

第3部 プログラミング実習

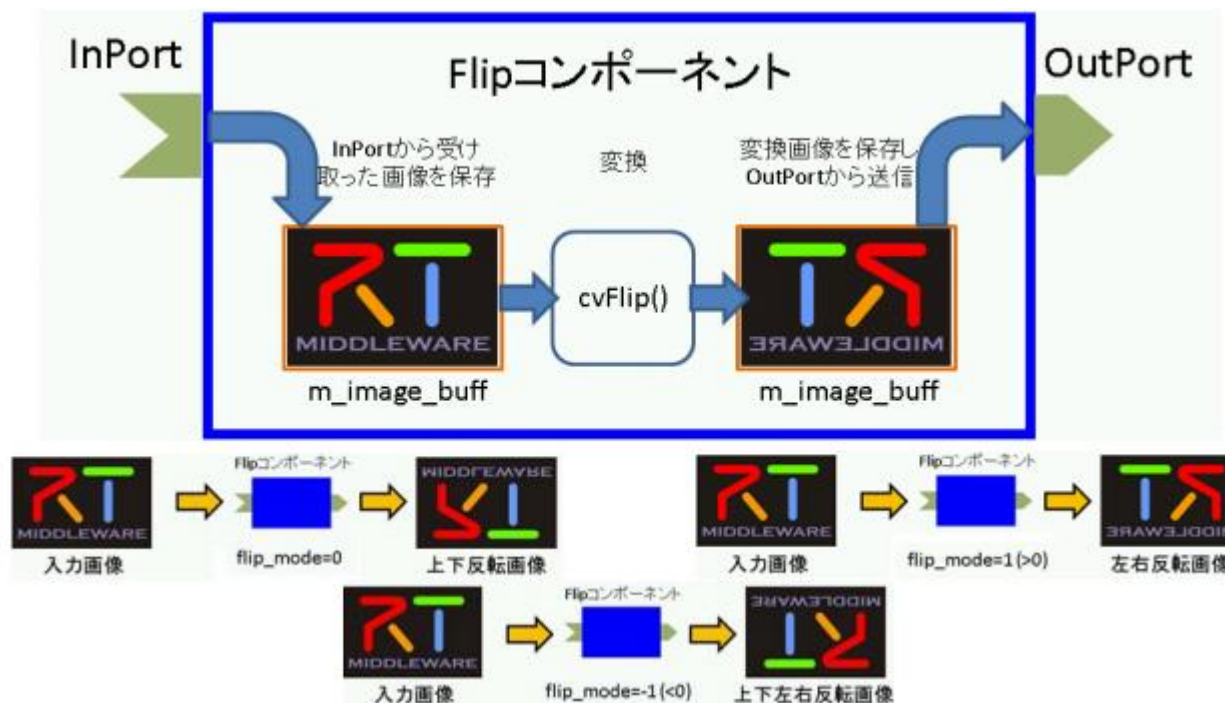
宮本 信彦

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



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

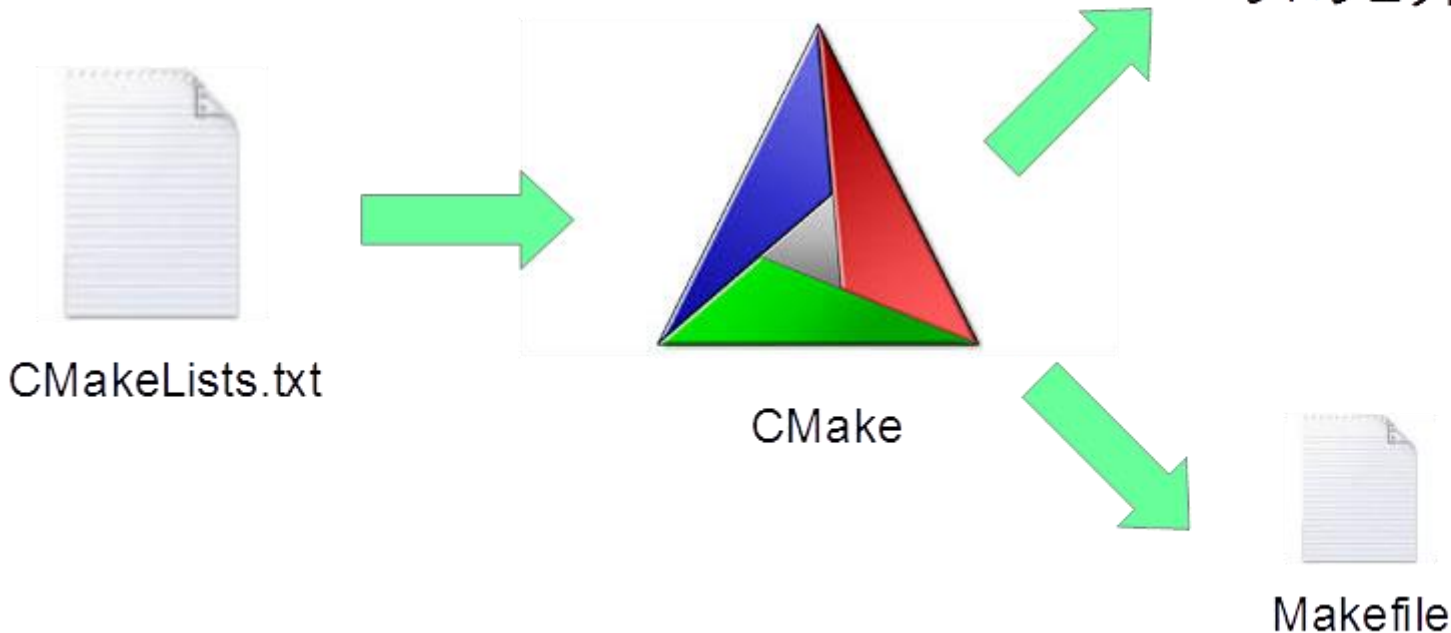
- RTC Builderによるスケルトンコードの作成
 - 作成済み
- ビルドに必要な各種ファイルを生成
 - CMakeLists.txtの編集
 - Cmakeにより各種ファイル生成
- ソースコードの編集
 - Flip.hの編集
 - Flip.cppの編集
- ビルド
- 動作確認

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



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



CMakeLists.txtの編集

- OpenCVを利用するためにCMakeLists.txtを修正する
 - workspace¥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 "${OPENRTM_INCLUDE_DIRS}") ↓  
9     list(APPEND OPENRTM_INCLUDE_DIRS "${CMAKE_CURRENT_SOURCE_DIR}/opencv/include")
```

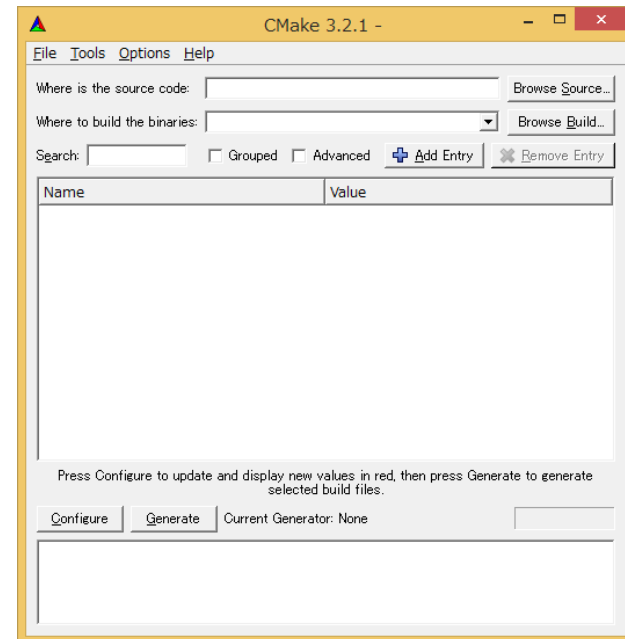
find_package(OpenCV REQUIRED)
を追加

```
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

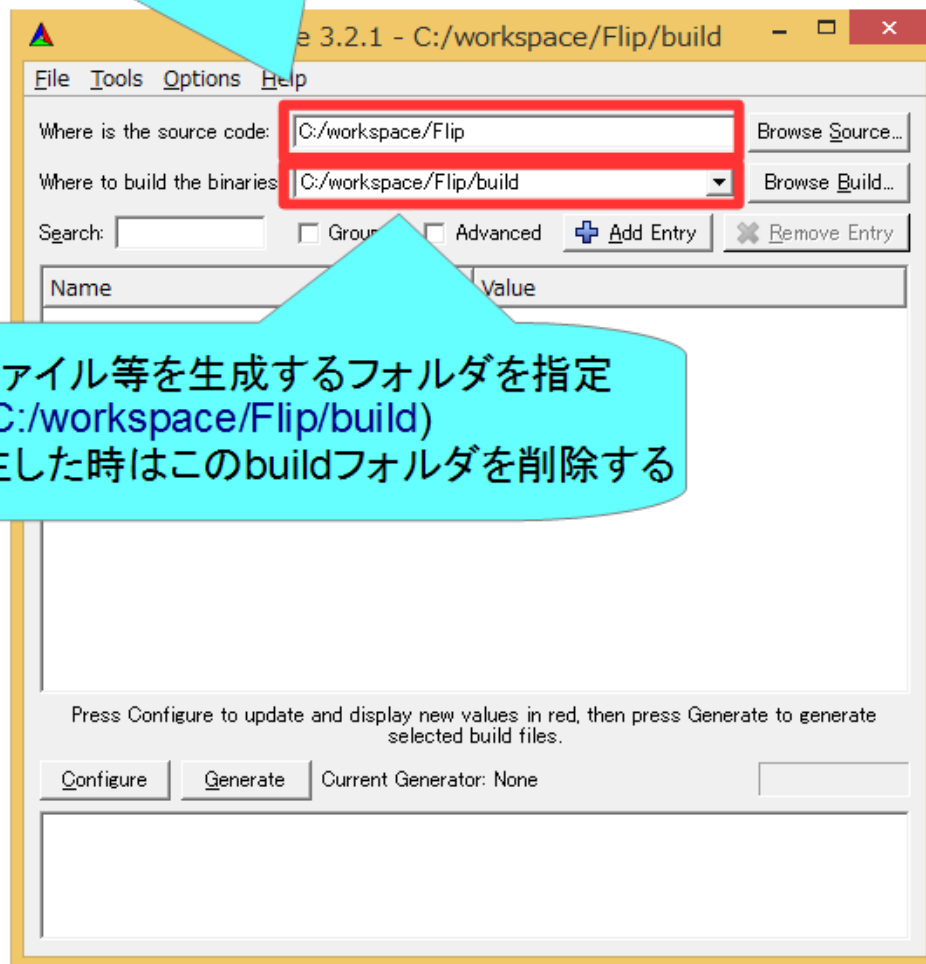
ビルドに必要なファイルの生成

- 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



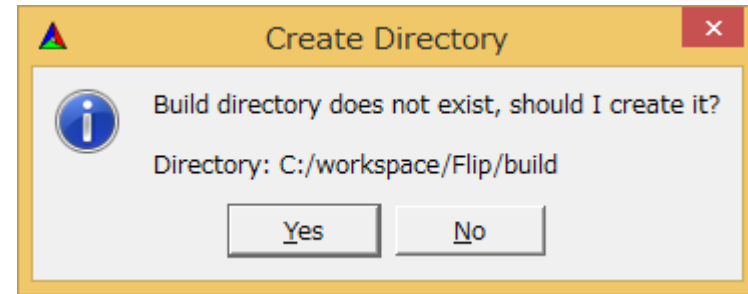
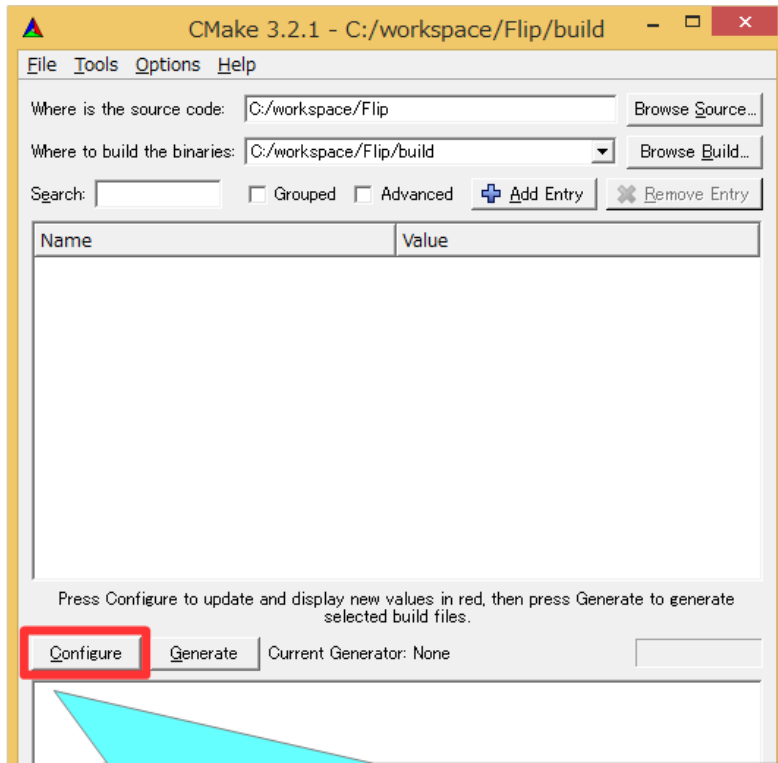
ビルドに必要なファイルの生成

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



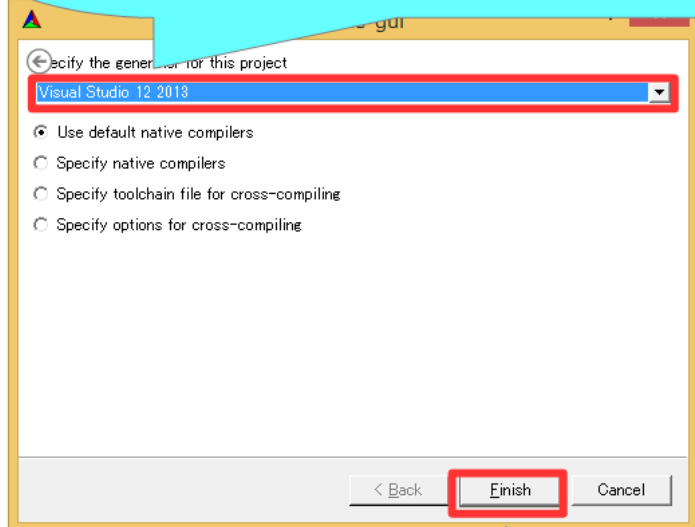
buildフォルダが存在しない場合は作成するかどうかきかれるため「Yes」を選択

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

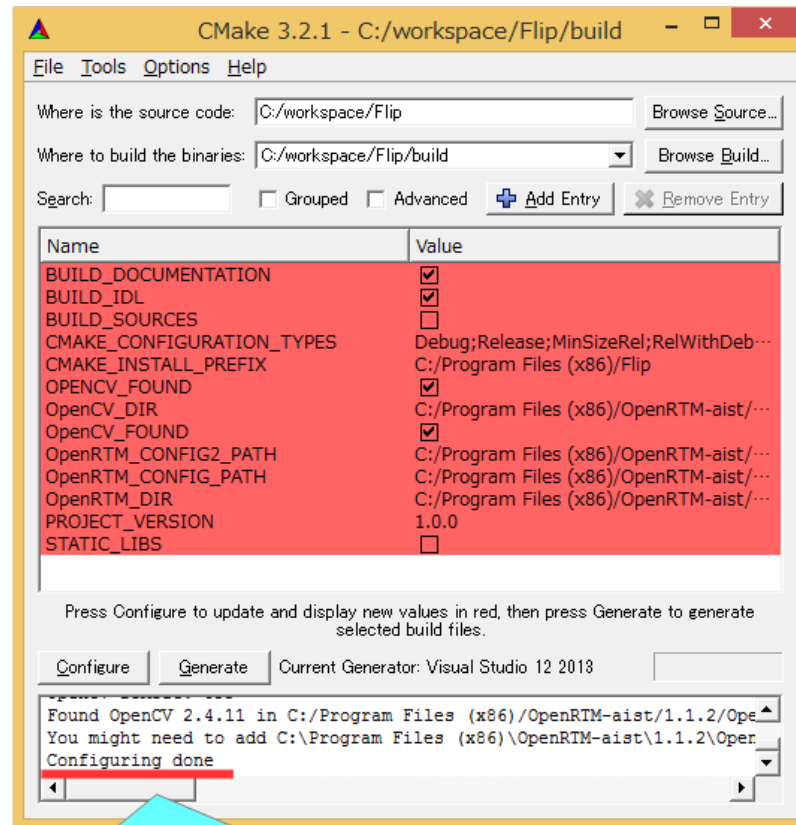
ビルドに必要なファイルの生成

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

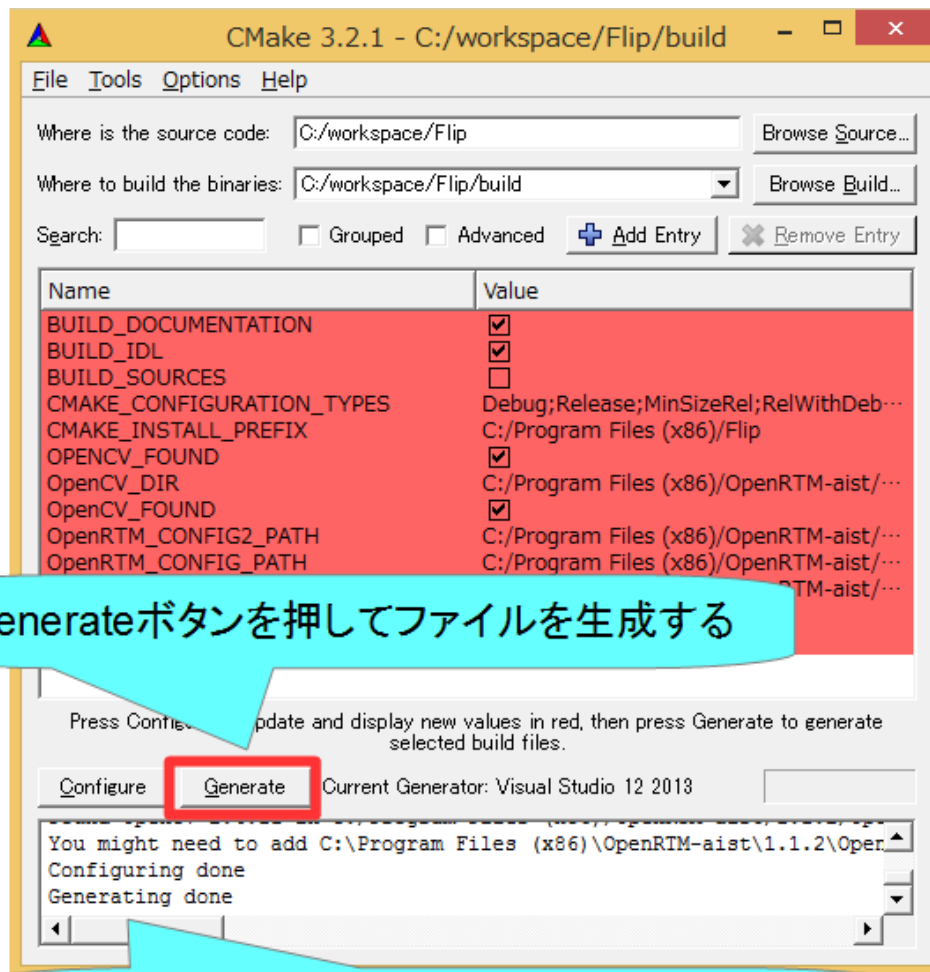


設定後、Finishボタンを押す



「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

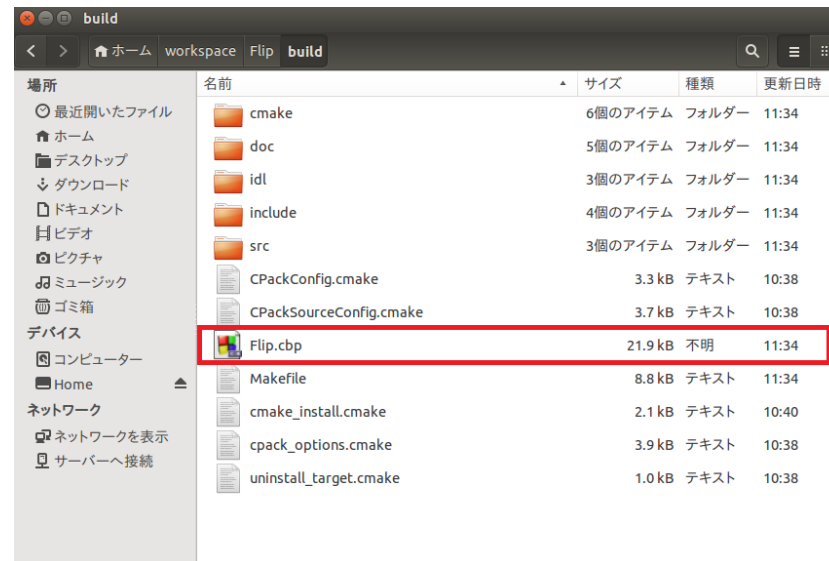
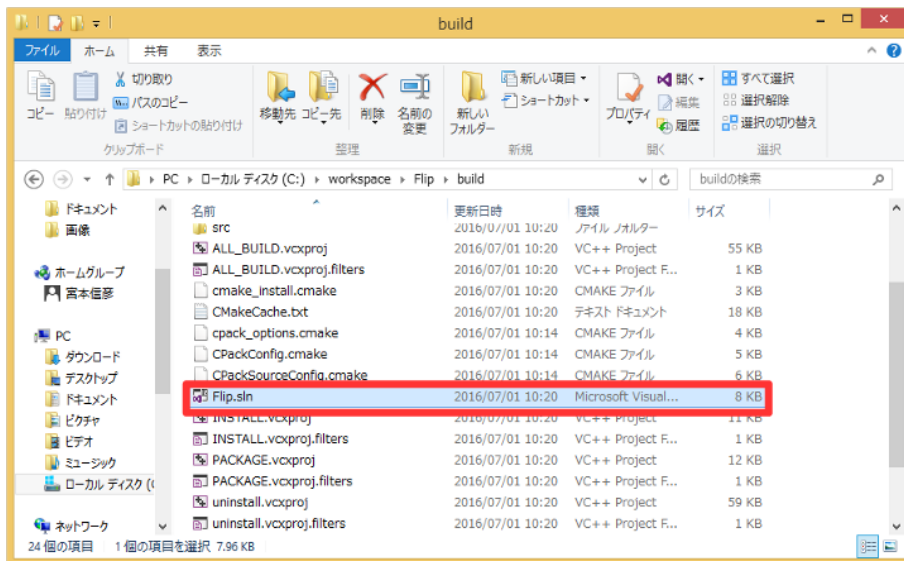


Generateボタンを押してファイルを生成する

「Generating done」が表示されていれば成功

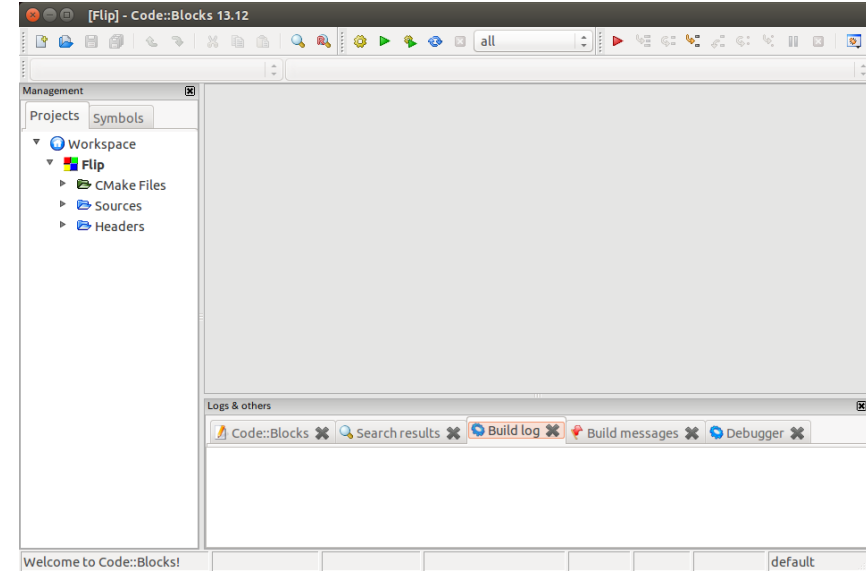
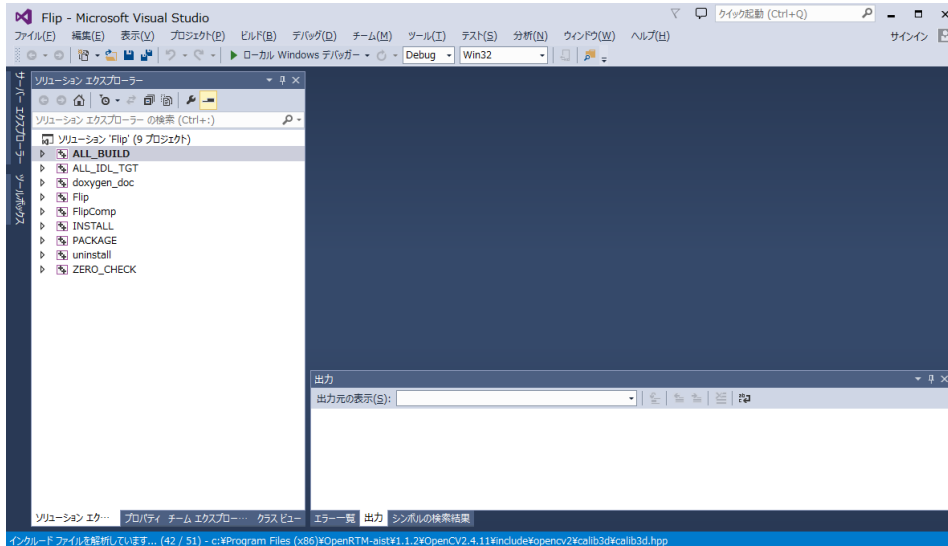
ソースコードの編集

- 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

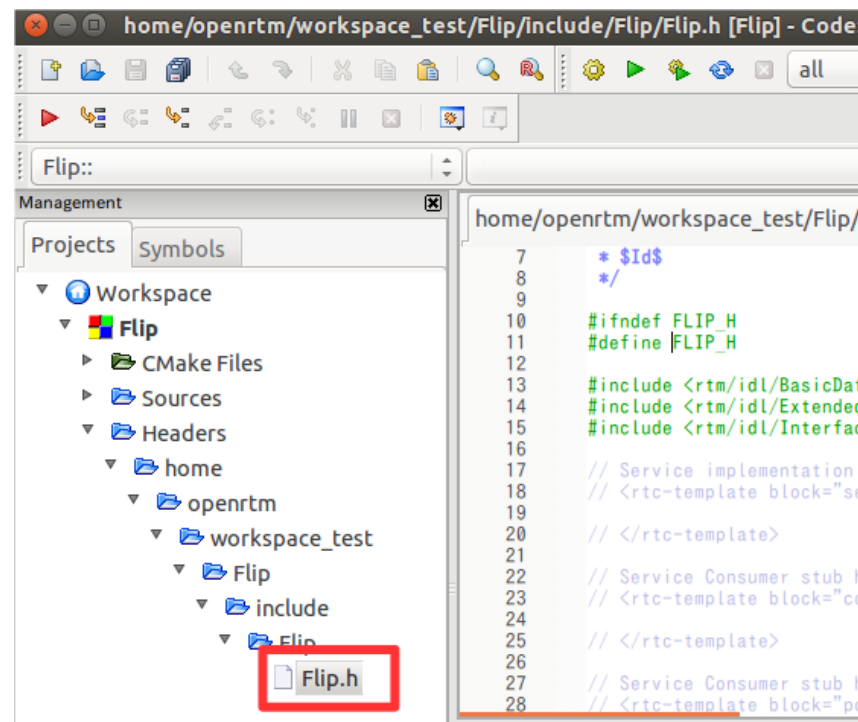
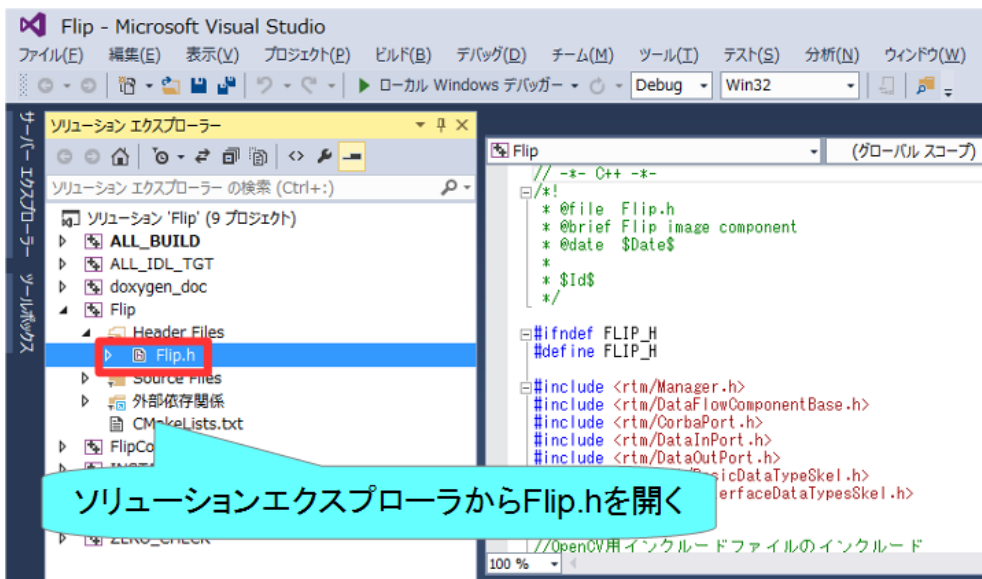


ソースコードの編集

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

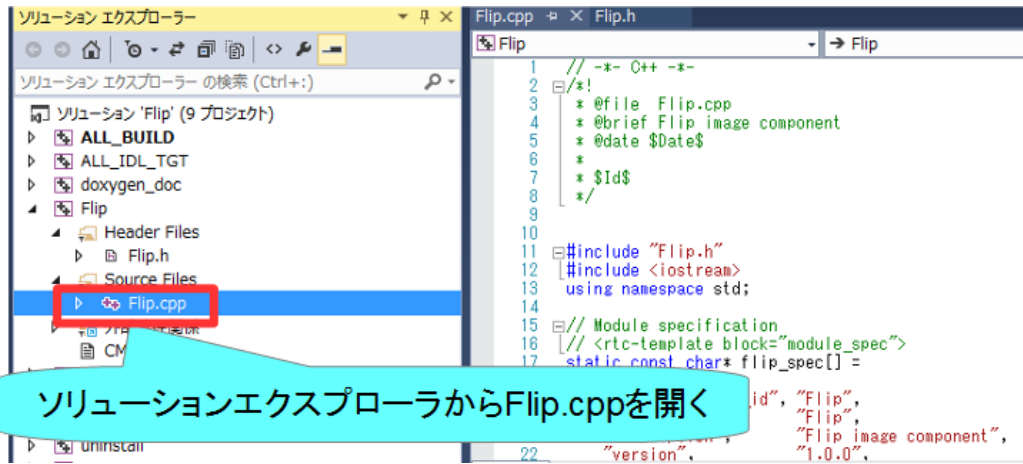
```

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

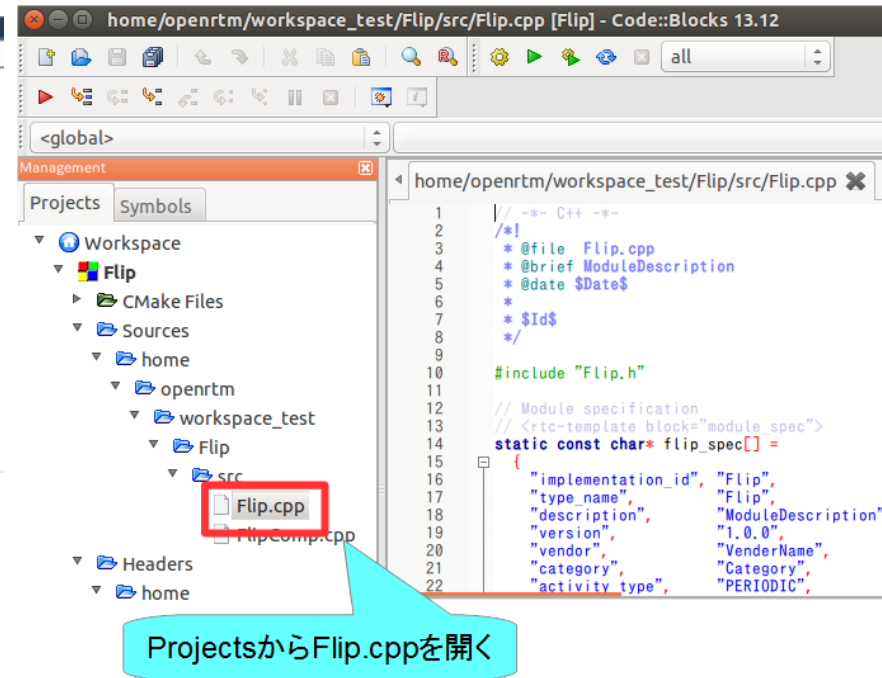
ソースコードの編集

- 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

```

134  RTC::ReturnCode_t Flip::onExecute(RTC::UniqueId ec_id)
135  {
136      // 新しいデータのチェック
137      if (m_originalImageIn.isNew()) {
138          // InPortデータの読み込み
139          m_originalImageIn.read();
140
141          // 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
142          if (m_originalImage.width != m_flippedImage.width || m_originalImage.height != m_flippedImage.height)
143          {
144              m_flippedImage.width = m_originalImage.width;
145              m_flippedImage.height = m_originalImage.height;
146
147              m_imageBuff.create(cv::Size(m_originalImage.width, m_originalImage.height), CV_8UC3);
148              m_flipImageBuff.create(cv::Size(m_originalImage.width, m_originalImage.height), CV_8UC3);
149
150          }
151
152          // InPortの画像データをm_imageBuffにコピー
153          memcpy(m_imageBuff.data, (void *)&(m_originalImage.pixels[0]), m_originalImage.pixels.length());
154
155          // 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
156          cv::flip(m_imageBuff, m_flipImageBuff, m_flipMode);
157
158          // 画像データのサイズ取得
159          int len = m_flipImageBuff.channels() * m_flipImageBuff.cols * m_flipImageBuff.rows;
160          m_flippedImage.pixels.length(len);
161
162          // 反転した画像データをOutPortにコピー
163          memcpy((void *)&(m_flippedImage.pixels[0]), m_flipImageBuff.data, len);
164
165          // 反転した画像データをOutPortから出力する。
166          m_flippedImageOut.write();
167      }
168  }
169
170  return RTC::RTC_OK;
171  }

```

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

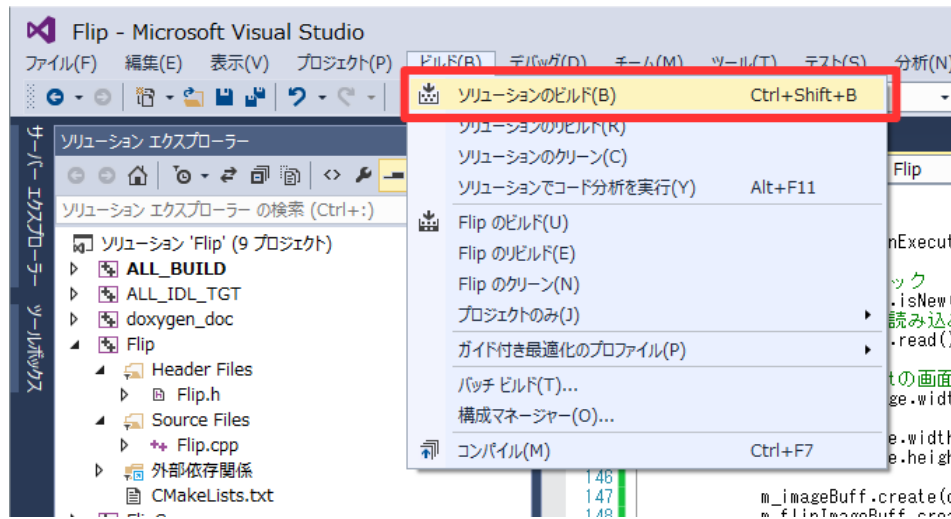
```
// 反転した画像データをOutPortにコピー
memcpy((void *)&(m_flippedImage.pixels[0]), m_flipImageBuff.data, len);

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

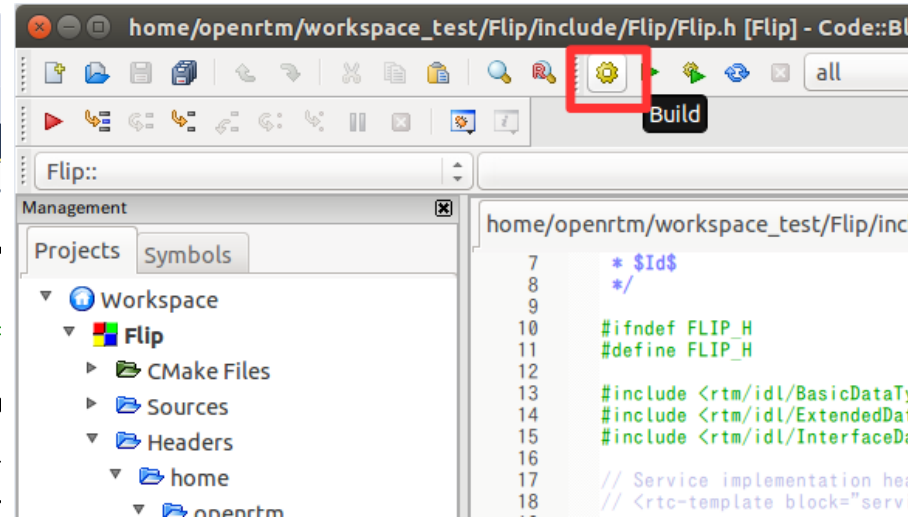
write関数でデータの書き込み

ソースコードのコンパイル

Visual Studio

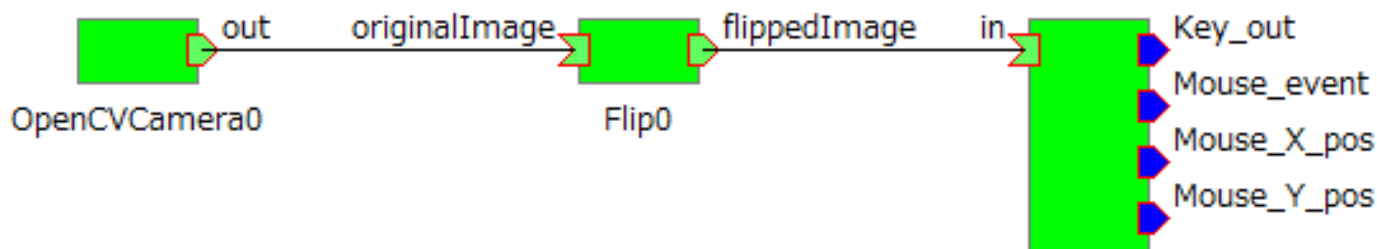


Code::Blocks



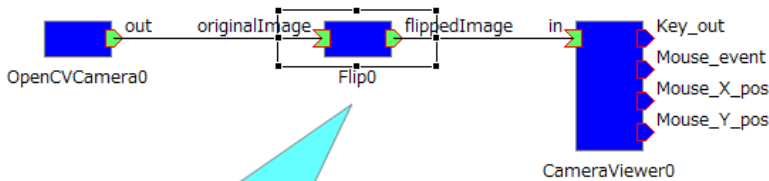
Flipコンポーネントの動作確認

- ネーミングサービスを起動する
- RT System Editor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/share/openrtm-1.1/components/c++/opencv-rtcs/CameraViewerComp`
 - `$ /usr/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



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



1. Flip0を選択する

Configuration... RT Manager Con... RT Composite C... RT Execution Co... RT RT Log View

ComponentName: Flip0 ConfigurationSet: default

active	config	name	value
<input checked="" type="checkbox"/>	default	flipMode	1

複製 追加

編集 適用 キャンセル

2. 編集ボタンを押す

Configuration

default

ConfigurationSet : default

flipMode -1 0 1

3. flipmodeを変更する

Apply

OK キャンセル

補足

rtc.confについて

- スケルトンコード作成時にFlipフォルダの下にrtc.confとFlip.confが生成されている
 - まずは**rtc.conf**、**Flip.conf**を実行ファイルを生成したフォルダ (build¥src¥Release、もしくはDebugフォルダ)にコピーしてください
 - この状態でFlipComp.exeを起動すると設定ファイルはrtc.confを読み込みます
 - コマンドで以下のように-fオプションで指定することも可能
 - FlipComp -f rtc.conf
 - rtc.confに以下の項目を追加してFlipComp.exeを起動することで動作が変化するかどうかを確認してください。
 - **naming.formats: %n.rtc**
 - ネームサーバへの登録名を変更する
 - **logger.enable: NO**
 - ログファイルを出力しないようにする
 - **logger.file_name: rtc_log.txt**
 - ログファイル名を変更する
 - **logger.log_level: TRACE**
 - ログレベルを変更する
 - **exec_cxt.periodic.rate: 1**
 - 実行周期を変更する

```

7 | # http://www.openrtm.org/openrtm/ko/content/rtcconf-%
8 | #↓
9 | ↓
10 | naming.formats: %n.rtc↓
11 | # logger.enable: NO↓
12 | logger.file_name: rtc_log.txt↓
13 | logger.log_level: TRACE↓
14 | exec_cxt.periodic.rate: 1↓
15 | ↓
16 | #-----
17 | # Typically used configuration parameters↓
^

```

rtc.confについて

- 各種設定を記述したファイル
 - マネージャが起動時に読み込み
 - 例えば以下の設定を記述することが可能
 - 1.2では起動時のポート自動接続、コンポーネントの自動アクティブ化等も設定可能になる予定です

項目名	内容
naming.formats	RTCをネームサーバーに登録する際のフォーマット
manager.modules.load_path	モジュール探索パス
manager.modules.preload	予めロードしておくモジュール
manager.components.precreate	予め起動しておくコンポーネント
logger.enable	ログの有効化、無効化(YES、NO)
logger.file_name	ログファイル名
logger.log_level	ログレベル(NORMAL、DEBUG等)
exec_cxt.periodic.type	デフォルトの実行コンテキストのタイプ (PeriodicExecutionContext等)
exec_cxt.periodic.rate	デフォルトの実行コンテキストの周期
corba.master_manager	マネージャをマスターかスレーブに設定(YES、NO)

コンポーネント名.confについて

- **Flip.conf**について
 - Flipのコンフィギュレーションパラメータの設定、実行コンテキストの設定等が可能
 - どのコンポーネントにどのconfファイルを対応させるかはrtc.confで設定可能
 - (カテゴリ名).(コンポーネント名). config_file
 - ImageProcessing.Flip.config_file: Flip.conf
 - conf.(コンフィギュレーションセット名).(パラメータ名)でパラメータのデフォルト値を設定できる
 - conf.default.flipMode: -1
 - コンフィギュレーションセットの追加も可能
 - conf.mode0.flipMode: -1
 - 実行コンテキストの設定手順はrtc.confと同じ
 - exec_cxt.periodic.type: PeriodicExecutionContext
 - exec_cxt.periodic.rate:1000.0

複数のコンポーネントを同一プロセスで起動する方法

- rtc.confを編集する
 - モジュール探索パスを設定
 - `manager.modules.load_path: ., C:¥¥Program Files (x86)¥¥OpenRTM-
aist¥¥1.1.2¥¥Components¥¥C++¥¥OpenCV¥¥vc12`
 - ※64bit版の場合、Visual Studio 2013以外の場合は適宜変更する
 - 予めロードするモジュールの設定
 - `manager.modules.preload: CameraViewer.dll, OpenCVCamera.dll`
 - 生成するコンポーネントの設定
 - `manager.components.precreate: CameraViewer, OpenCVCamera`
- 編集後、FlipComp.exeを実行する
 - CameraViewerとOpenCVCameraも同時に起動するため、CameraViewerとOpenCVCameraを既に起動している場合は終了させる
 - CameraViewer、OpenCVCamera、Flipが同時に起動できたかを確認する
- 同一プロセスで起動するメリットについて
 - データポート、サービスポートの通信が関数呼び出しになるため高速になる
 - 加えて1.2ではデータポートのdirectモードでの接続機能が追加されており、変数ポインタを直接渡すため非常に高速になる
- 同一プロセスで起動するデメリットについて
 - どれか一つのコンポーネントにトラブルが発生してプロセスが異常終了すると、すべてのコンポーネントが落ちる