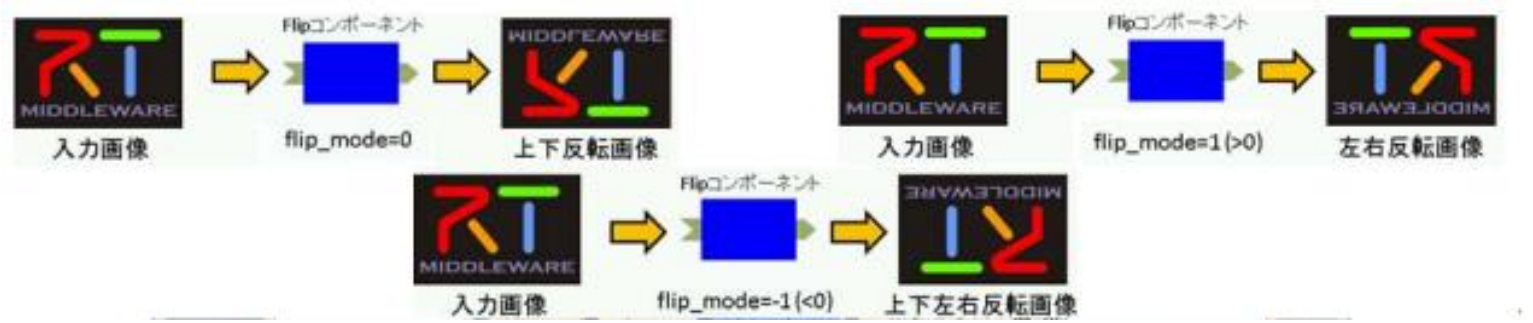




コンフィギュレーションの設定

- 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





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：slider
- Step：10

- 制約条件：0<=x<=100
- Widget：spin
- 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

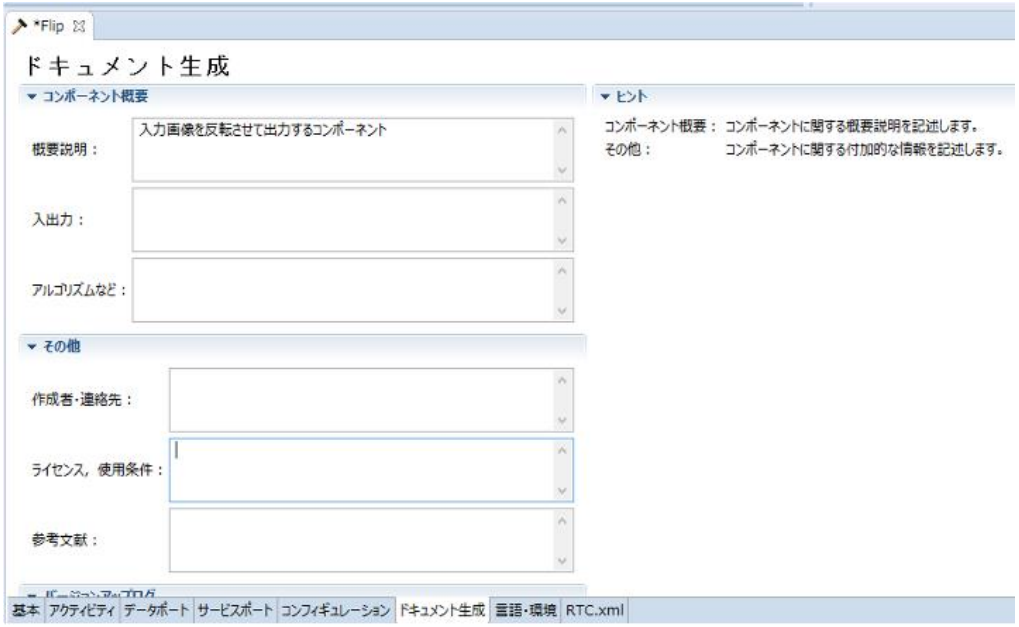
- 制約条件：(0,1,2,3)
- Widget：radio

- 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

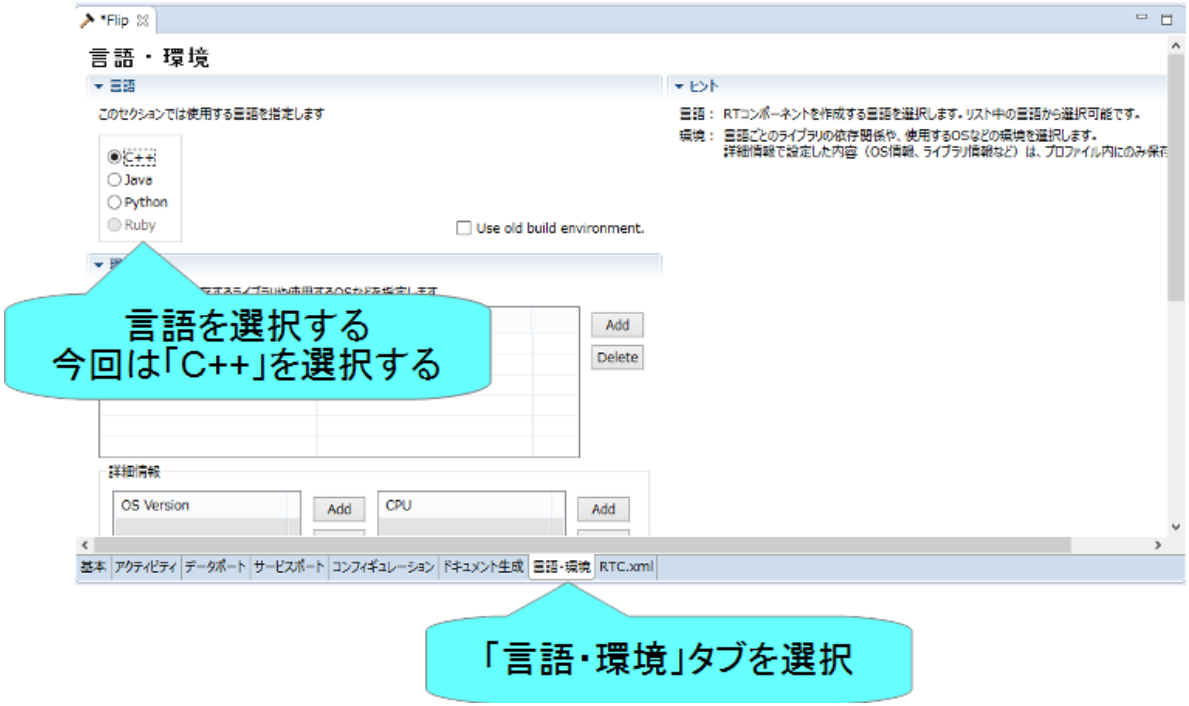


「ドキュメント生成」タブを選択

- 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

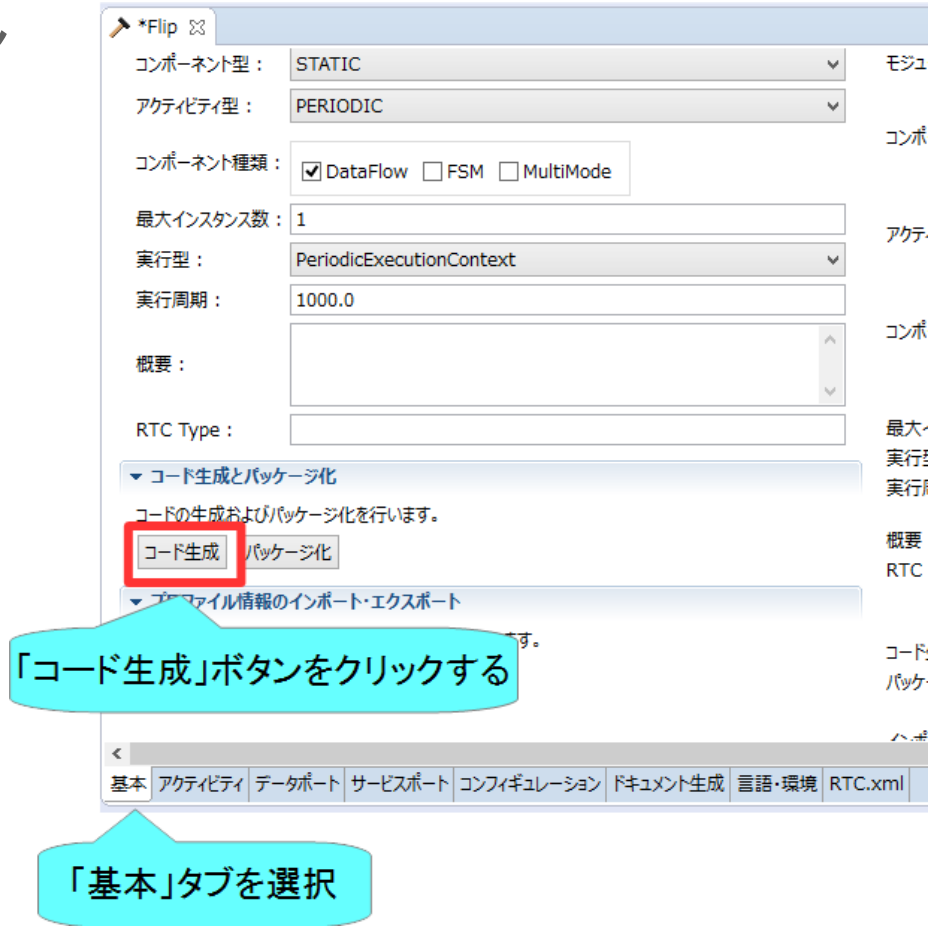
言語の設定

- 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

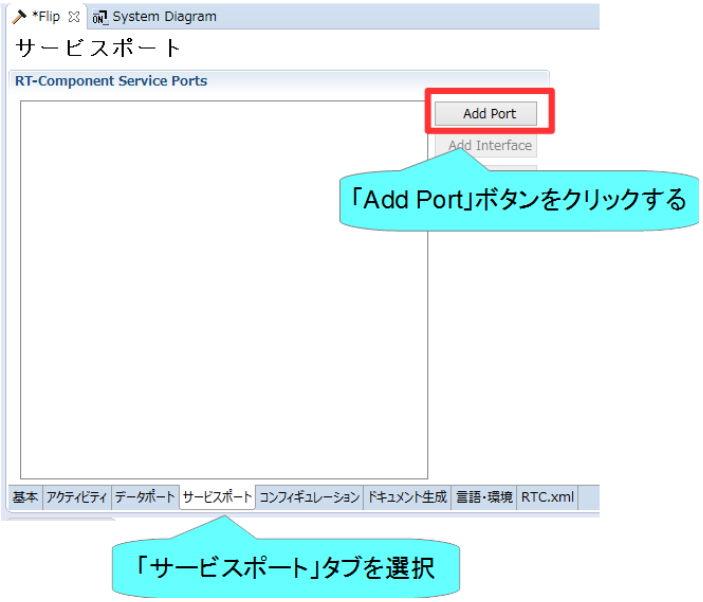
- 基本タブからコード生成ボタンを押すことでスケルトンコードが生成される
  - ソースコード
    - C++ソースファイル(.cpp)
    - ヘッダーファイル(.h)
      - このソースコードに画像を反転させる処理を記述する
  - CMakeの設定ファイル
    - CMakeLists.txt
  - 以下略

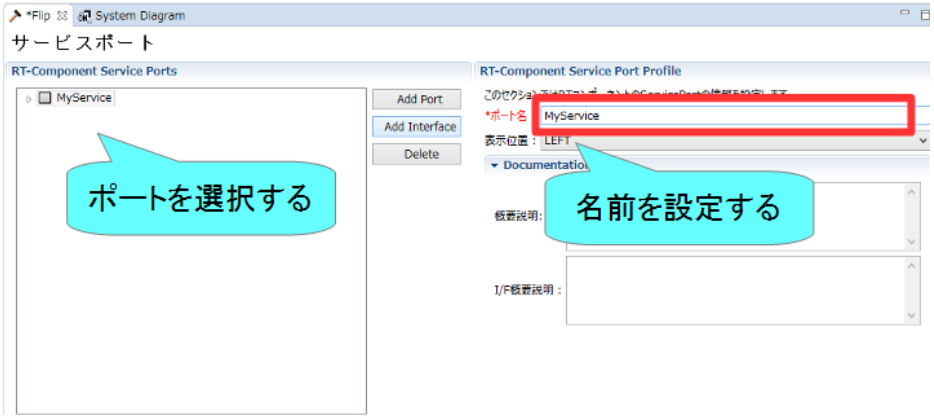


RTC Builder
補足

サービスポートの設定

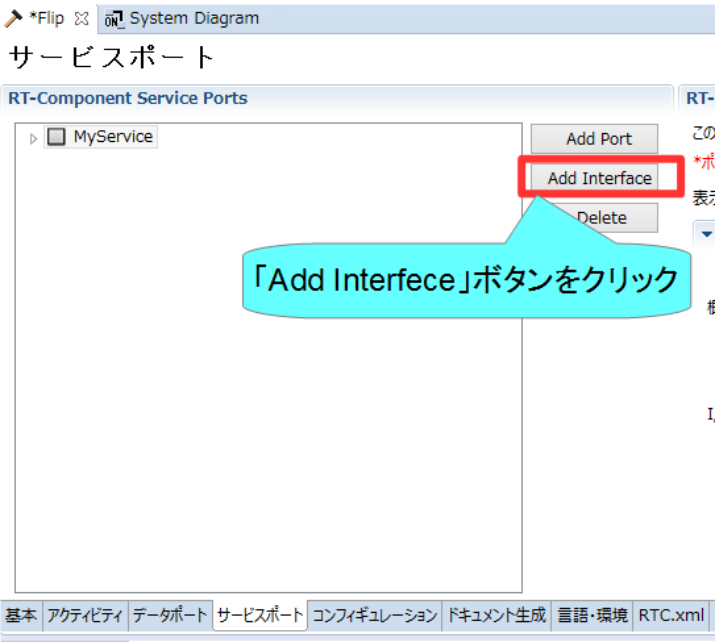
- サービスポートの追加、インターフェースの追加、設定を行う

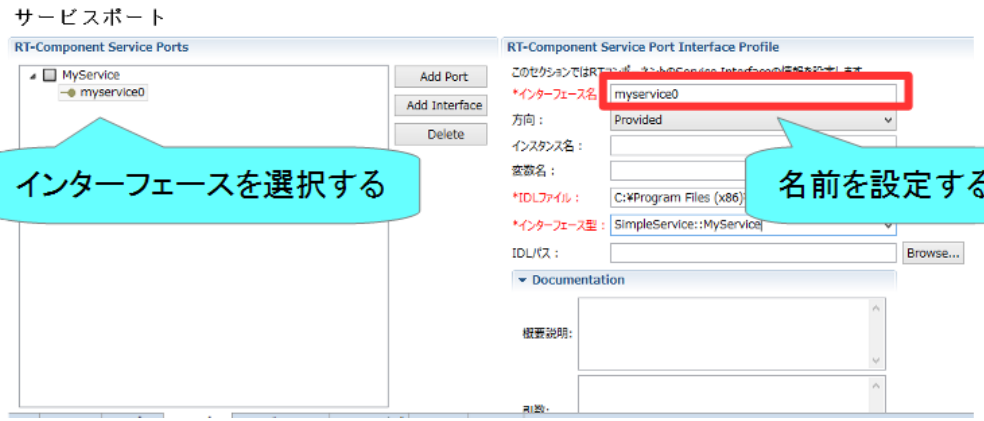




サービスポートの設定

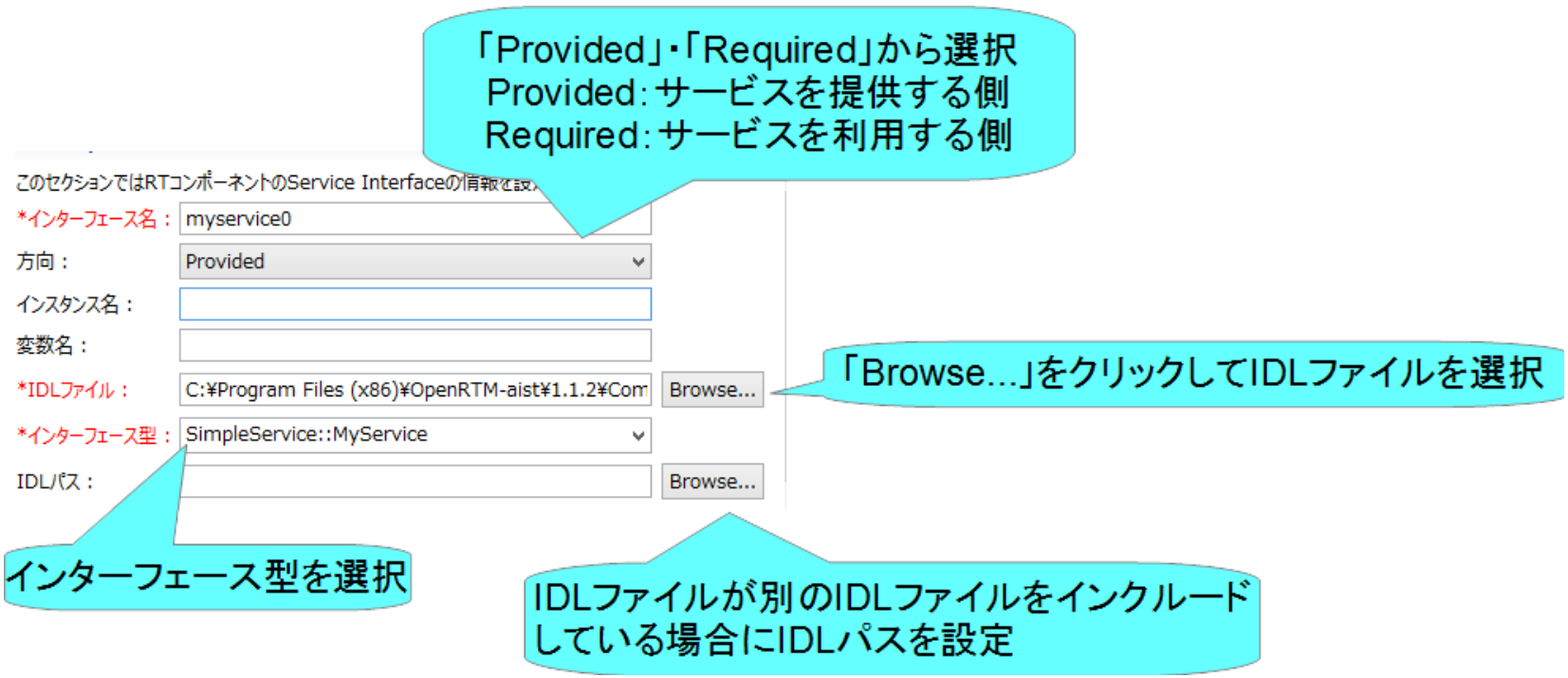
- インターフェースを追加する





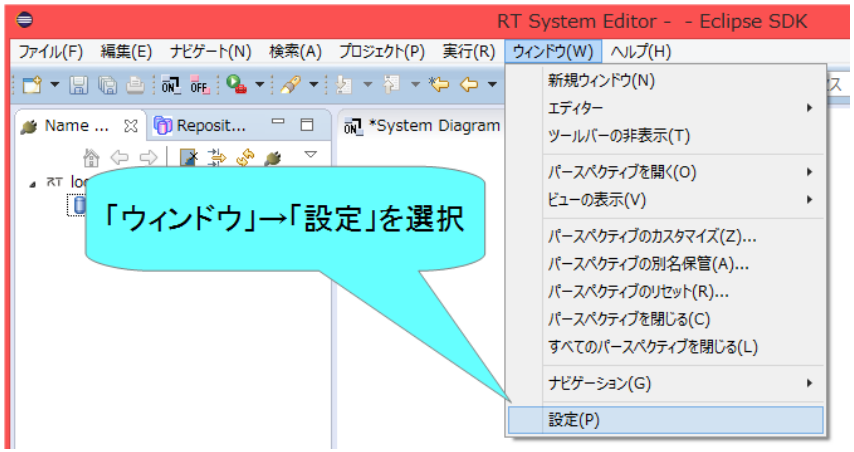
サービスポートの設定

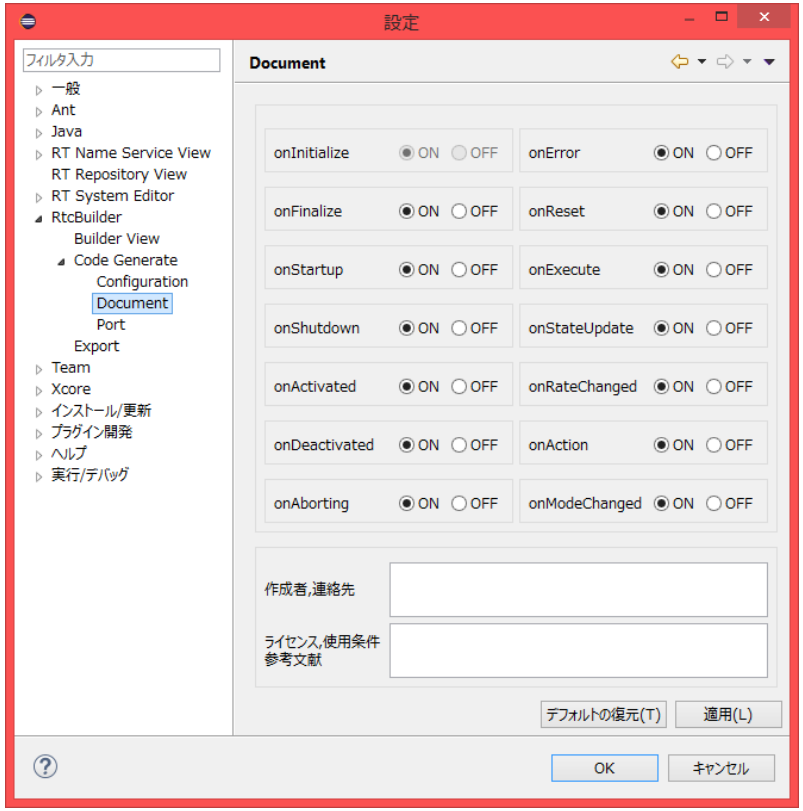
- インターフェースの設定を行う



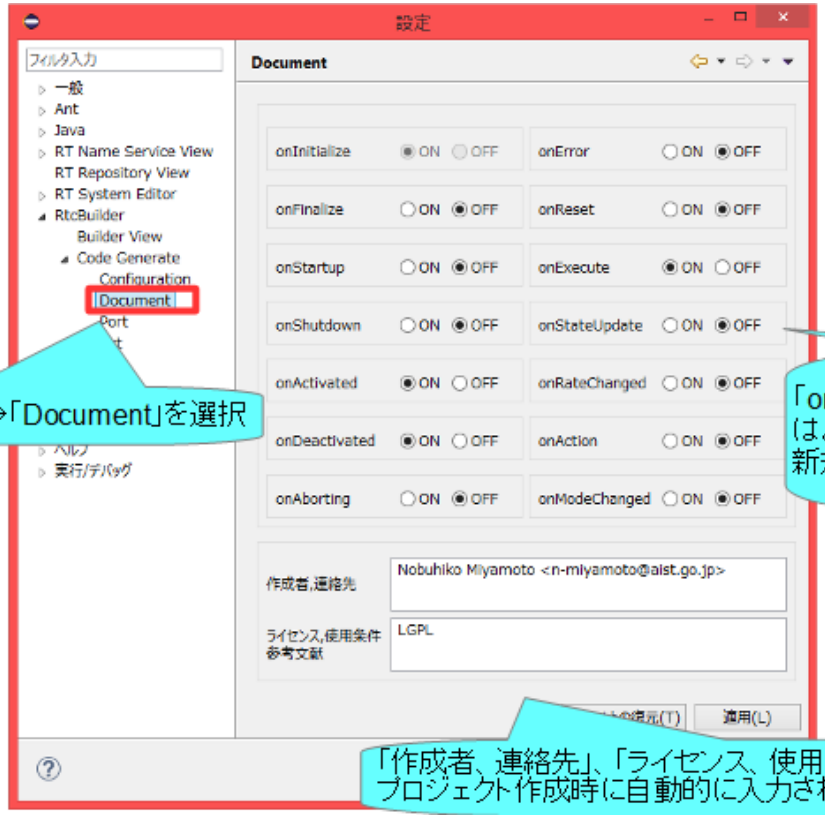
- コード生成後、Pythonの場合はidlcompile.bat(idlcompile.sh)を起動する

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。

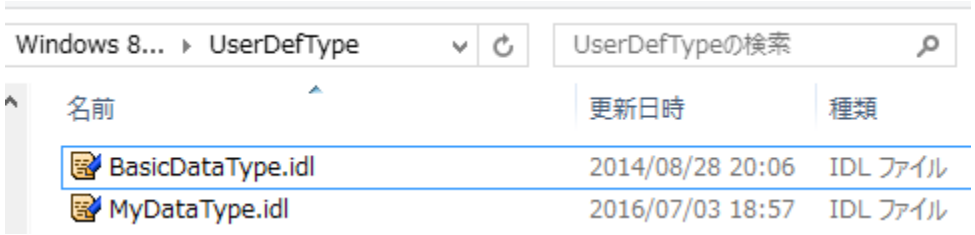
「作成者、連絡先」、「ライセンス、使用条件、参考文献」を入力しておくとプロジェクト作成時に自動的に入力されるので便利。

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

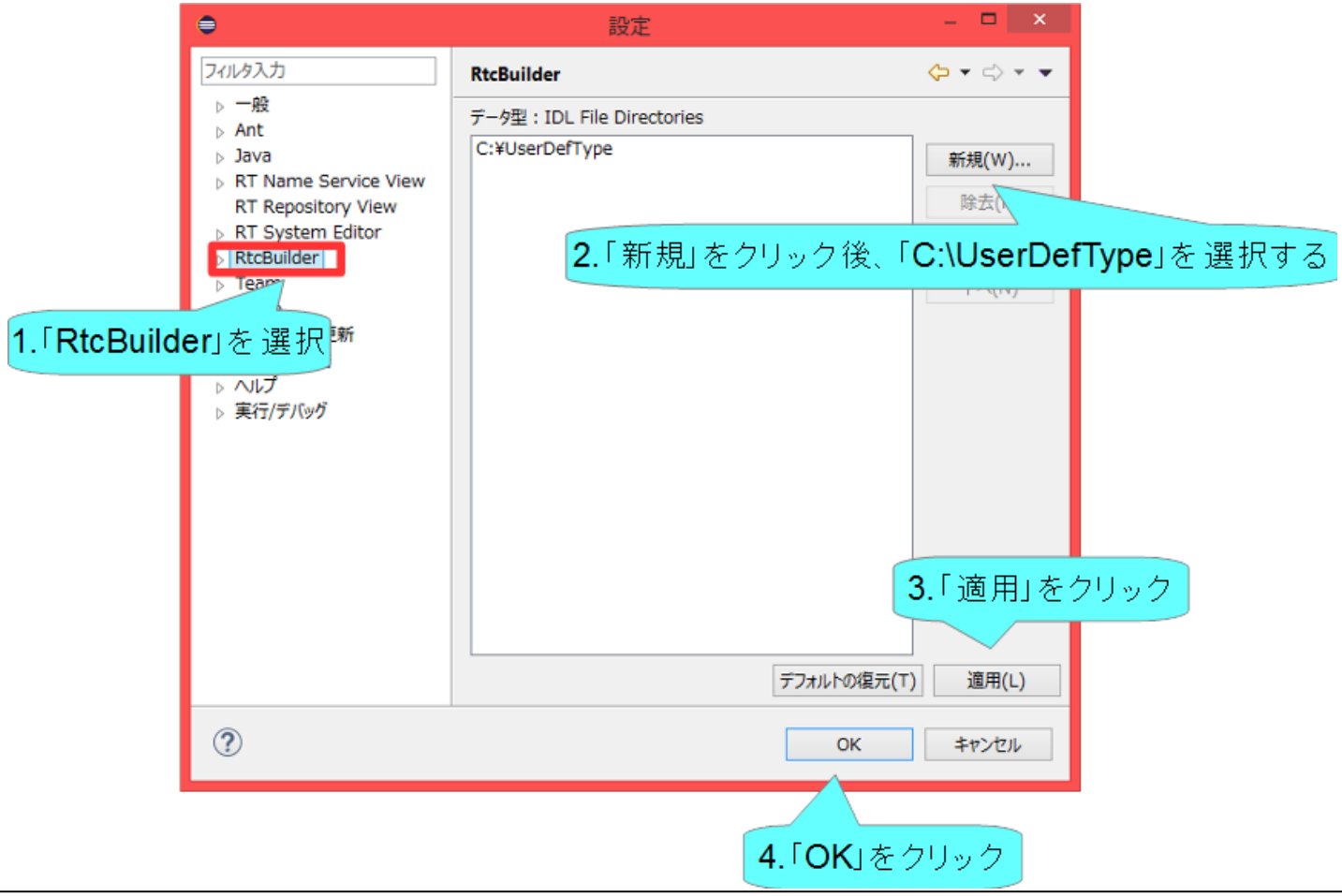
```
1  // @file MyDataType.idl
2  #include "BasicDataType.idl"
3
4  struct MyData
5  {
6    RTC::Time tm;
7    short shortVariable;
8    long longVariable;
9    sequence<double> data;
10 };
```

    - 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