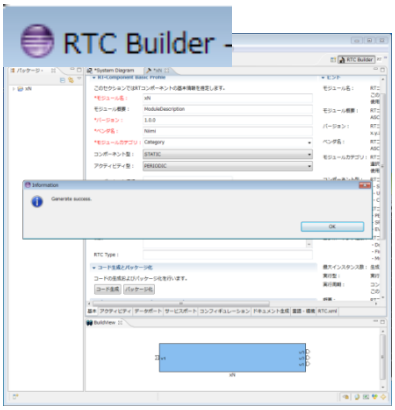
StarTno_00 説明書 02

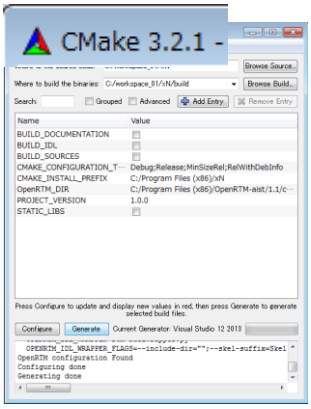
RTC の作り方２

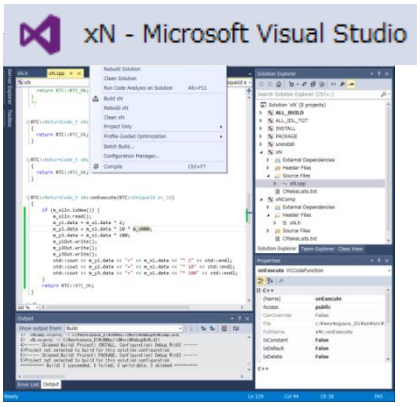
(Win10, Visual Studio2015)

作成手順

RTC ビルダーでコンポーネントを構築し，CMake で必要なファイルを作り，VsualC++でコードを書いてビルドする． RT システムエディタで動作確認を行う．

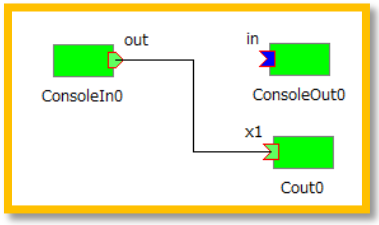


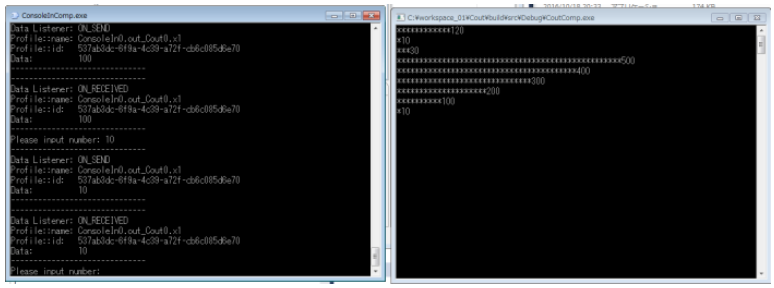




Cout(001～022)

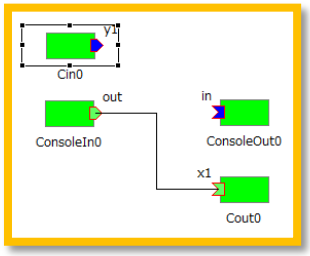
コンソールアウトに相当する RTC．簡易棒グラフ機能付き．

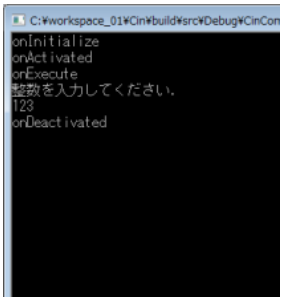


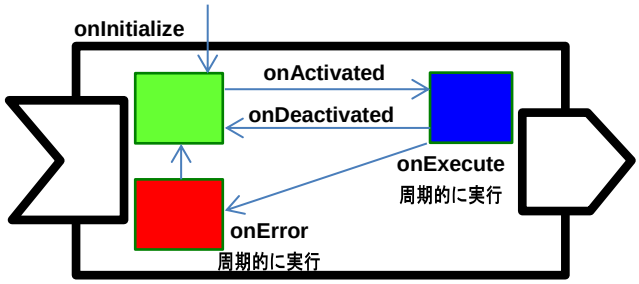


Cin(023～027)

コンソールインに相当する RTC．状態の遷移を表示する．

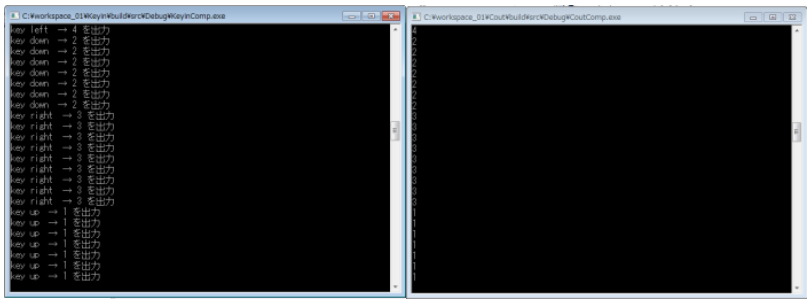






Keyin(028)

1, 2, a, ↑, ↓, ←, →のキーを押すと, 1, 2, 1, もと＋10, もと－10, もと＋1, もと－1 を出力する RTC. .

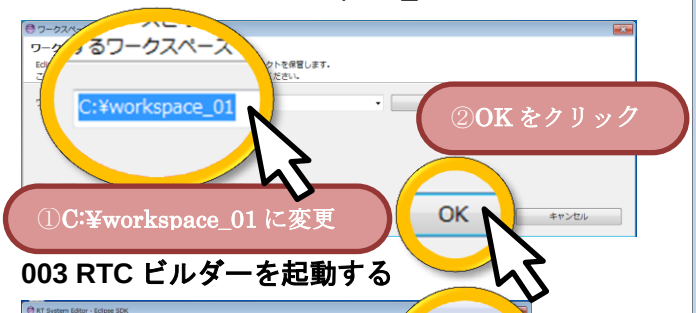


001　OpenRTP1.1.0 を起動する　①

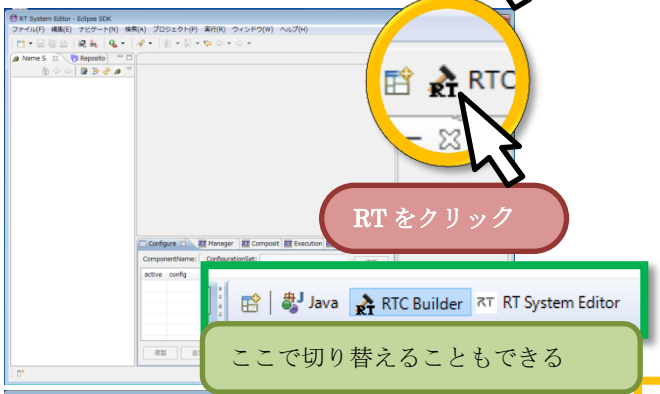
すべてのアプリから

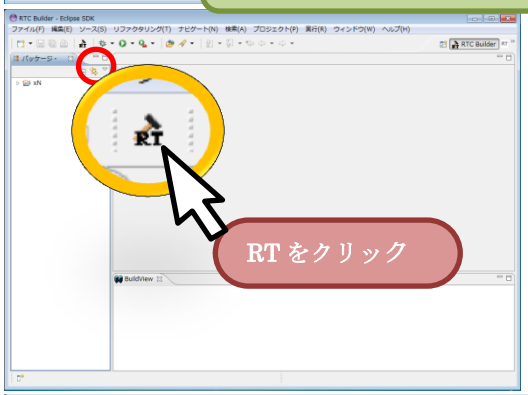
002　ワークスペースを設定する

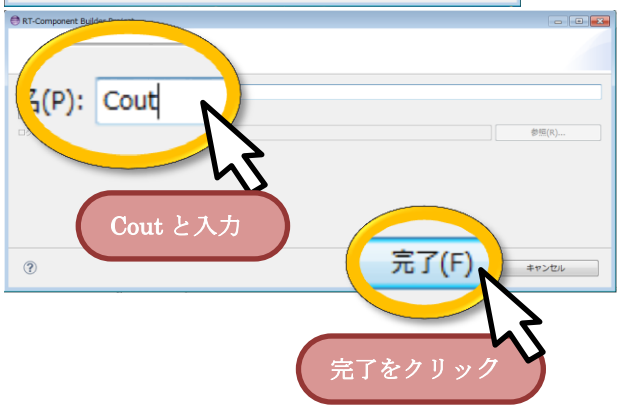
ここでは C:¥workspace_01 を設定した.



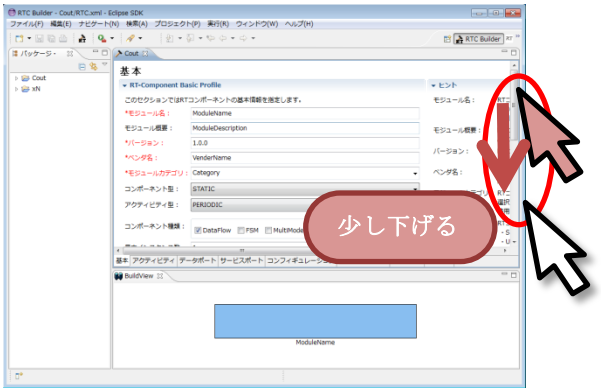
003 RTC ビルダーを起動する



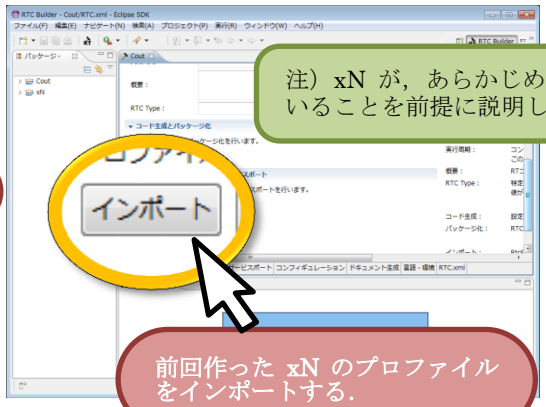


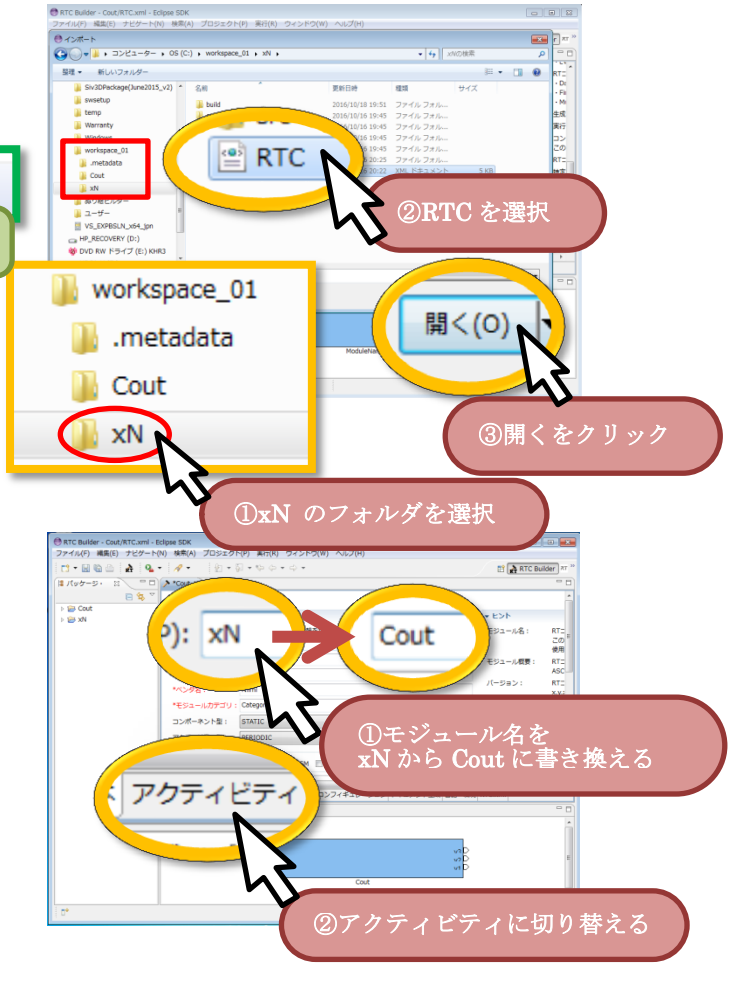


005 RTC ビルダーの画面

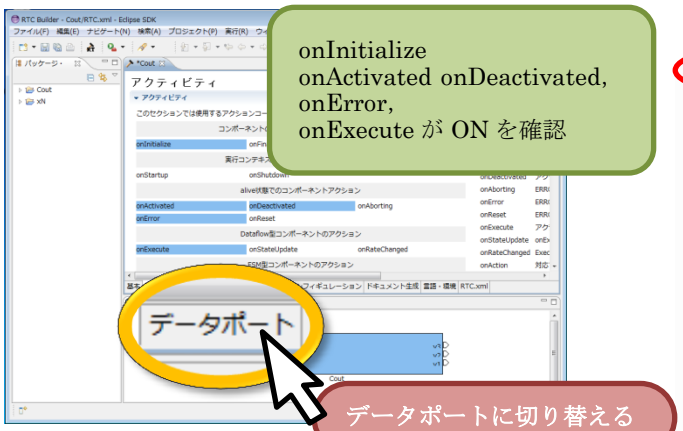


006 プロファイルをインポートする

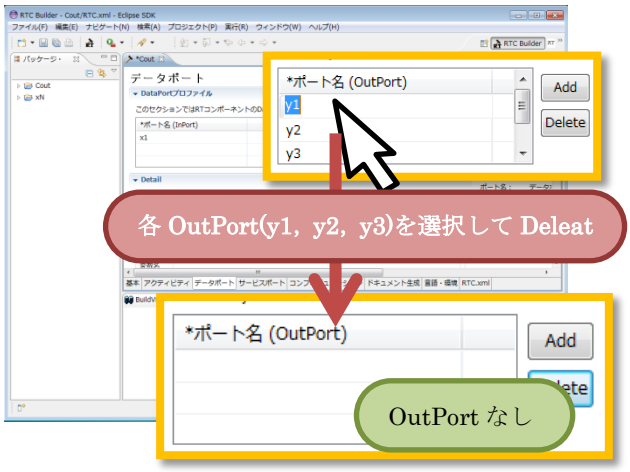


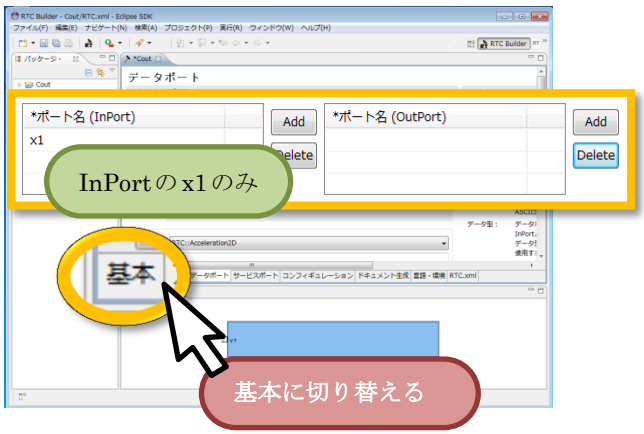


007 アクティビティを確認する．(5 か所)

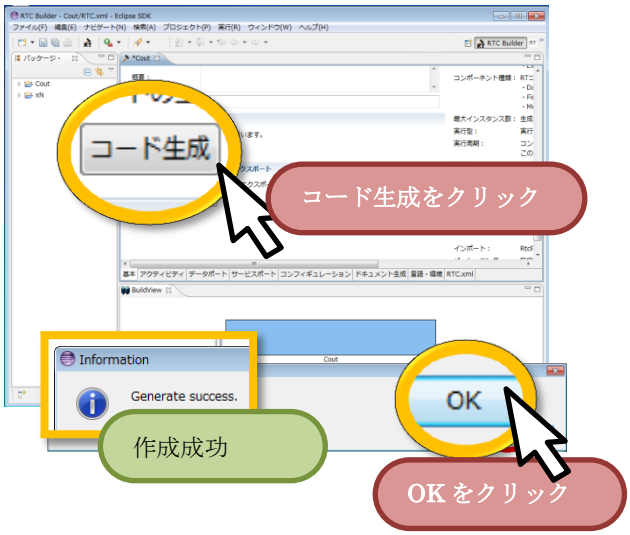


008 データポートは，x1 のみ．y1y2y3 削除

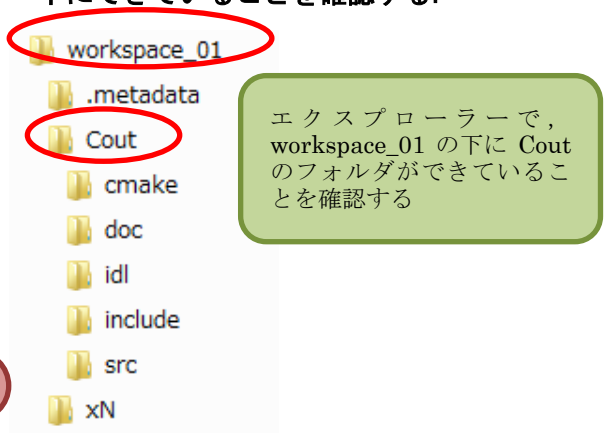




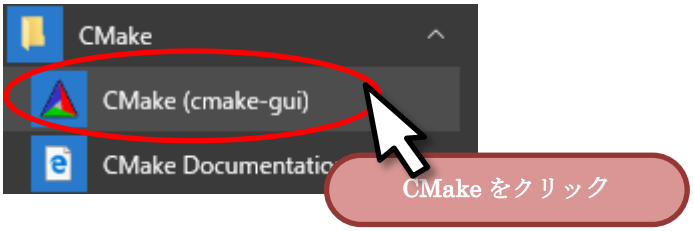
009 コード生成

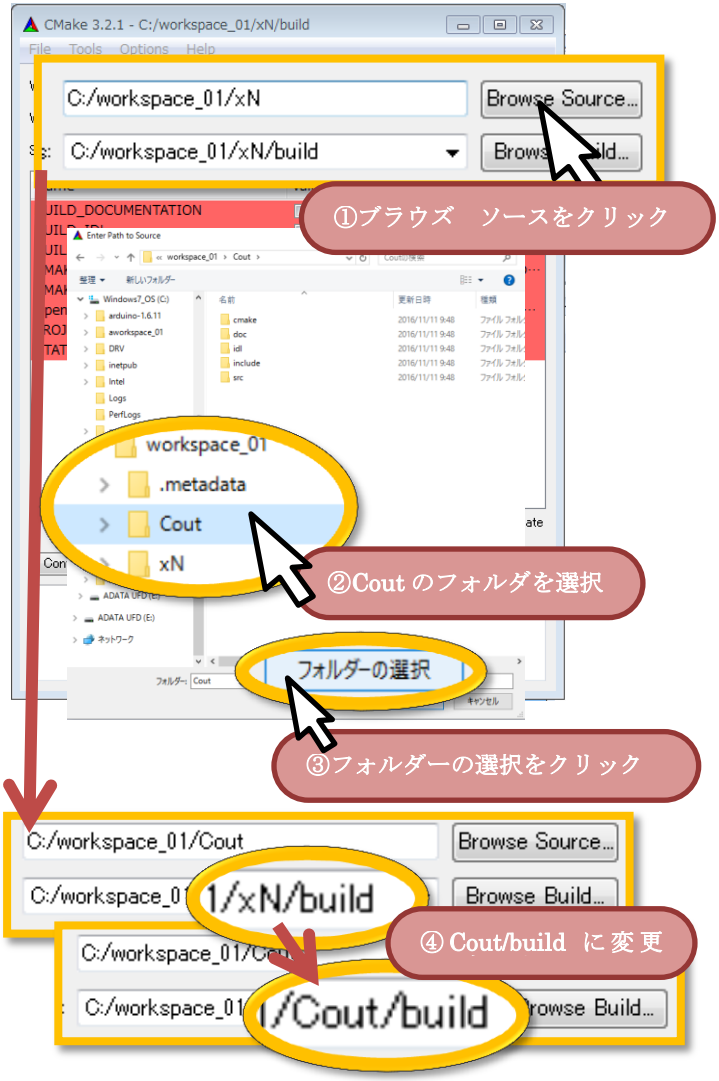


010 Cout というフォルダーが workspace_01 の下にできていることを確認する．

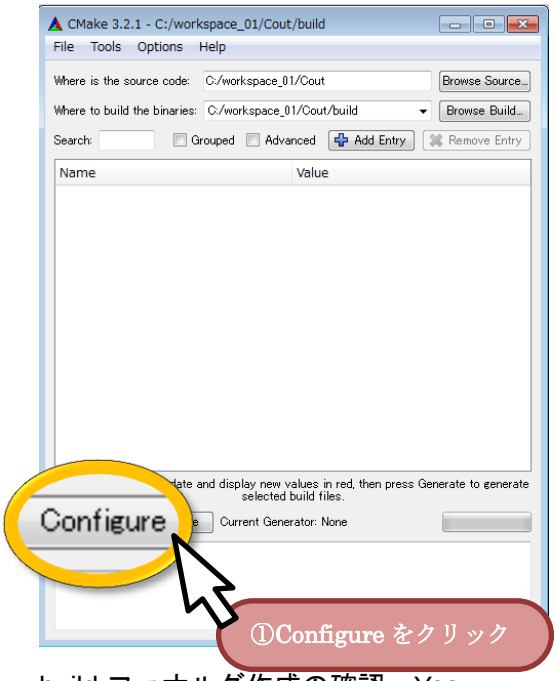


011 CMake の設定

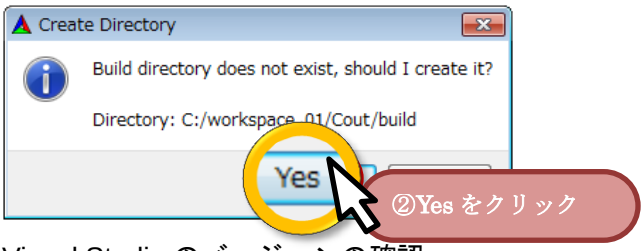




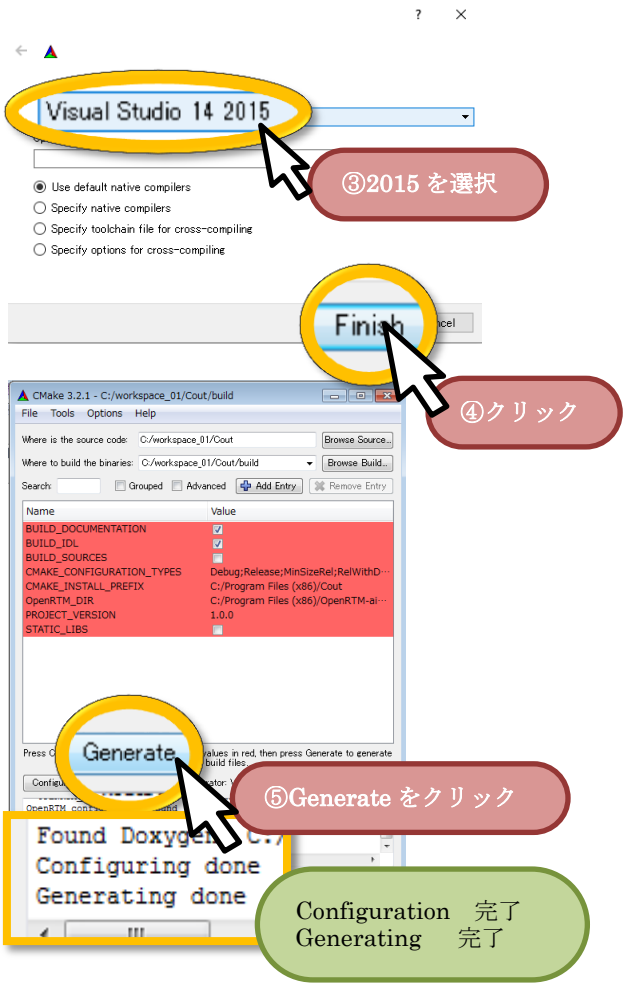
012 Cmake： Configure，Generate



build ファオルダ作成の確認． Yes.

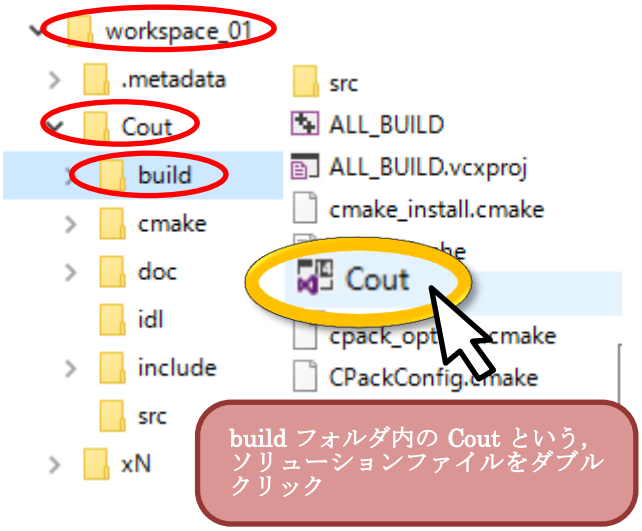


Visual Studio のバージョンの確認



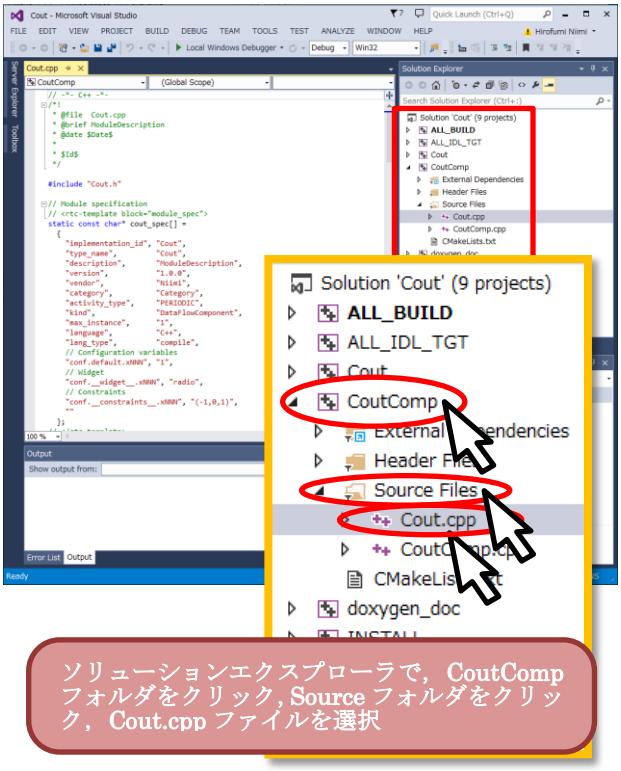
013 エクスプローラーで Cout の下の build を開く．

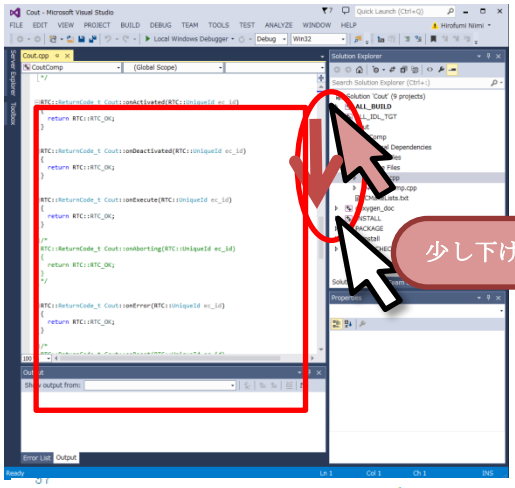
Cout のソリューションファイルをダブルクリック



Visual　Studio の起動中の画面

Cout.cpp ファイルを開く



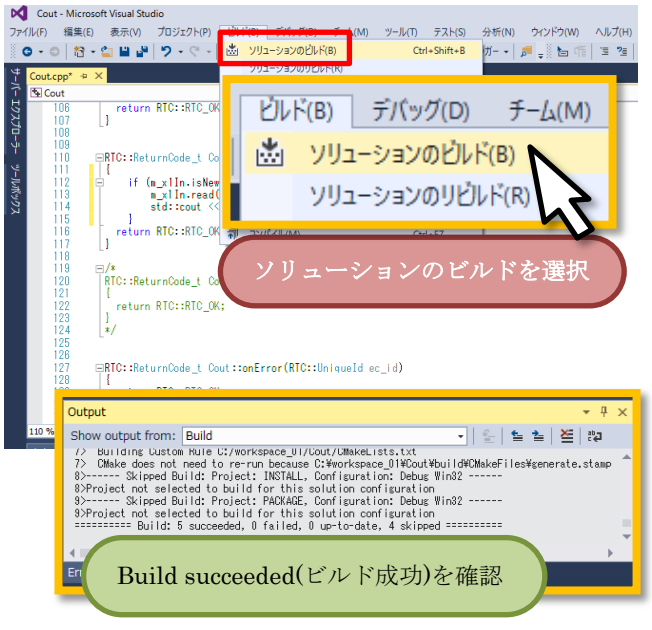



```
98      RTC::ReturnCode_t Cout::onActivated(RTC::UniqueId ec_id)
99      {
100       return RTC::RTC_OK;
101     }
102
103
104     RTC::ReturnCode_t Cout::onDeactivated(RTC::UniqueId ec_id)
105     {
106       return RTC::RTC_OK;
107     }
108
109
110     RTC::ReturnCode_t Cout::onExecute(RTC::UniqueId ec_id)
111     {
112       return RTC::RTC_OK;
113     }
114
115     /*
116     RTC::ReturnCode_t Cout::onAborting(RTC::UniqueId ec_id)
117     {
118       return RTC::RTC_OK;
119     }
120     */
121
122
123     RTC::ReturnCode_t Cout::onError(RTC::UniqueId ec_id)
124     {
125       return RTC::RTC_OK;
126     }
127
```

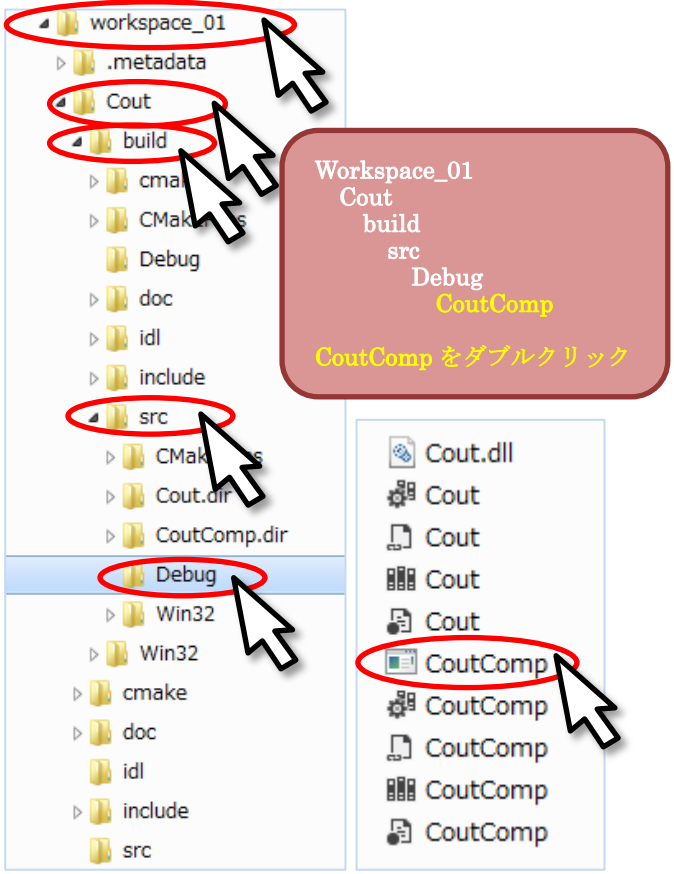
ここにコードを書く

```
RTC::ReturnCode_t Cout::onExecute(RTC::UniqueId ec_id)
{
    if (m_x1In.isNew()) {
        m_x1In.read();
        std::cout << m_x1.data  << std::endl;
    }
  return RTC::RTC_OK;
}
```

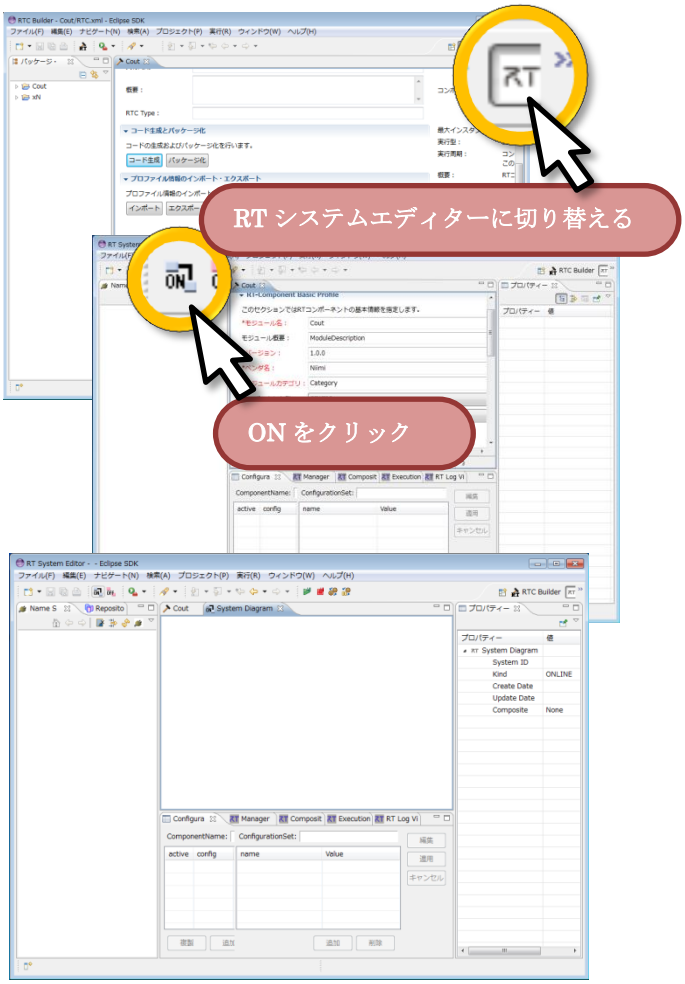
注意：エックス_いち　エンド_エル



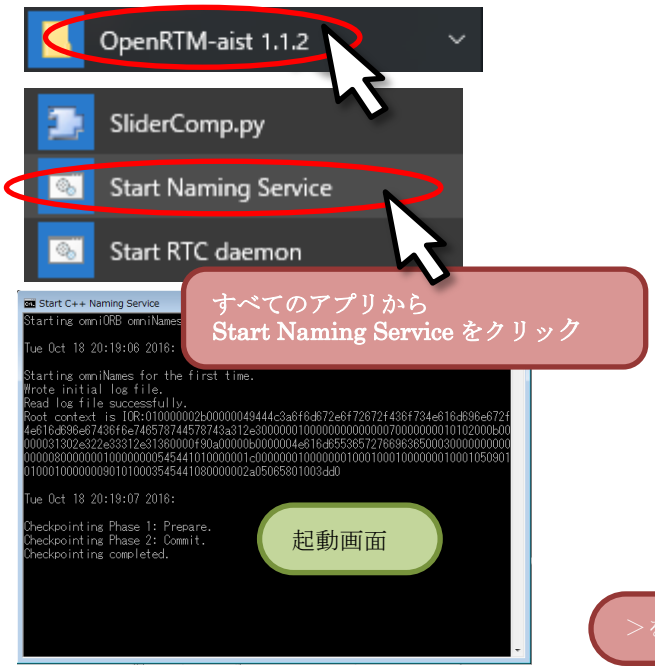
014 CoutComp の起動



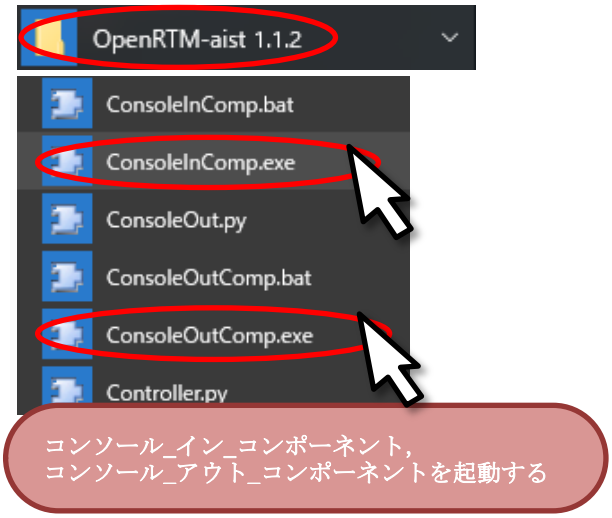
015 RT システムエディターの起動



016 NamingSeavice の起動

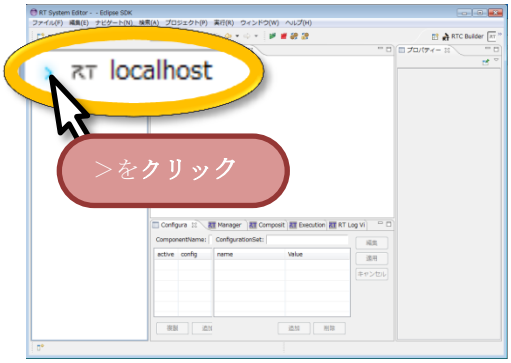


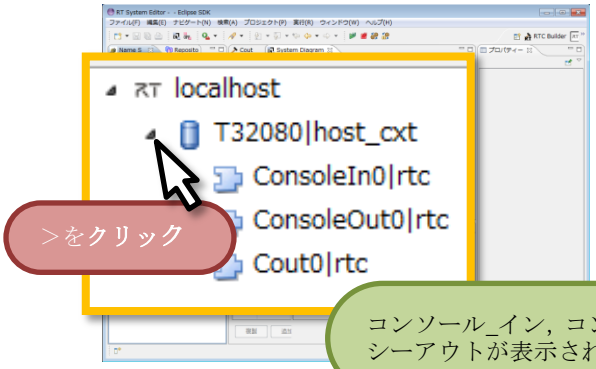
017 ConsoleInComp,
ConsoleOutComp の起動



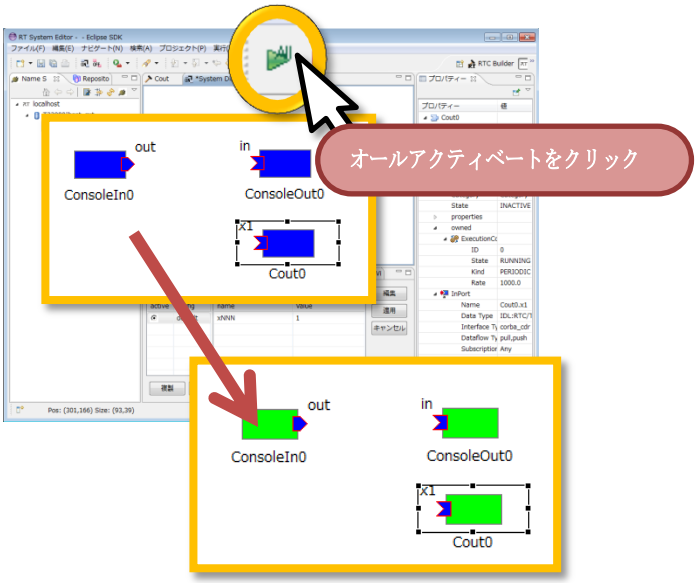
018 ネームサービスの追加

019 RTC の確認

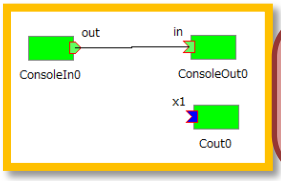




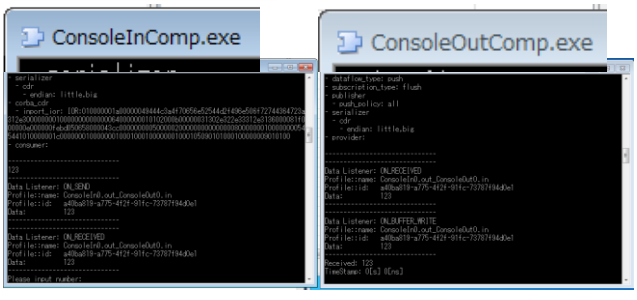
020 RTC の配置

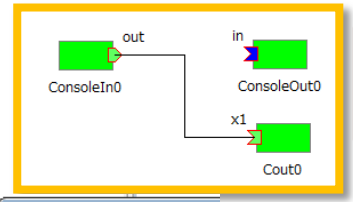


021 RTC の接続と動作確認

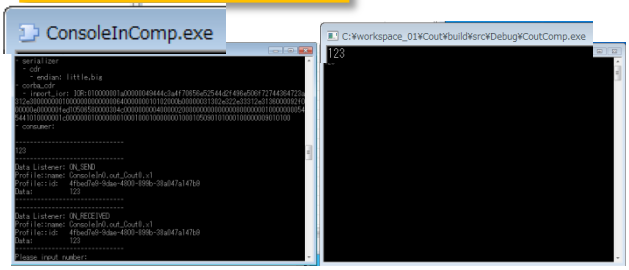


ConsoleIn,ConsoleOut を接続し，
動作を確認する．
123 を入力すると，
123 が出力される．

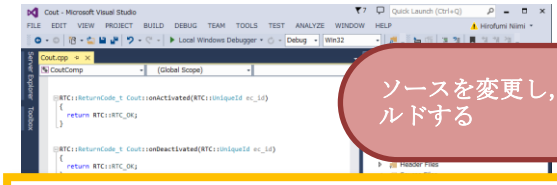




前回の接続線を削除後，ConsoleIn,Cout を接続．123 を入力すると，123 が出力される．

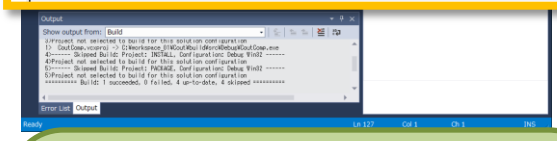


022 Cout のソースの更新

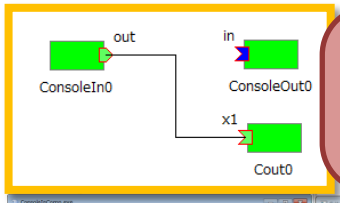


ソースを変更し，ビルドする

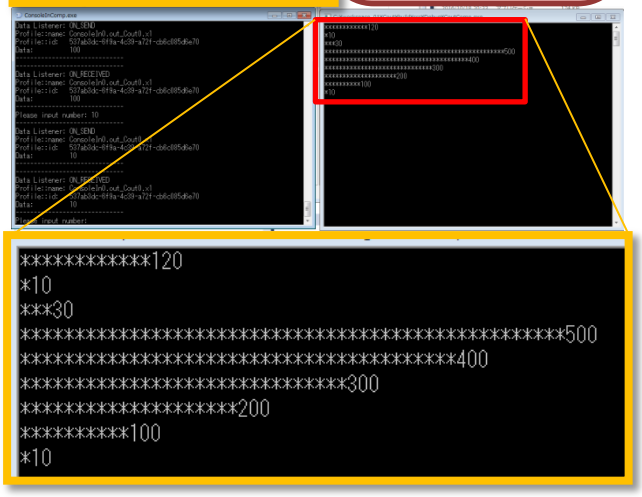
```
RTC::ReturnCode_t Cout::onExecute(RTC::UniqueId ec_id)
{
    if (m_x1In.isNew()) {
        m_x1In.read();
        for (int i = 0; i < m_x1.data / 10;i++)printf("*");
        std::cout << m_x1.data << std::endl;
    }
  return RTC::RTC_OK;
}
```



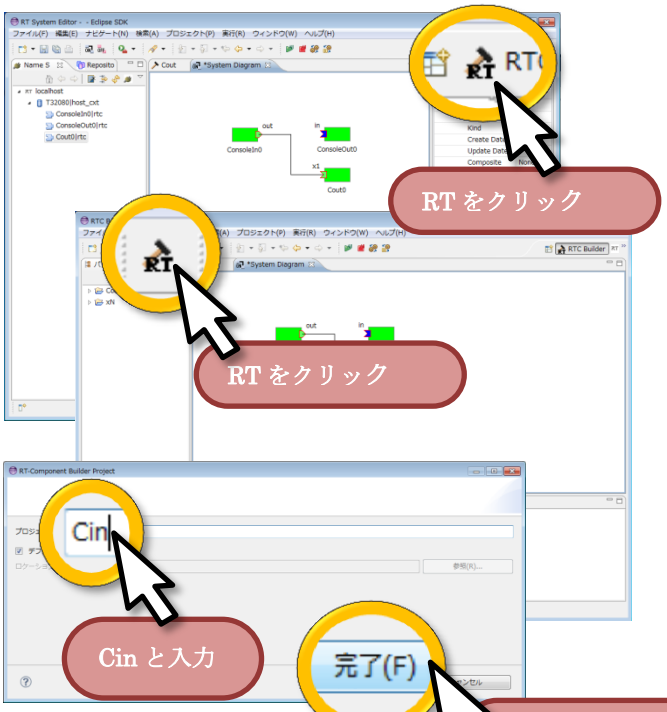
Cout を Build する．CoutComp が起動状態では，Build は失敗する．失敗した場合，CoutComp を閉じて，再び Build する．Cout を起動，接続し，動作確認を行う．10 ごとに，「*」を表示する棒グラフ．



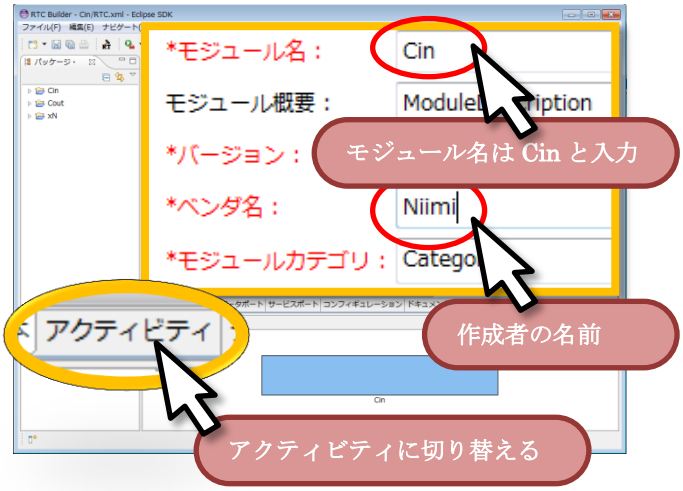
数字を入力すると 10 で「*」が一つ表示される．棒グラフの様になる．

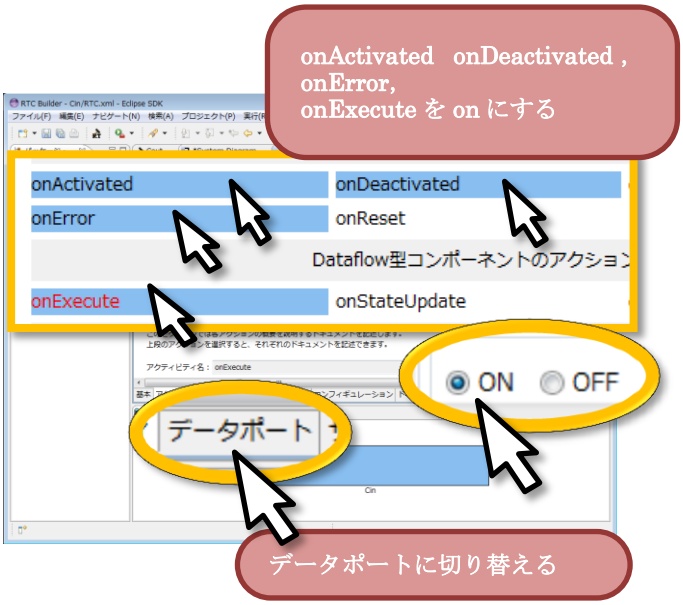


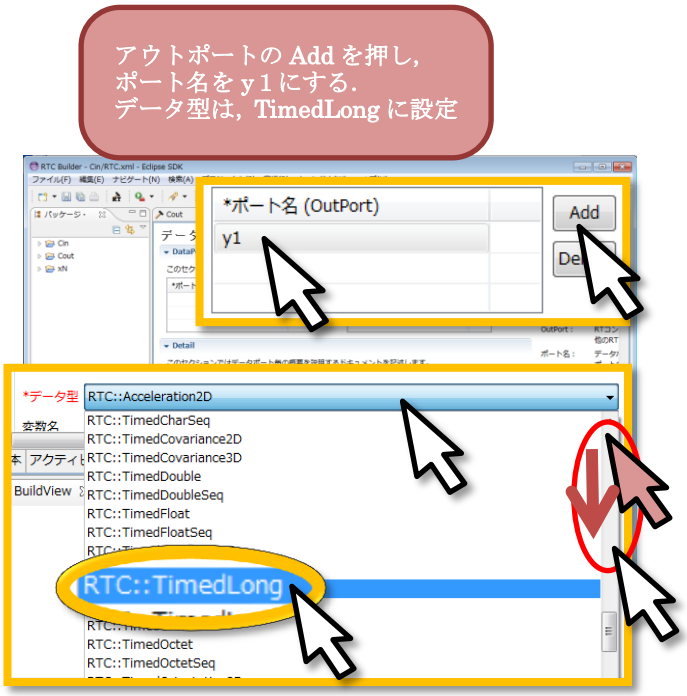
023 Cin コンポーネントの作成準備

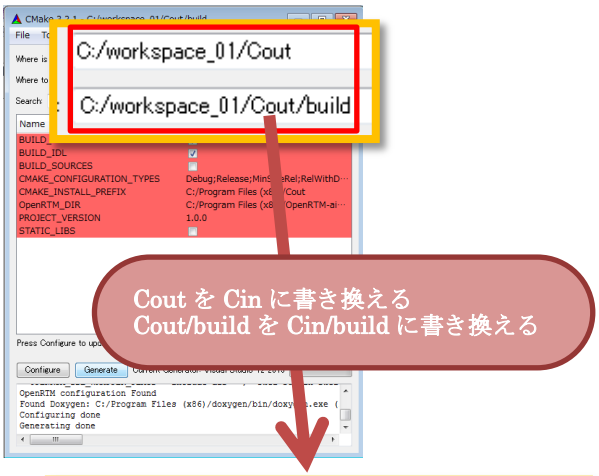


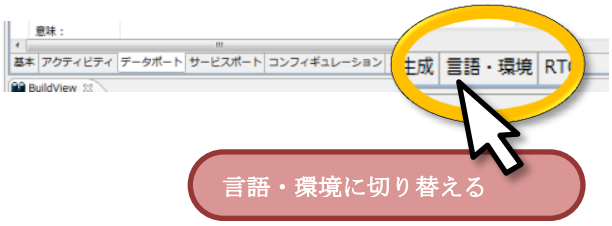
024 RTC ビルダーで設計する

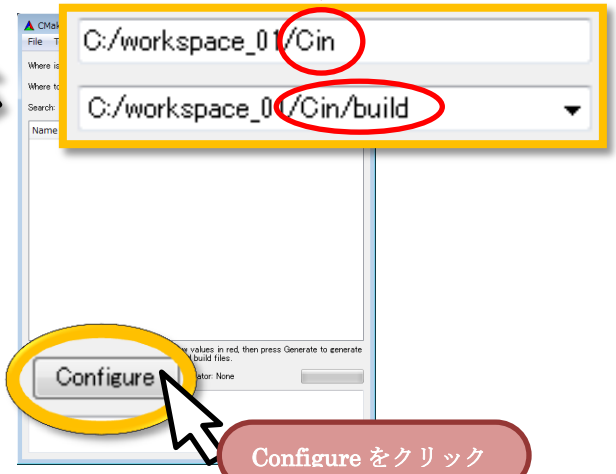




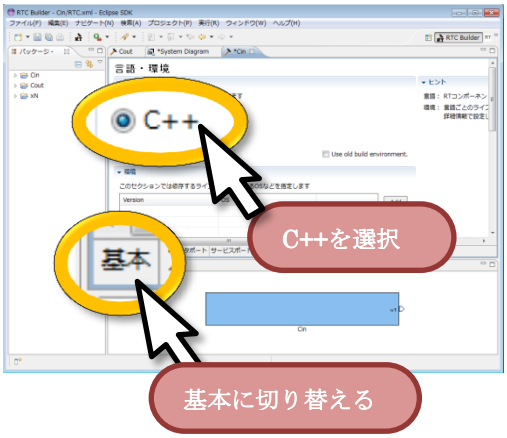


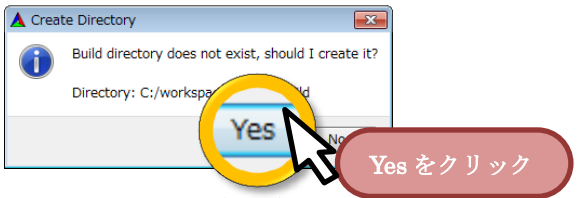




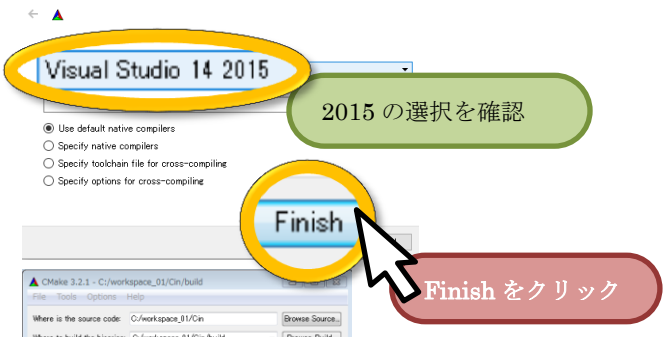


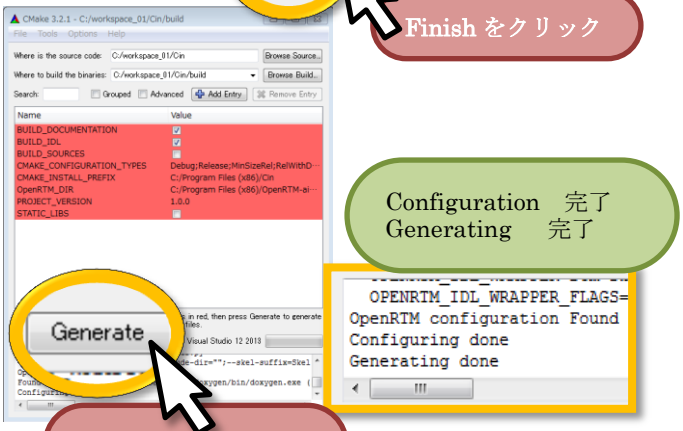
build フォルダ作成の確認． Yes.





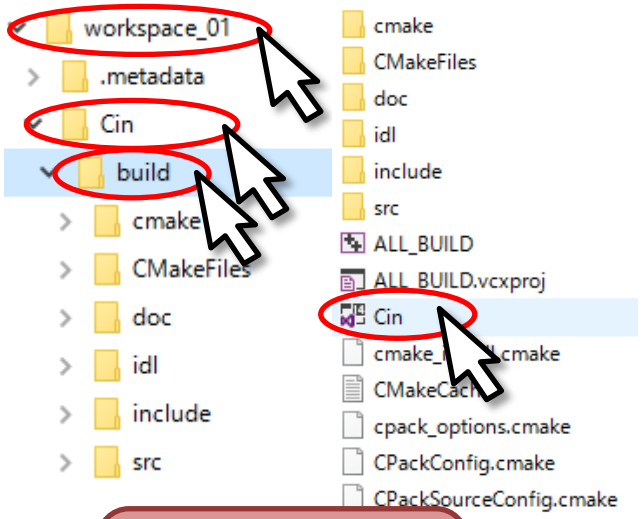
Visual Studio のバージョンの確認





026 エクスプローラーで Cin の下の build を開く.

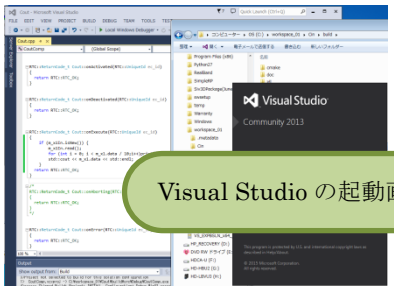
Cin のソリューションファイルをダブルクリック



Workspace_01
Cin
build
Cin

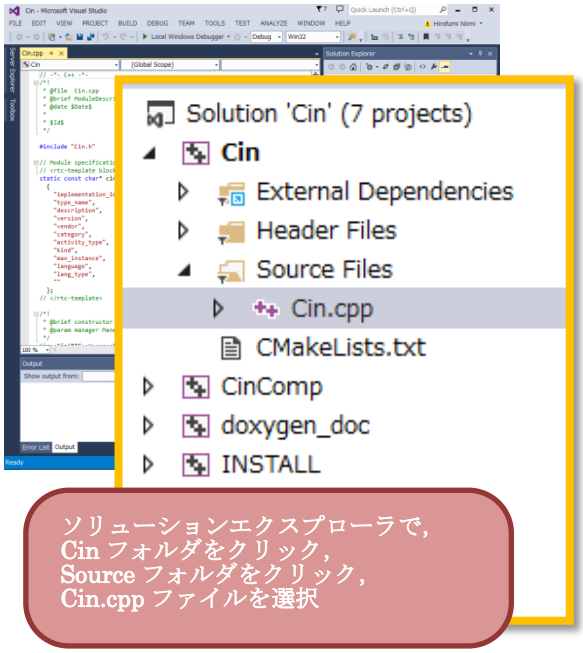
Cin をダブルクリック

Visual Studio の起動中の画面

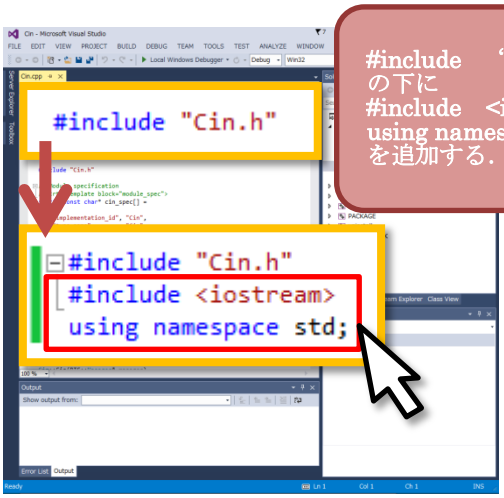


Visual Studio の起動画面

Cin.cpp ファイルを開く



ソリューションエクスプローラで,
Cin フォルダをクリック,
Source フォルダをクリック,
Cin.cpp ファイルを選択



#include “Cin.h”
の下に
#include <iostream>
using namespace std ;
を追加する.

```
55      RTC::ReturnCode_t Cin::onInitialize()
56      {
57        // Registration: InPort/OutPort/Service
58        // <rtc-template block="registration">
59        // Set InPort buffers
60
61        // Set OutPort buffer
62          cout << "onInitialize ¥n";
63        addOutPort("y1", m_y1Out);
64
65        // Set service provider to F
66
67        // Set servic
68
69        // Set CORBA
70
71        // </rtc-temp
72
73        // <rtc-template block="bind_config">
74        // </rtc-template>
75
76        return RTC::RTC_OK;
77      }
101     RTC::ReturnCode_t Cin::onActivated(RTC::UniqueId ec_id)
102     {
103         cout << "onActivated ¥n";
104       return RTC::RTC_OK;
105     }
106
107
108     RTC::ReturnCode_t Cin::onDeactivated(RTC::UniqueId ec_id)
109     {
110         cout << "onDeactivated ¥n";
111       return RTC::RTC_OK;
112     }
113
114
115     RTC::ReturnCode_t Cin::onExecute(RTC::UniqueId ec_id)
116     {
117         cout << "整数を入力してください. ¥n";
118         cin >> m_y1.data;
119         m_y1Out.write();
120       return RTC::RTC_OK;
121     }
```

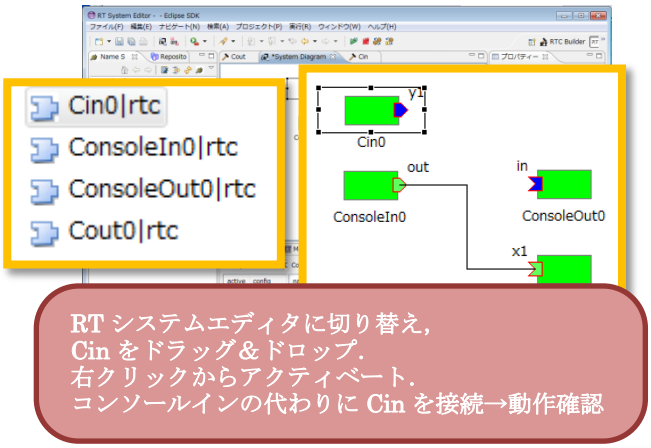
onInitialize()に,
cout<< “onInitialize” ;
を追加

onActivated()に, cout << “onActivated ¥n” ; を追加
onDeactivated()に, cout << “onDeactivated ¥n” ; を追加
onExecute()に, cout << “onExecute ¥n” ; を追加
onExecute()に, cin >> m_y1.data ; を追加
ecute()に, cout << “myOut.write() ; を追加
onError()に, cout << “on Error ¥n” ; を追加

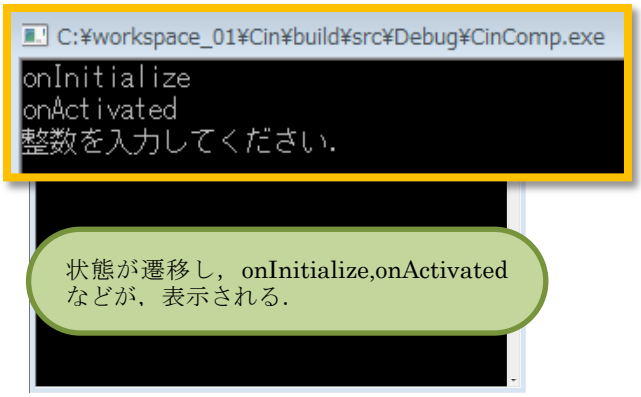
```
131     RTC::ReturnCode_t Cin::onError(RTC::UniqueId ec_id)
132     {
133         cout << "onError ¥n";
134       return RTC::RTC_OK;
135     }
```

Cin をビルドする

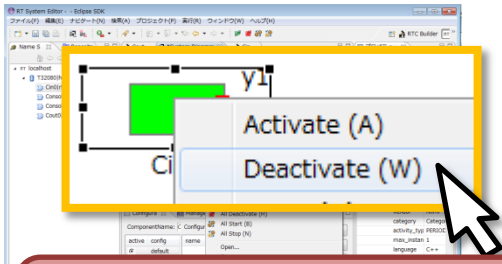
027 Cin の動作確認



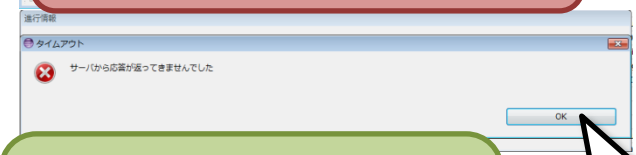
RT システムエディタに切り替え，
Cin をドラッグ＆ドロップ．
右クリックからアクティベート．
コンソールインの代わりに Cin を接続→動作確認



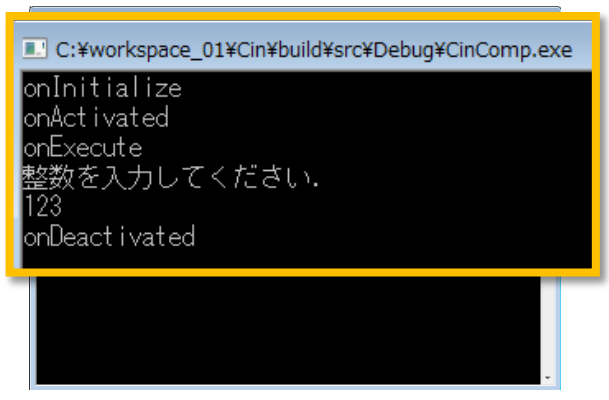
状態が遷移し，onInitialize,onActivated
などが，表示される．



右クリックから，ディアクテイベートする

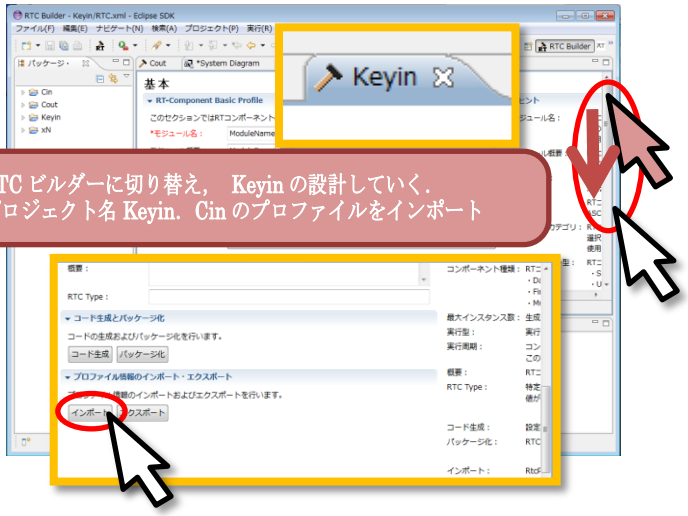


ディアクテイベートが入力待ち状態のため，失敗．
数字を入力すると，ディアクテイベートできる．



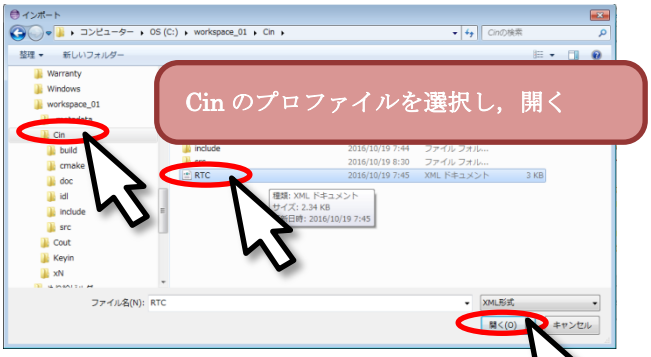
028 Keyin の設計

RTC Builder を使い，Keyin の設計していく．



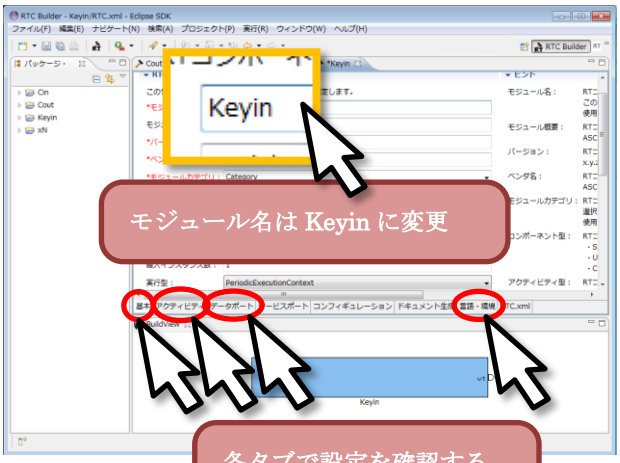
RTC ビルダーに切り替え， Keyin の設計していく．
プロジェクト名 Keyin．Cin のプロファイルをインポート

Cin のプロファイルをインポート



Cin のプロファイルを選択し，開く

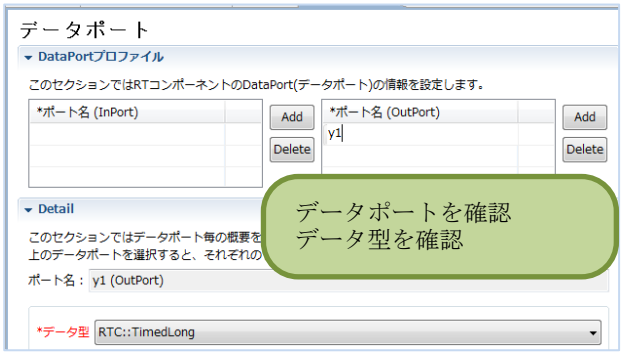
モジュール名を Cin→Keyin

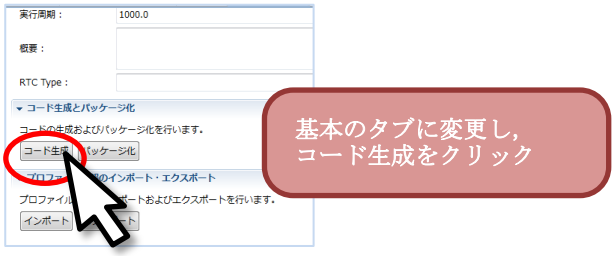


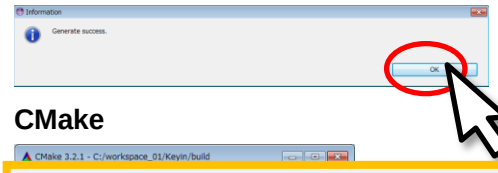
モジュール名は Keyin に変更

各タブで設定を確認する

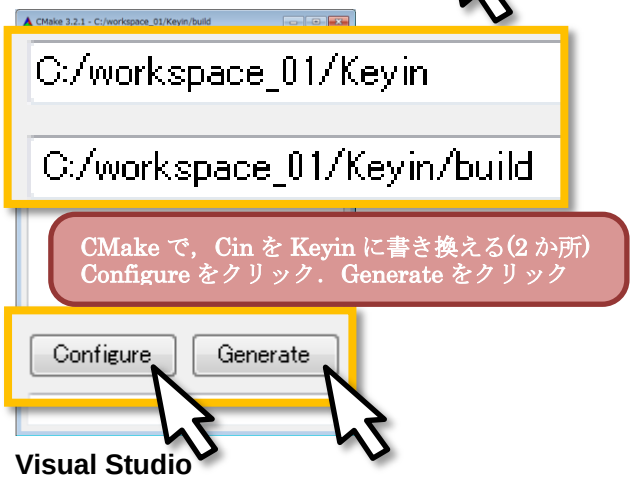
アクティビティを確認



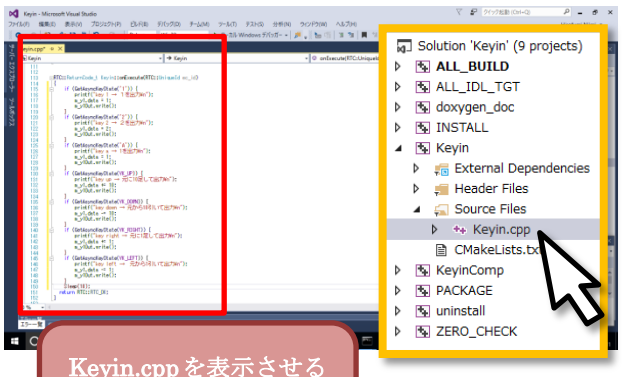




CMake



Visual Studio



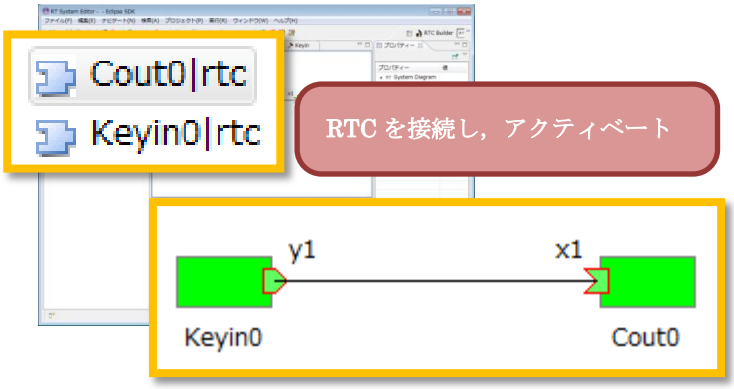
解説

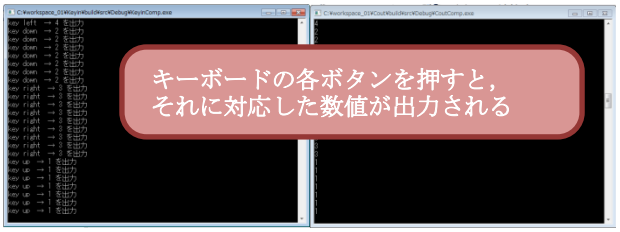
```
98  RTC::ReturnCode_t Keyin::onActivated(RTC::UniqueId ec_id)
99  {
100     printf("Activated [1,2,a,↑↓→←]¥n");
101     m_y1.data = 0;
102     m_y1Out.write();
103   return RTC::RTC_OK;
104 }
```

```
113 RTC::ReturnCode_t Keyin::onExecute(RTC::Un
114 {
115     if (GetAsyncKeyState('1')) {
116         printf("key 1 → 1を出力¥n");
117         m_y1.data = 1;
118         m_y1Out.write();
119     }
120     if (GetAsyncKeyState('2')) {
121         printf("key 2 → 2を出力¥n");
122         m_y1.data = 2;
123         m_y1Out.write();
124     }
125     if (GetAsyncKeyState('A')) {
126         printf("key a → 1を出力¥n");
127         m_y1.data = 1;
128         m_y1Out.write();
129     }
130     if (GetAsyncKeyState(VK_UP)) {
131         printf("key up → 元に10足して出力¥n");
132         m_y1.data += 10;
133         m_y1Out.write();
134     }
135     if (GetAsyncKeyState(VK_DOWN)) {
136         printf("key down → 元から10引いて出力¥n");
137         m_y1.data -= 10;
138         m_y1Out.write();
139     }
140     if (GetAsyncKeyState(VK_RIGHT)) {
141         printf("key right → 元に1足して出力¥n");
142         m_y1.data += 1;
143         m_y1Out.write();
144     }
145     if (GetAsyncKeyState(VK_LEFT)) {
146         printf("key left →  元から1引いて出力¥n");
147         m_y1.data -= 1;
148         m_y1Out.write();
149     }
150     Sleep(10);
151   return RTC::RTC_OK;
152 }
```

入力が大変な場合，if 文の塊を一つだけ入力して動作チェックを行うと良い．printf(“この部分”)は，コンソールに出力させる文字なので，簡略化しても良い．

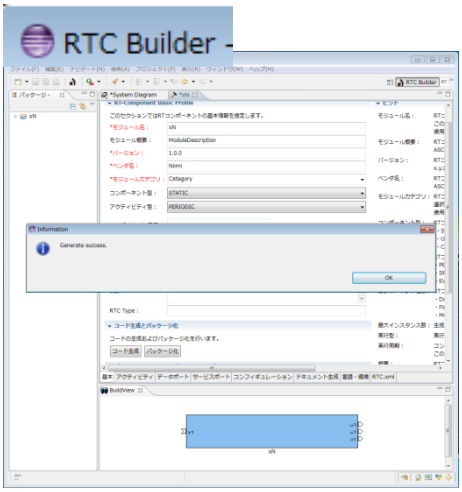
動作確認

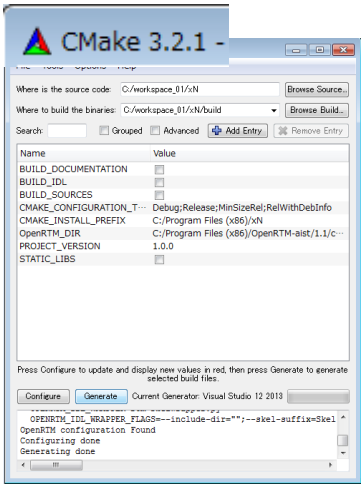


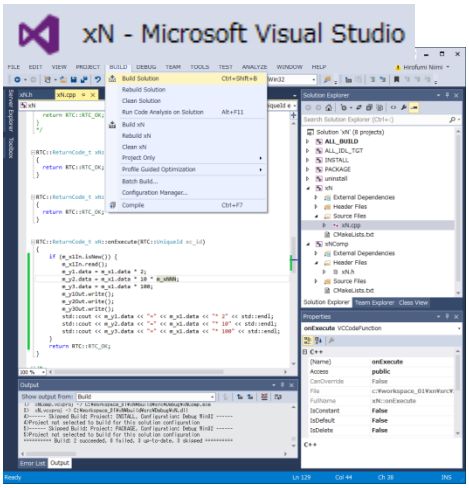


この練習課題の意図

RTC ビルダーでコンポーネントを構築し，CMake で必要なファイルを作り，VsualC++でコードを書いてビルドする． RT システムエディタで操作確認を行う．この操作を 3 回練習することにより，RTC 作成の基本手順が身につくように工夫した．RTC ビルダーでプロファイルをインポートする練習は，以前に作成した RTC に新たにポートを追加する場合に，使うことにできる方法である．

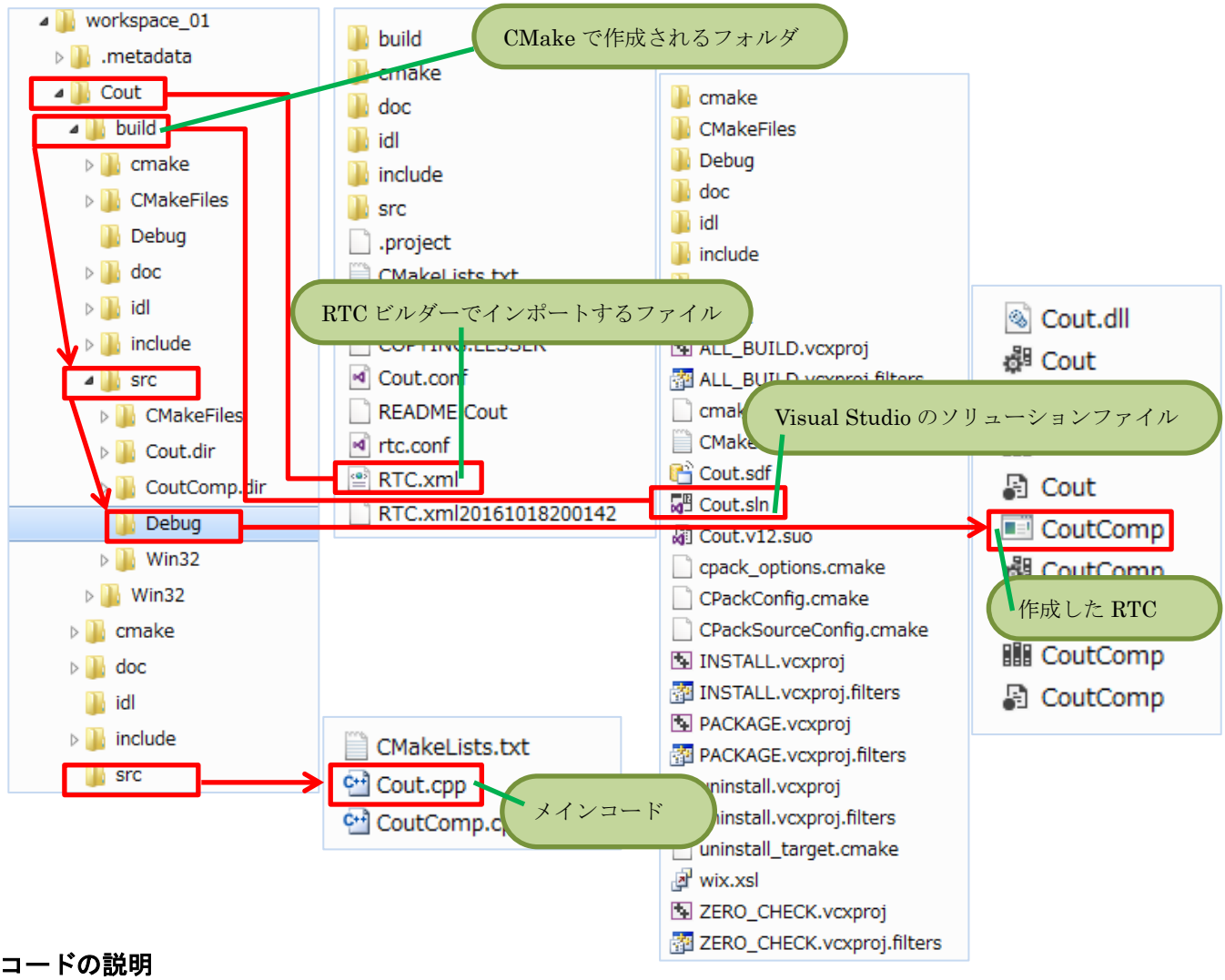






作成されたファイルの構造

ソリューションファイルや作成した RTC の場所を下記に示す．



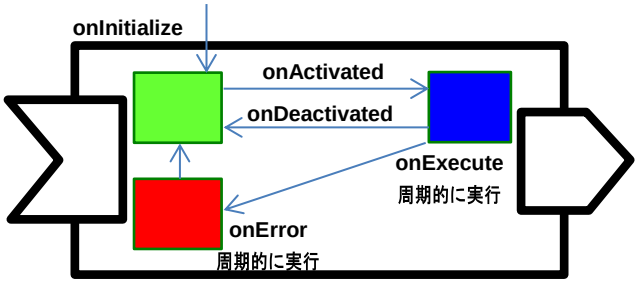
各コードの説明

Cout：コンソールアウトに相当する RTC．簡易棒グラフ機能付き．

x1In に新しい入力があったら，x1In を読む．"*"を出力する for 文．10 で“*”を 1 つ表示するとともに，x 1 の値を出力する．

```
RTC::ReturnCode_t Cout::onExecute(RTC::UniqueId ec_id)
{
    if (m_x1In.isNew()) {
        m_x1In.read();
        for (int i = 0; i < m_x1.data / 10;i++)printf("*");
        std::cout << m_x1.data << std::endl;
    }
  return RTC::RTC_OK;
}
```

Cin(021～027)



コンソールインに相当する RTC．状態の遷移を表示する．

#include<iostream>で標準入出力関数が使えようにする．

using namespace　std　使う名前空間の指定．

cout << "onInitialize ¥n";は onInitialize が実行されたとき，コンソールに表示する．onActivated, onDeaActivated, onError も同様に状態を表示できるようにする．

cout << "整数を入力してください．¥n";　　整数を入力したくださいと表示．

cin >> m_y1.data;　データの入力待ち状態．入力値は y 1 に渡される．

m_y1Out.write();　y 1 の値を出力する．

```
// -*- C++ -*-
/*!
 * @file  Cin.cpp
 * @brief ModuleDescription
 * @date $Date$
 *
 * $Id$
 */

#include "Cin.h"
#include <iostream>
using namespace std;
```

```
RTC::ReturnCode_t Cin::onInitialize()
{
  // Registration: InPort/OutPort/Service
  // <rtc-template block="registration">
  // Set InPort buffers

  // Set OutPort buffer
    cout << "onInitialize \n";
  addOutPort("y1", m_y1Out);

  // Set service provider to Ports

  // Set service consumers to Ports

  // Set CORBA Service Ports

  // </rtc-template>

  // <rtc-template block="bind_config">
  // </rtc-template>

  return RTC::RTC_OK;
}
```

```
RTC::ReturnCode_t Cin::onActivated(RTC::UniqueId ec_id)
{
    cout << "onActivated \n";
    return RTC::RTC_OK;
}


RTC::ReturnCode_t Cin::onDeactivated(RTC::UniqueId ec_id)
{
    cout << "onDeactivated \n";
    return RTC::RTC_OK;
}


RTC::ReturnCode_t Cin::onExecute(RTC::UniqueId ec_id)
{
    cout << "整数を入力してください．\n";
    cin >> m_y1.data;
    m_y1Out.write();

    return RTC::RTC_OK;
}

/*
RTC::ReturnCode_t Cin::onAborting(RTC::UniqueId ec_id)
{
  return RTC::RTC_OK;
}
*/


RTC::ReturnCode_t Cin::onError(RTC::UniqueId ec_id)
{
    cout << "onError \n";
    return RTC::RTC_OK;
}
```

Keyin(028)

1, 2,　a, ↑, ↓,←,→のキーが押されると, 1,2,1,もと＋10,もと－10,もと＋1,もと－1 を出力する RTC. キーが押されると，TimedLong 型の数値を出力する．1 が押されると 1 が出力し，2 が押されると 2 を出力する．a が押されると 1 を出力する．同じ要領で，3,4,5 や b,c,d を押すと数値を出力するプログラムをつくることができる．↑を押すと元に値に 10 足された値,↓を押すと元の値から 10 引かれた値が出力される．→を押すと元に値に 1 足された値, ←を押すと元の値から 1 引かれた値が出力される．

```
RTC::ReturnCode_t Keyin::onActivated(RTC::UniqueId ec_id)
{
    printf("Activated [1,2,a,↑↓→←]\n");
    m_y1.data = 0;
    m_y1Out.write();
    return RTC::RTC_OK;
}
```

```
RTC::ReturnCode_t Keyin::onExecute(RTC::UniqueId ec_id)
{
        if (GetAsyncKeyState('1')){
            printf("key 1  → 1 を出力\n");
            m_y1.data = 1;
            m_y1Out.write();
        }
        if (GetAsyncKeyState('2')){
            printf("key 2  → 2 を出力\n");
            m_y1.data = 2;
            m_y1Out.write();
        }
        if (GetAsyncKeyState('A')){
            printf("key a  → 1 を出力\n");
            m_y1.data = 1;
            m_y1Out.write();
        }
        if (GetAsyncKeyState(VK_UP)){
            printf("key up  → 元に10足して出力\n");
            m_y1.data += 10;
            m_y1Out.write();
        }
        if (GetAsyncKeyState(VK_DOWN)){
            printf("key down  → 元から10引いて出力\n");
            m_y1.data -= 10;
            m_y1Out.write();

        }
        if (GetAsyncKeyState(VK_RIGHT)){
            printf("key right  → 元に1足して出力\n");
            m_y1.data += 1;
            m_y1Out.write();
        }
        if (GetAsyncKeyState(VK_LEFT)){
            printf("key left  → 元から1引いて出力\n");
            m_y1.data -= 1;
            m_y1Out.write();
        }

    Sleep(10);
    return RTC::RTC_OK;
}
```

RTC で使用するデータの型

今回はすべて，TimedLong 型を使用した.