Google Translate

Select Language

Powered by Google Translate

ナビゲーション

- ホーム
- ダウンロード
- ドキュメント
  - OpenRTM-aistとは?
  - OpenRTM-aistを10分で始めよう!
  - インストール
  - ツール
  - コンポーネント
  - デベロッパーズガイド
  - OpenRTM-aist クラスリファレンス
  - サンプルコンポーネント
  - ケーススタディ
    - 画像処理コンポーネントの作成
    - RTコンポーネント作成
    - RTコンポーネント作成(OpenCV編 CameraImage型の使用)
    - RTコンポーネント作成(LEGO Mindstorm)
    - RTコンポーネント作成(VC++編)
    - RTコンポーネント作成(Java版)
    - RTコンポーネントのクロス開発

ホーム >> 画像処理コンポーネントの作成 (OpenRTM-aist-1.1, CMake, VC2010)

画像処理コンポーネントの作成 (OpenRTM-aist-1.1, CMake, VC2010)

投稿者:root 投稿日時:木, 2011-11-10 17:43

Table of contents

はじめに

このケーススタディーでは、簡単な画像処理をコンポーネント化する方法 を紹介します。既存のカメラコンポーネントと画像表示コンポーネントを 利用し、カメラからの画像をj左右(または上下)に反転させる処理部分を コンポーネントとして作成してカメラの画像を反転させ表示するシステム を作成します。

画像を反転する処理は簡単に実装することができますが、ここではさらに 簡単のためにOpenCVライブラリを利用しより汎用性の高いRTコンポーネン トを作成します。

OpenCVとは

OpenCV (オープンシーブイ) とはかつてインテル が、現在はWillow Garageが開発・公開しているオープンソースのコン ピュータビジョン向けライブラリです。

リンク

OpenHRP3

動力学シミュレータ

Choreonoid

モーションエディタ/シミュレータ

OpenHRI

対話制御コンポーネント群

OpenRTP

統合開発プラットフォーム

産総研RTC集

産総研が提供するRTC集

Wikipediaより抜粋。

作成する**RT**コンポーネント

- Flipコンポーネント: OpenCVライブラリが提供する様々な画像処理関数のうち、cvFlip() 関数を用いて画像の反転を行うRTコンポーネント。

cvFlip関数の**RT**コンポーネント化

入力された画像を左右または上下に反転し出力するRTコンポーネントを、OpenCVライブラリのcvFlip関数を利用して作成します。作成および実行環 境はWindows上のVisual C++を想定しています。対象OpenRTM-aistのバー ジョンは 1.1.0 です。

作成手順はおおよそ以下のようになります。

- 動作環境・開発環境についての確認
- OpenCVとcvFlip関数についての確認
- コンポーネントの仕様を決める
- RTCBuilderを用いたソースコードのひな形の作成
- アクティビティ処理の実装
- コンポーネントの動作確認

cvFlip関数について

cvFlip 関数は、OpenCVで標準的に用いられているIplImage型の画像デー タを垂直軸 (左右反転)、水平軸 (上下反転)、または両軸 (上下左右反転) に対して反転させます。関数プロトタイプと入出力の引数の意味は以下の 通りです。

```
void cvFlip(IplImage* src, IplImage* dst=NULL, int fl
#define cvMirror cvFlip

src        入力配列
dst        出力配列。もしdst=NULLであれば、srcが上書きされま
flipMode 配列の反転方法の指定内容:
  flipMode = 0: X軸周りでの反転(上下反転)
  flipMode > 0: Y軸周りでの反転(左右反転)
  flipMode < 0: 両軸周りでの反転(上下左右反転)
```

コンポーネントの仕様

これから作成するコンポーネントをFlipコンポーネントと呼ぶことにします。

このコンポーネントは画像データ型の入力ポート (InPort) と、反転処理 した画像を出力するための出力ポート (OutPort) を持ちます。それぞれ のポートの名前を 入力ポート(InPort)名: **originalImage**, 出力ポー ト(OutPort)名: **flippedImage** とします。

OpenRTM-aistには OpenCVを使用したビジョン関連のコンポーネントがサ ンプルとして付属しています。これらのコンポーネントのデータポートは 画像の入出力に以下のような CameraImage 型を使用しています。

```
  struct CameraImage
```

```
    {
        /// Time stamp.
        Time tm;
        /// Image pixel width.
        unsigned short width;
        /// Image pixel height.
        unsigned short height;
        /// Bits per pixel.
        unsigned short bpp;
        /// Image format (e.g. bitmap, jpeg, etc.).
        string format;
        /// Scale factor for images, such as disparity
        /// where the integer pixel value should be di
        /// by this factor to get the real pixel value
        double fDiv;
        /// Raw pixel data.
        sequence<octet> pixels;
    };
```

このFlipコンポーネントではこれらのサンプルコンポーネントとデータの やり取りができるよう同じくCameraImage型をInPortとOutPortに使用する ことにします。CameraImage型はInterfaceDataTypes.idlで定義されており、C++であれば、InterfaceDataTypesSkel.hをインクルードすると使えるようになります。

また、画像を反転させる方向は、左右反転、上下反転、上下左右反転の3 通りが有ります。これを実行時に指定できるように、RTコンポーネントの コンフィギュレーション機能を使用して指定できるようにします。パラメータ名は **flipMode** という名前にします。

flipModeは cvFlip 関数の仕様に合わせて、型は int 型とし 上下反転、左右反転、上下左右反転それぞれに0, 1, -1 を割り当てることにします。

flipModeの各値での画像処理のイメージを図1に示します。

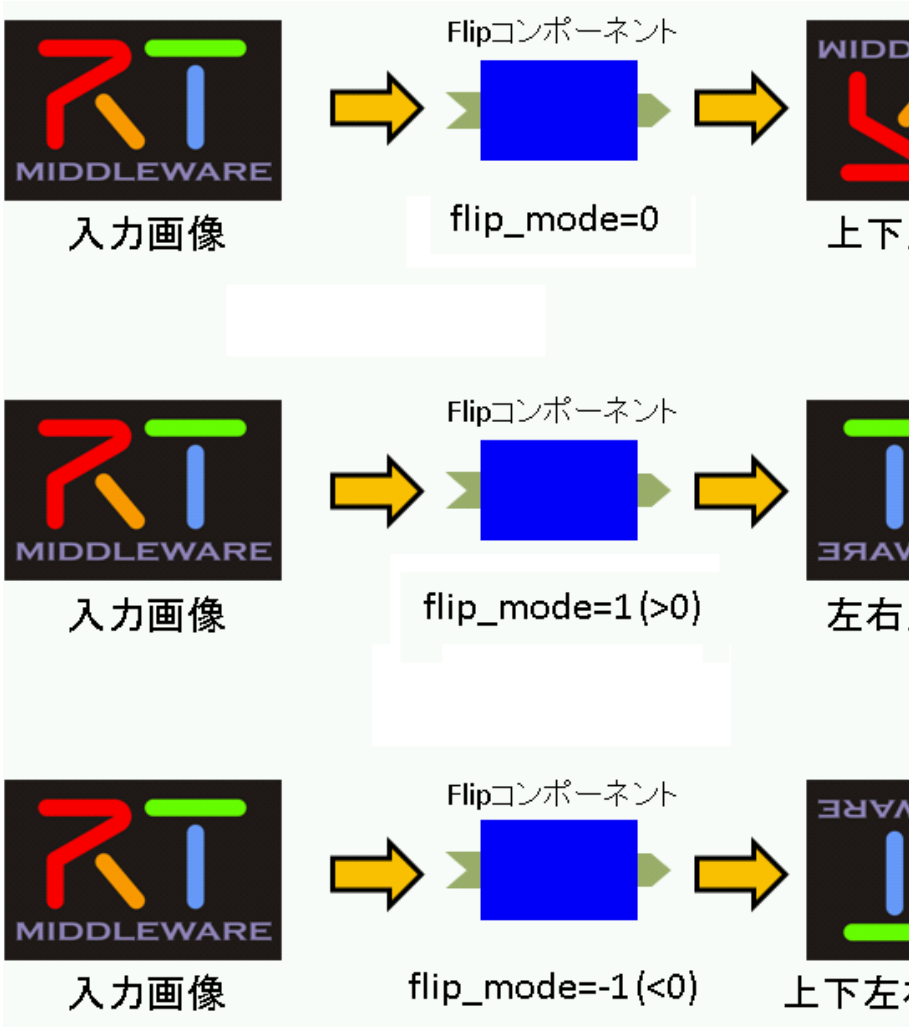

図1. **Flip**コンポーネントの**flipMode**指定時の画像反転パターン

以上からFlipコンポーネントの仕様をまとめます。

コンポーネント名称	Flip
InPort	
ポート名	originalImage
型	CameraImage
意味	入力画像
OutPort	
ポート名	flippedImage
型	CameraImage
意味	反転された画像
Configuration	
パラメータ名	flipMode
型	int

意味	反転モード 上下反転: 0 左右反転: 1 上下左右反転: -1

動作環境・開発環境

ここで動作環境および開発環境を確認しておきます。

- OS: Windows XP SP3 (Vista, 7でも可能)
- コンパイラ: Visual C++ 2010 Express Edition 日本語版
- OpenRTM-aist-1.1.0-RC3 (C++版), Win32 VC2010
- RTSystemEditor 1.1
- RTCBuilder 1.1
  - Eclipse3.4.2+RTSE+RTCB(1.1.0-RC2) Windows用全部入り
- Doxygen ドキュメント生成に必要
- CMake
- 解凍ツール(Lhaplus)

OpenRTM-aist-1.1以降では、コンポーネントのビルドにCMakeを使用しま す。また、RTCのひな形生成ツール RTCBuilder では、ドキュメントを入 力してこれを Doxygen に処理させることで、コンポーネントのマニュア ルも自動で生成することができるようになっており、このためCMakeで Configureを行うときにDoxygenが要求されるため予めインストールしてお く必要があります。

Flipコンポーネントの雛型の生成

Flipコンポーネントの雛型の生成は、RTCBuilderを用いて行います。

RTCBuilderの起動

新規ワークスペースを指定してeclipseを起動すると、以下のような Welcomeページが表示されます。

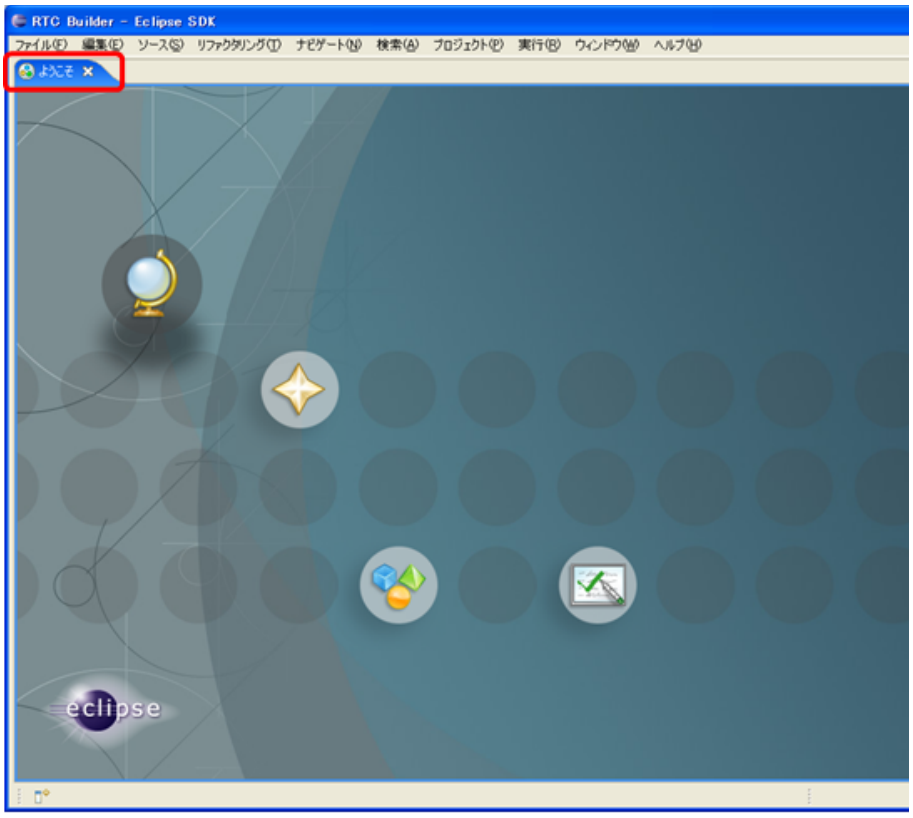

図**2. Eclipse**の初期起動時の画面

Welcomeページはいまは必要ないので左上の「x」ボタンを押して閉じてください。

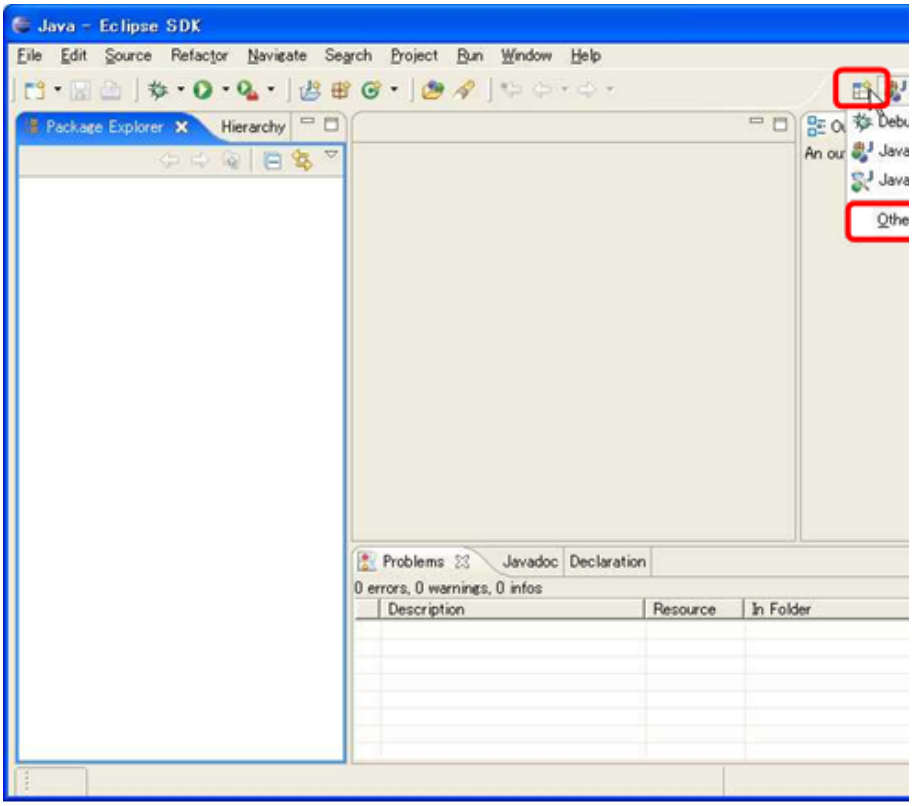

図**3.** パースペクティブの切り替え

右上の「Open Perspective」ボタンを押下し、プルダウンの「Other…」ボタンを押下します。

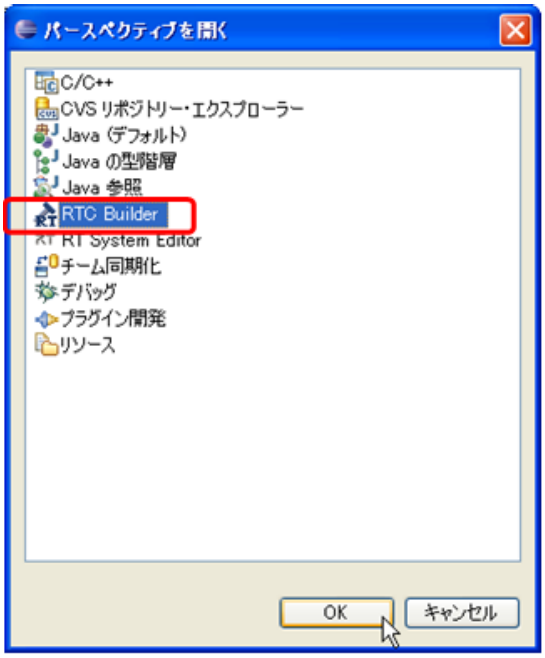

図**4.** パースペクティブの選択

「RTC Builder」を選択することで、RTCBuilderが起動します。

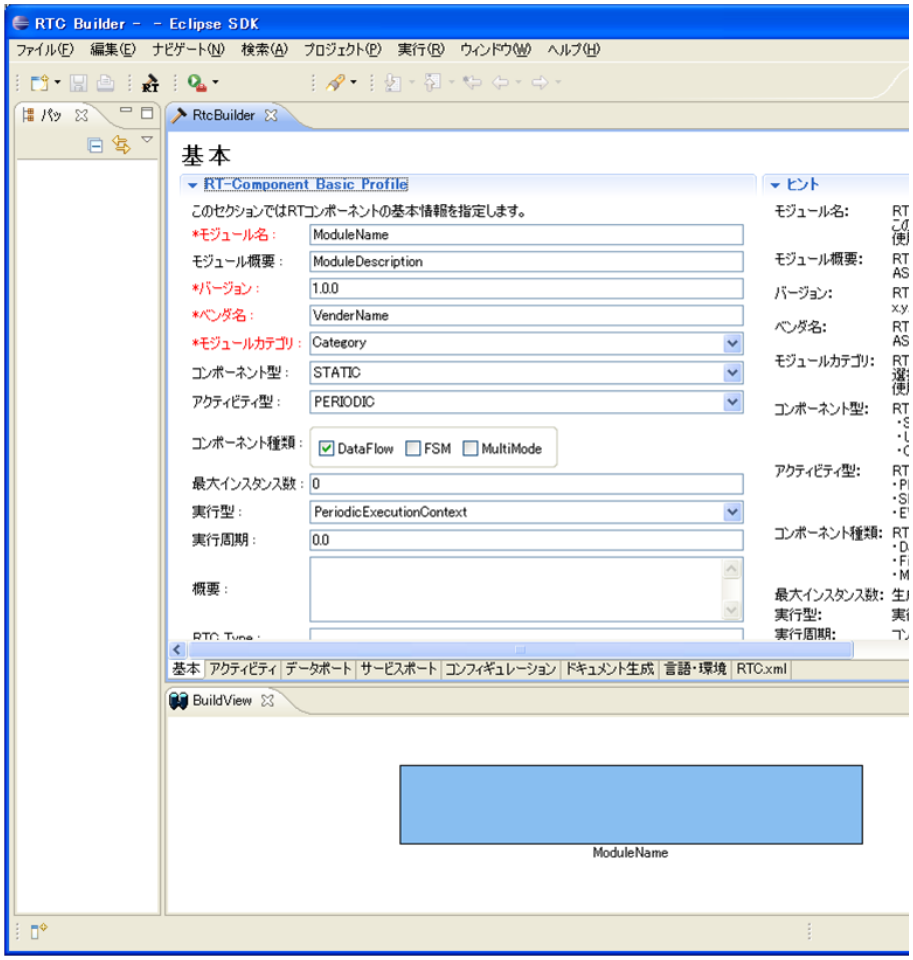

図5. RTC Builderの初期起動時画面

新規プロジェクトの作成

Flipコンポーネントを作成するために、RTCBuilderで新規プロジェクトを 作成する必要が有ります。画面上部のメニューから[ファイル]−[新規]− [プロジェクト]を選択します。

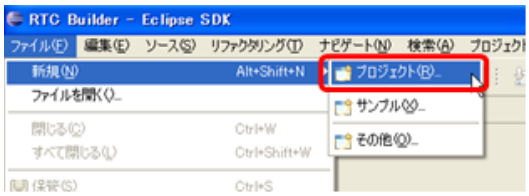

図6. RTC Builder 用プロジェクトの作成　1

表示された「新規プロジェクト」画面において,「その他」−「RTCビルダ」を選択し、「次へ」をクリックします。

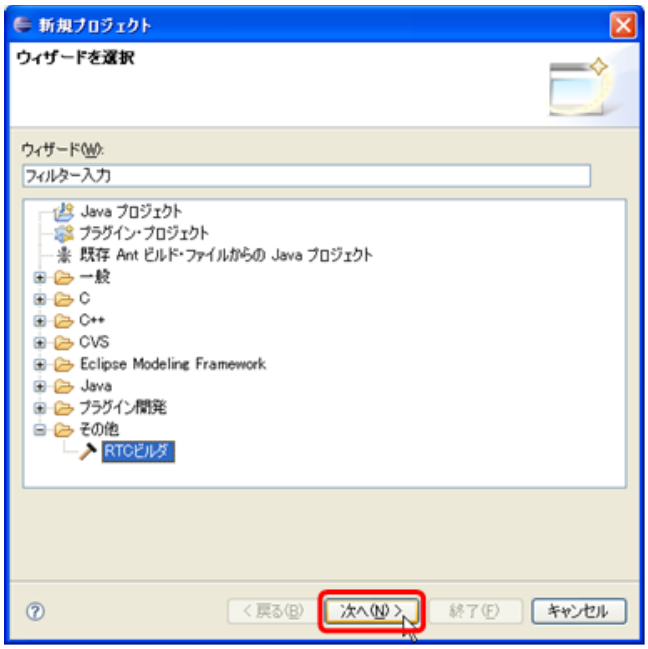

図**7. RTC Builder** 用プロジェクトの作成　2

「プロジェクト名」欄に作成するプロジェクト名 (ここでは **Flip**) を入力して「終了」をクリックします。

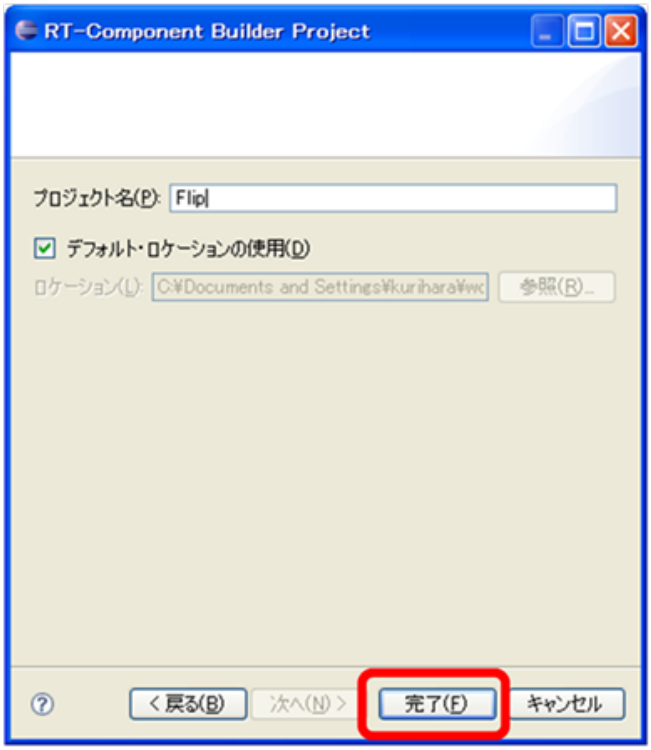

図**8. RTC Builder** 用プロジェクトの作成　3

指定した名称のプロジェクトが生成され、パッケージエクスプローラ内に追加されます。

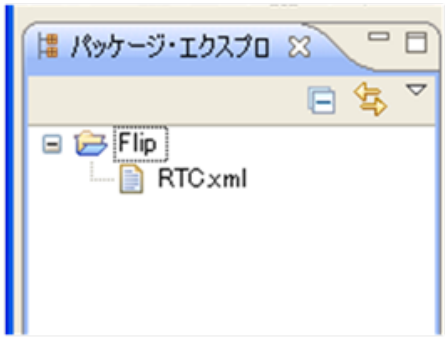

図**9. RTC Builder** 用プロジェクトの作成　4

生成したプロジェクト内には、デフォルト値が設定された RTC プロファ イル XML(RTC.xml) が自動的に生成されます。

RTC プロファイルエディタの起動

RTC.xmlが生成された時点で、このプロジェクトに関連付けられているワークスペースとして RTCBuilder のエディタが開くはずです。もし開かない場合は、ツールバーの「Open New RtcBuilder Editor」ボタンを押下するか、メニューバーの [file]-[Open New Builder Editor] を選択します。

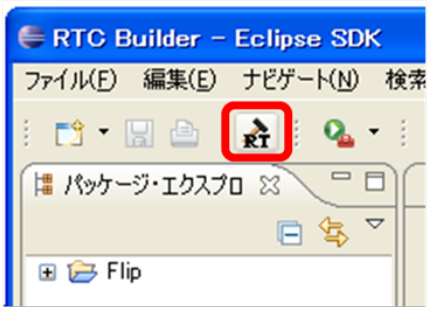

図**10.** ツールバーから **Open New RtcBuilder Editor**

図11. **File** メニューから **Open New Builder Editor**

プロファイル情報入力とコードの生成

まず、いちばん左の「基本」タブを選択し、基本情報を入力します。先ほ ど決めたFlipコンポーネントの仕様(名前)の他に、概要やバージョン等を 入力してください。ラベルが赤字の項目は必須項目です。その他はデフォ ルトで構いません。

- モジュール名: Flip
- モジュール概要: Flip image component
- バージョン: 1.0.0
- ベンダ名: AIST
- モジュールカテゴリ: Category
- コンポーネント型: STATIC
- アクティビティ型: PERIODIC
- コンポーネント種類: DataFlowComponent
- 最大インスタンス数: 1
- 実行型: PeriodicExecutionContext
- 実行周期: 0.0 (図13では1.0となってますが,0.0として下さい.)

  -Output Project: Flip

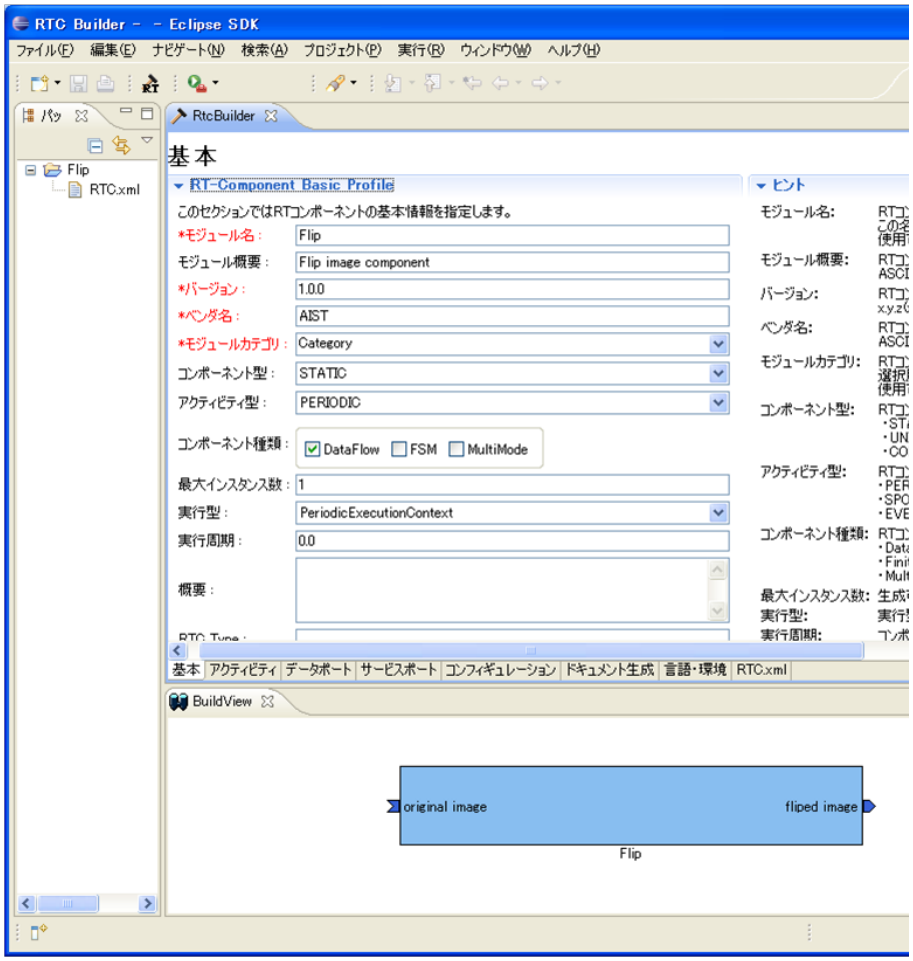

図13. 基本情報の入力

次に、「アクティビティ」タブを選択し、使用するアクションコールバッ クを指定します。

Flipコンポーネントでは、onActivated(),onDeactivated(),onExecute() コールバックを使用します。図14のように①のonAtivatedをクリック後に (2)のラジオボタンにて"on"にチェックを入れます。onDeactivated, onExecuteについても同様の手順を行います。

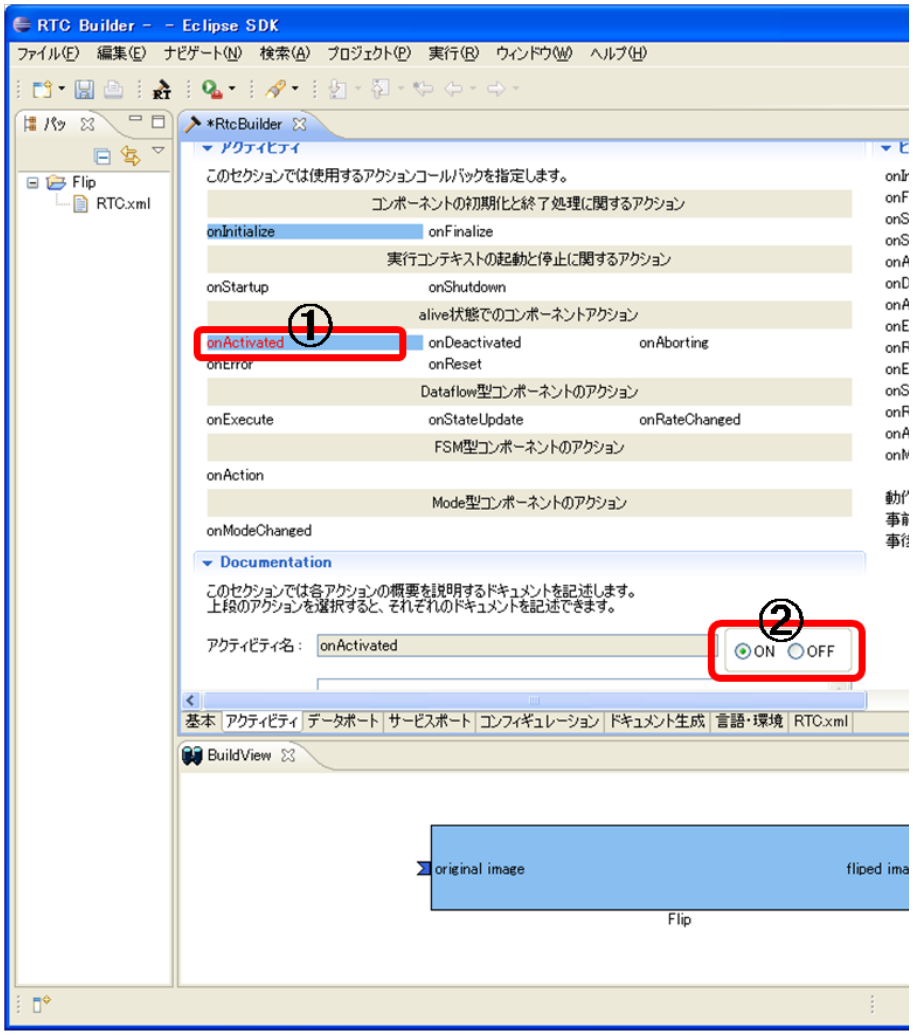

図**14.** アクティビティコールバックの選択

さらに、「データポート」タブを選択し、データポートの情報を入力します。先ほど決めた仕様を元に以下のように入力します。なお、変数名や表示位置はオプションで、そのままで結構です。

-InPort Profile:

- ポート名: originalImage
- データ型: CameraImage
- 変数名: originalImage
- 表示位置: left

-OutPort Profile:

- ポート名: flippedImage
- データ型: CameraImage
- 変数名: flippedImage
- 表示位置: right

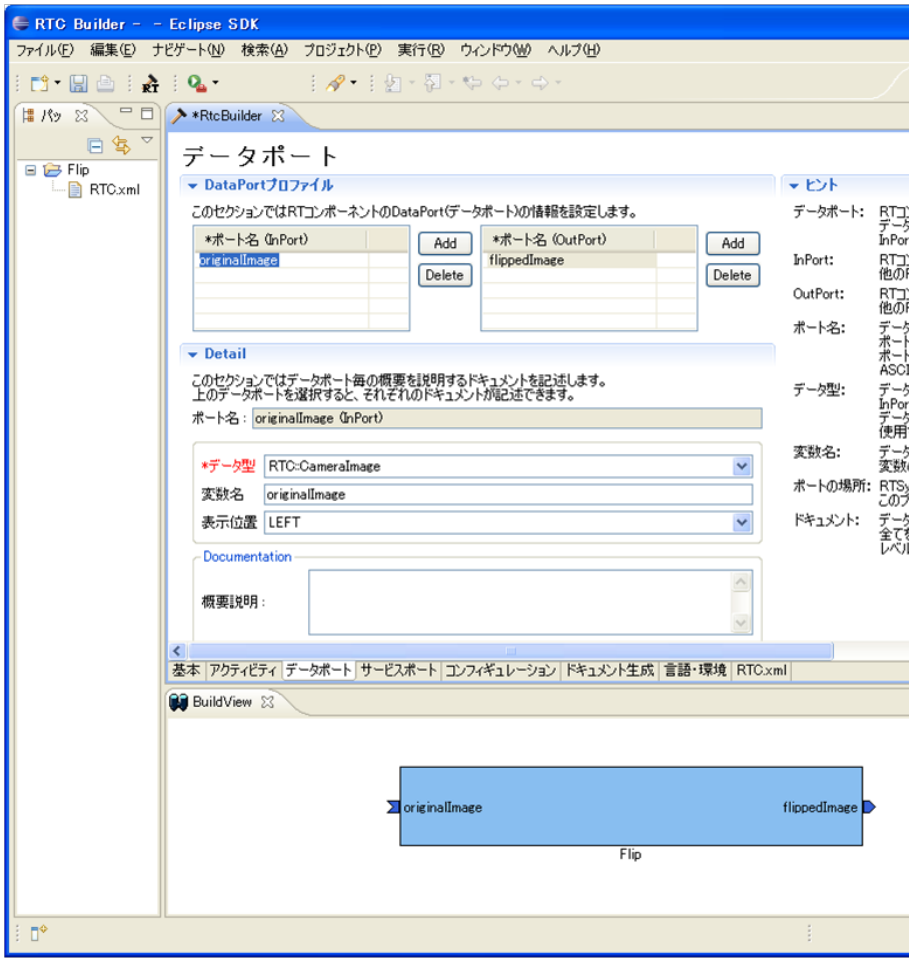

図**15.** データポート情報の入力

次に、「コンフィギュレーション」タブを選択し、先ほど決めた仕様を元 に、Configurationの情報を入力します。制約条件およびWidgetとは、RTSystemEditorでコンポーネントのコンフィギュレーションパラメータを 表示する際に、スライダ、スピンボタン、ラジオボタンなど、GUIで値の 変更を行うためのものです。

ここでは、flipModeが取りうる値は先ほど仕様を決めたときに、-1,0,1の 3つの値のみ取ることにしたので、ラジオボタンを使用することにします。

-flipMode

- 名称: flipMode
- データ型: int
- デフォルト値: 1

- 変数名: flipMode
- 制約条件: (-1, 0, 1) ※ (-1: 上下左右反転, 0: 上下反転, 1: 左右反転)
- Widget: radio

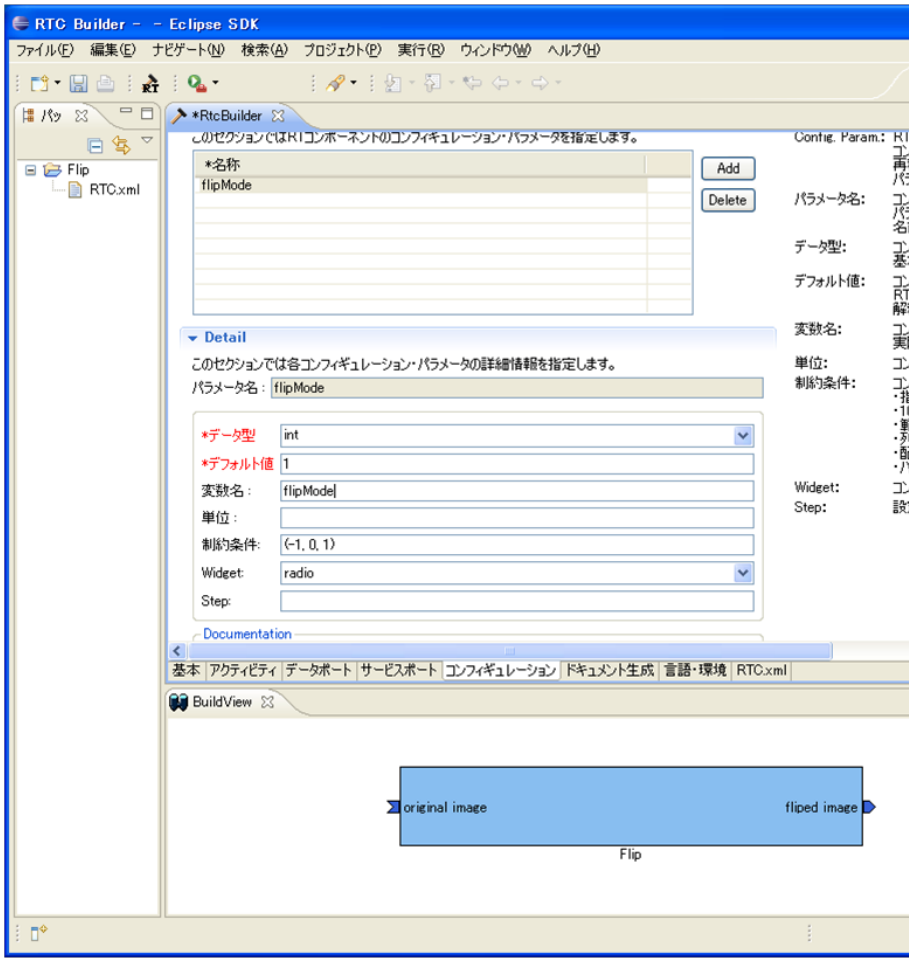

図**16.** コンフィグレーション情報の入力

次に、「言語・環境」タブを選択し、プログラミング言語を選択します。ここでは、C++(言語)を選択します。なお、言語・環境はデフォルト等が 設定されておらず、指定子忘れるとコード生成時にエラーになりますので、必ず言語の指定を行うようにしてください。

また、C++の場合デフォルトではCMakeを利用してビルドすることになって いますが、旧式のVCのプロジェクトやソリューションを直接RTCBuilderが 生成する方法を利用したい場合は **Use old build environment** を チェックしてください。

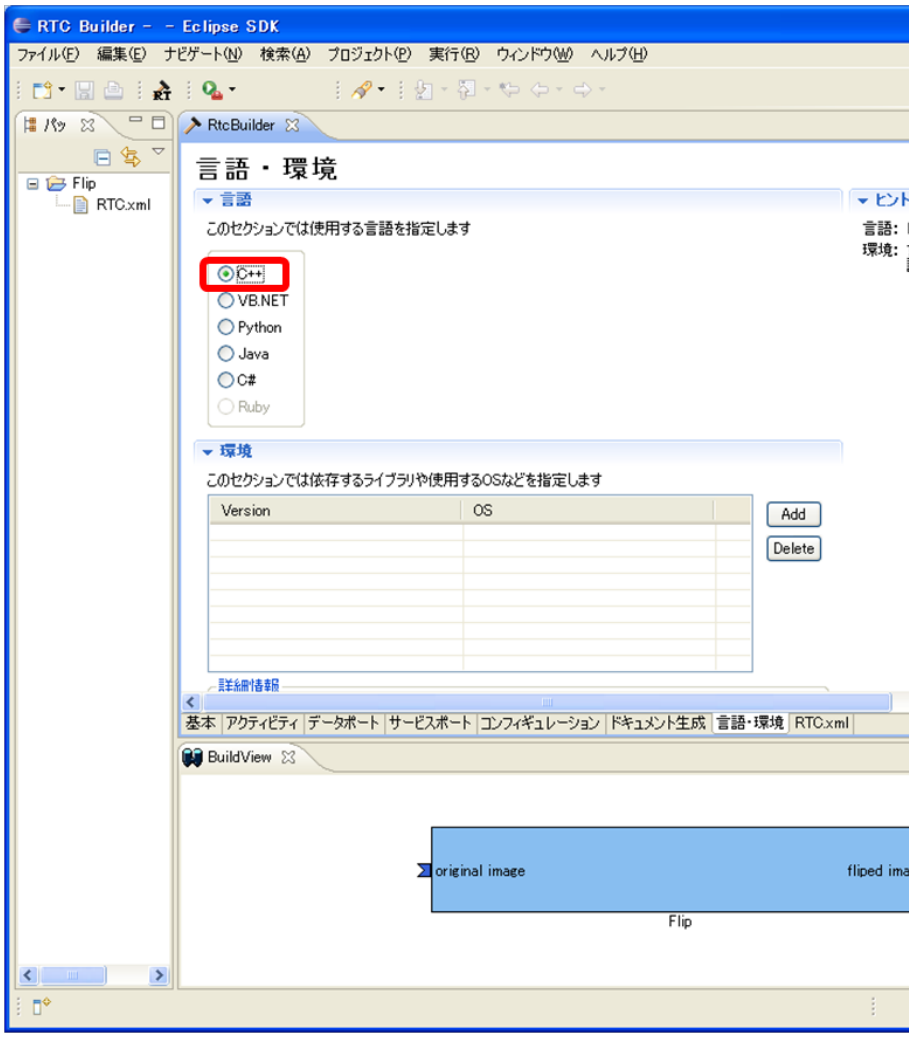

図**17.** プログラミング言語の選択

最後に、「基本」タブにある"コード生成"ボタンをクリックし、コンポー ネントの雛型を生成します。

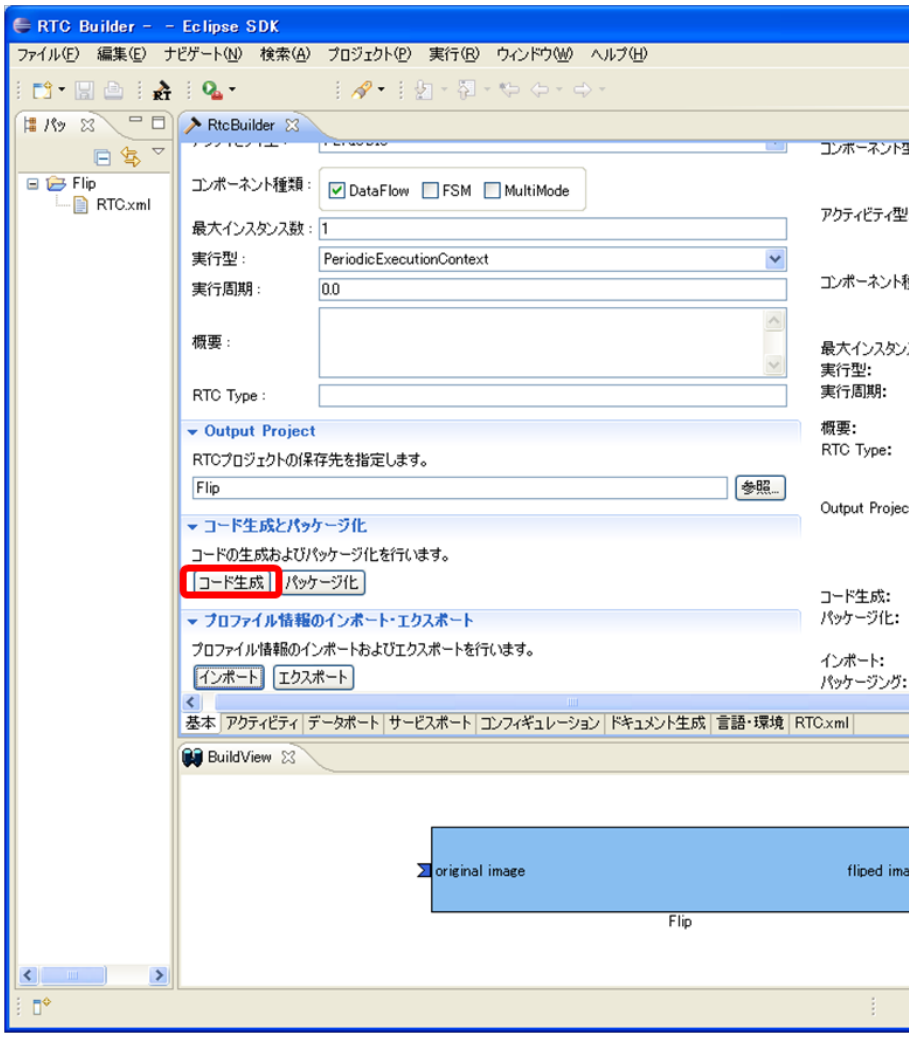

図18. 雛型の生成(Generate)

※ 生成されるコード群は、eclipse起動時に指定したワークスペースフォルダの中に生成されます。現在のワークスペースは、「ファイル(F)」>「ワークスペースの切り替え(W)...」で確認することができます。

仮ビルド

さて、ここまででFlipコンポーネントのソースコードが生成されました。処理の中身は実装されていないので、InPortに画像を入力しても何も出力 されませんが、生成直後のソースコードだけでもコンパイルおよび実行は できます。

※サービスポートとプロバイダを持つコンポーネントの場合、実装を行わないとビルドが通らないものもあります。

では、まずCMakeを利用してビルド環境のConfigureを行います。Linuxで あれば、Flipコンポーネントのソースが生成されたディレクトリで

```
$ cmake .
$ make
```

とすれば、Configureおよびビルドが完了するはずです。

Windowsの場合はGUIを利用してConfigureしてみます。スタートメニューなどから **CMake (cmake-gui)** を起動します。

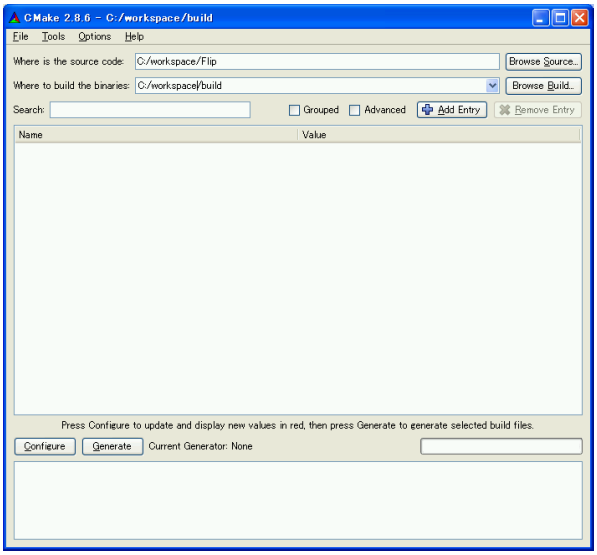

図**19. CMake GUI**の起動とディレクトリの指定

画面上部に以下のようなテキストボックスがありますので、それぞれソー スコードの場所 (CMakeList.txtが有る場所) と、ビルドディレクトリを指 定します。

- Where is the soruce code ^ Where to build the binaries

ソースコードの場所はFlipコンポーネントのソースが生成された場所で CMakeList.txtが存在するディレクトリです。デフォルトでは <ワークス ペースディレクトリ>/Flip になります。

また、ビルドディレクトリとは、ビルドするためのプロジェクトファイル やオブジェクトファイル、バイナリを格納する場所のことです。場所は任 意ですが、この場合 <ワークスペースディレクトリ>/Flip/build のよう に分かりやすい名前をつけたFlipのサブディレクトリを指定することをお 勧めします。

指定したら、下のConfigureボタンを押します。すると図20のようなダイ アログが表示されますので、生成したいプロジェクトの種類を指定します。 今回はVisual Studio 10 とします。VS8やVS9を利用している方はそれぞ れ読み替えてください。

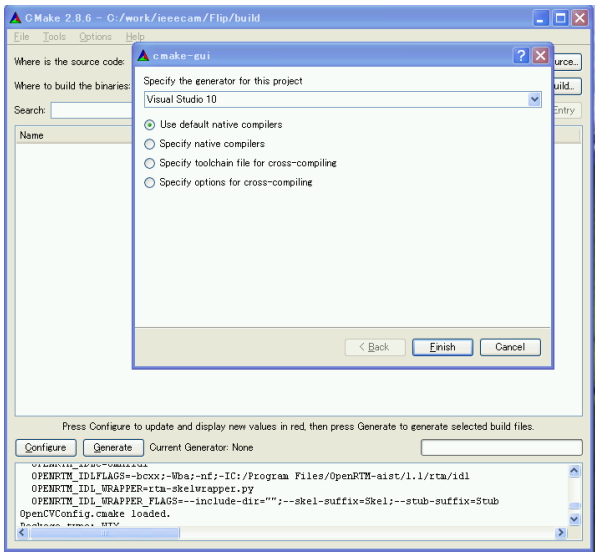

図**20.** 生成するプロジェクトの種類の指定

ダイアログでFinishを押すとConfigureが始まります。問題がなければ下部のログウインドウに**Configuring done**と出力されますので、続けて **Generate** ボタンを押します。**Generating done**と出ればプロジェクトファイル・ソリューションファイル等の出力が完了します。

なお、CMakeはConfigureの段階でキャッシュファイルを生成しますので、トラブルなどで設定を変更したり環境を変更した場合は [File]-[Delete Cache] でキャッシュを削除してConfigureからやり直してください。

次に先ほど指定したbuildディレクトリの中のFlip.slnをダブルクリックしてVisual Studio 2010を起動します。

起動後、ソリューションエクスプローラーの **ALL_BUILD** を右クリッ クししビルドを選択してビルドします。特に問題がなければ正常にビ ルドが終了します。

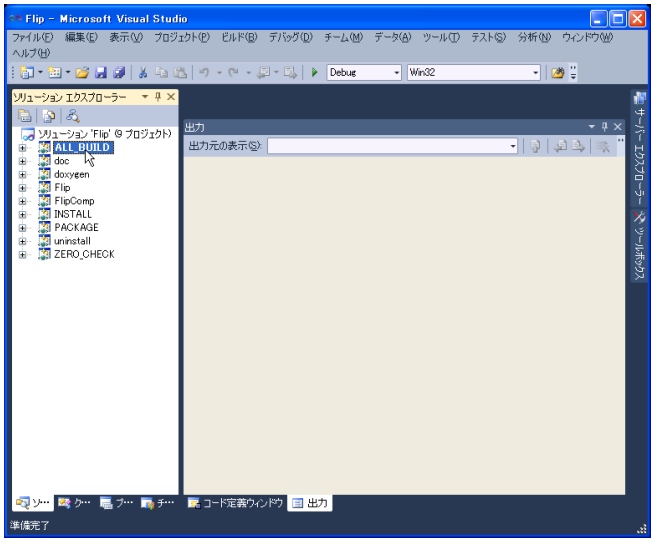

図**21.** ビルド画面

ヘッダ、ソースの編集

アクティビティ処理の実装

Flipコンポーネントでは、InPortから受け取った画像を画像保存用バッファ に保存し、その保存した画像をOpenCVのcvFlip()関数にて変換します。そ の後、変換された画像をOutPortから送信します。

onActivated(),onExecute(),onDeactivated()での処理内容を図22に示します。

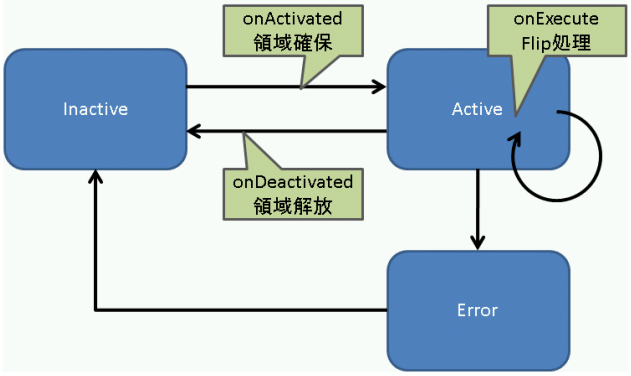

図**22.** アクティビティ処理の概要

onExecute()での処理を図23に示します。

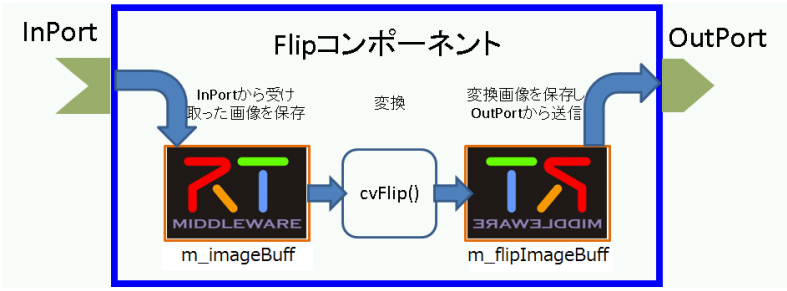

図**23. onExucete()**での処理内容

ヘッダファイル **(Flip.h)** の編集

OpenCVのライブラリを使用するため、OpenCVのインクルードファイルをイ ンクルードします。

```
//OpenCV用インクルードファイルのインクルード
#include<cv.h>
#include<cxcore.h>
#include<highgui.h>
```

このcvFlipコンポーネントでは、画像領域の確保、Flip処理、確保した画 像領域の解放のそ

れぞれの処理を行います。これらの処理は、それぞれ
onActivated(),onDeactivated(),onExecute() のコールバック関数にて行 います。

```
/***
 *
 * The activated action (Active state entry action)
 * former rtc_active_entry()
 *
 * @param ec_id target ExecutionContext Id
 *
 * @return RTC::ReturnCode_t
 *
 *
 */
virtual RTC::ReturnCode_t onActivated(RTC::UniqueId

/***
 *
 * The deactivated action (Active state exit action)
 * former rtc_active_exit()
 *
 * @param ec_id target ExecutionContext Id
 *
 * @return RTC::ReturnCode_t
 *
 *
 */
virtual RTC::ReturnCode_t onDeactivated(RTC::Uniquel

/***
 *
 * The execution action that is invoked periodically
 * former rtc_active_do()
 *
 * @param ec_id target ExecutionContext Id
 *
 * @return RTC::ReturnCode_t
 *
 *
 */
virtual RTC::ReturnCode_t onExecute(RTC::UniqueId ec
```

反転した画像の保存用にメンバー変数を追加します。

```
IplImage* m_imageBuff;
```

```
  IplImage* m_flipImageBuff;
```

ソースファイル **(Flip.cpp)** の編集

下記のように、onActivated(),onDeactivated(),onExecute()を実装します。

```
RTC::ReturnCode_t Flip::onActivated(RTC::UniqueId ec_i
{
  // イメージ用メモリの初期化
  m_imageBuff = NULL;
  m_flipImageBuff = NULL;

  // OutPortの画面サイズの初期化
  m_flippedImage.width = 0;
  m_flippedImage.height = 0;

  return RTC::RTC_OK;
}


RTC::ReturnCode_t Flip::onDeactivated(RTC::UniqueId ec
{
  if(m_imageBuff != NULL)
  {
    // イメージ用メモリの解放
    cvReleaseImage(&m_imageBuff);
    cvReleaseImage(&m_flipImageBuff);
  }

  return RTC::RTC_OK;
}


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
    if( m_originalImage.width != m_flippedImage.width
      {
    m_flippedImage.width = m_originalImage.width;
    m_flippedImage.height = m_originalImage.height;
```

```
    // InPortのイメージサイズが変更された場合
    if(m_imageBuff != NULL)
      {
        cvReleaseImage(&m_imageBuff);
        cvReleaseImage(&m_flipImageBuff);
      }

    // イメージ用メモリの確保
    m_imageBuff = cvCreateImage(cvSize(m_originalImage
    m_flipImageBuff = cvCreateImage(cvSize(m_originall
      }

    // InPortの画像データをIplImageのimageDataにコピー
    memcpy(m_imageBuff->imageData,(void *)&(m_original

    // InPortからの画像データを反転する。 m_flipMode 0: X軸
    cvFlip(m_imageBuff, m_flipImageBuff, m_flipMode);

    // 画像データのサイズ取得
    int len = m_flipImageBuff->nChannels * m_flipImage
    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_flipI

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

  return RTC::RTC_OK;
}
```

CMakeによるビルドに必要なファイルの生成

CMakeList.txt の編集

このコンポーネントではOpenCVを利用していますので、OpenCVのヘッダの インクルードパス、ライブラリやライブラリサーチパスを与えてやる必要 が有ります。幸いOpenCVはCMakeに対応しており、以下の4行を追加するだ けでOpenCVのライブラリがリンクされ使えるようになります。

```
# check doxygen installed
find_package(Doxygen)
if(DOXYGEN_FOUND STREQUAL "NO")
    message(FATAL_ERROR "Doxygen not found.")
endif()
```

```
# 以下の5行をCMakeList.txt に加える
cmake_policy(SET CMP0015 NEW)
find_package(OpenCV REQUIRED)
list(APPEND INCLUDE_DIRS ${OpenCV_INCLUDE_DIRS})
list(APPEND LIBRARY_DIRS ${OpenCV_LIB_DIR})
list(APPEND LIBRARIES ${OpenCV_LIBS})
```

VC++によるビルド

ビルドの実行

CMakeList.txtを編集したので、再度CMake GUIでConfigureおよびGenerateを行います。
CMakeのGenerateが正常に終了した事を確認し、Flip.slnファイルをダブルクリックし、
Visual C++ 2010を起動します。

Visual C++ 2010の起動後、図24のようにし、コンポーネントのビルドを行います。

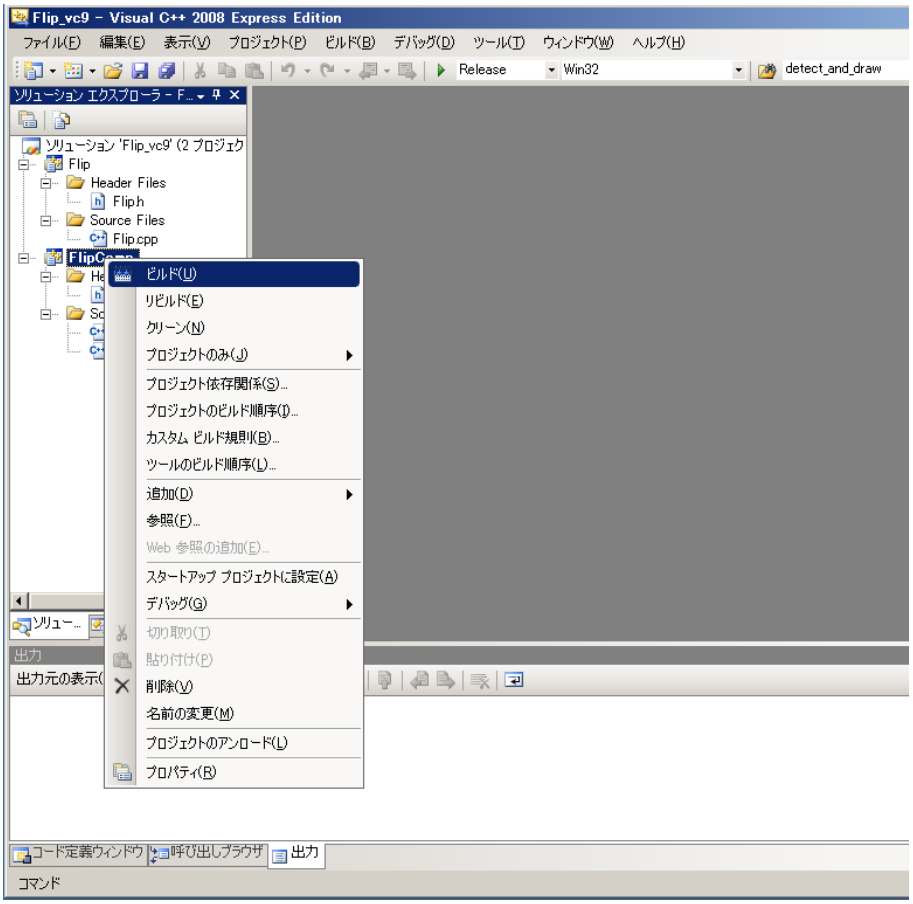

図**24.** ビルドの実行

Flipコンポーネントの動作確認

ここでは、OpenRTM-aist-1.1以降で同梱されるようになったカメラコンポーネント(OpenCVCameraComp, またはDirectShowCamComp)とビューアコンポーネント(CameraViewerComp)を接続し動作確認を行います。

NameServiceの起動

コンポーネントの参照を登録するためのネームサービスを起動します。

[スタート] > [すべてのプログラム(P)] > [OpenRTM-aist] > [C++] > [tools] の順に辿り、「Start Naming Service」をクリックして下さい。

&color(RED){※「Star Naming Service」をクリックしてもomniNamesが起動されない場合は、フルコンピュータ名が14文字以内に設定されているかを確認してください。

rtc.confの作成

RTコンポーネントでは、ネームサーバーのアドレスやネームサーバーへの 登録フォーマットなどの情報をrtc.confというファイルで指定する必要が あります。

下記の内容をrtc.confというファイル名で保存し、Flip\FlipComp\Debug(もしくは、Release)フォルダに置いて下さい。

※ Eclipse起動時にworkspaceをデフォルトのままにしていた場合、Flipフォルダのパスは、

C:\Documents and Settings\<ログインユーザー名>\workspace　となります。

```
corba.nameservers: localhost
naming.formats: %n.rtc
```

Flipコンポーネントの起動

Flipコンポーネントを起動します。

先程rtc.confファイルを置いたフォルダにある、FlipComp.exeファイルを実行して下さい。

カメラコンポーネントとビューアコンポーネントの起動

USBカメラのキャプチャ画像をOutPortから出力するOpenCVCameraComp, または DirectShowCamCompコンポーネントと、InPortで受け取った画像を画面に表示する CameraViewerCompを起動します。

これら2つのコンポーネントは、下記の手順にて起動できます。

[スタート] > [すべてのプログラム(P)] > [OpenRTM-aist] > [components] > [C++] > [examples] > [opencv-rtcs] の順に辿り、「OpenCVCameraComp」と 「CameraViewerComp」をそれぞれクリックして実 行します。

OpenCVCameraCompではうまくカメラを認識しない場合が有ります。その場 合は DirectShowCamCompを使用してみてください。

コンポーネントの接続

図25のように、RTSystemEditorにて USBCameraAqcuireComp,Flip,USBCameraMonitorCompコンポーネントを接続 します。

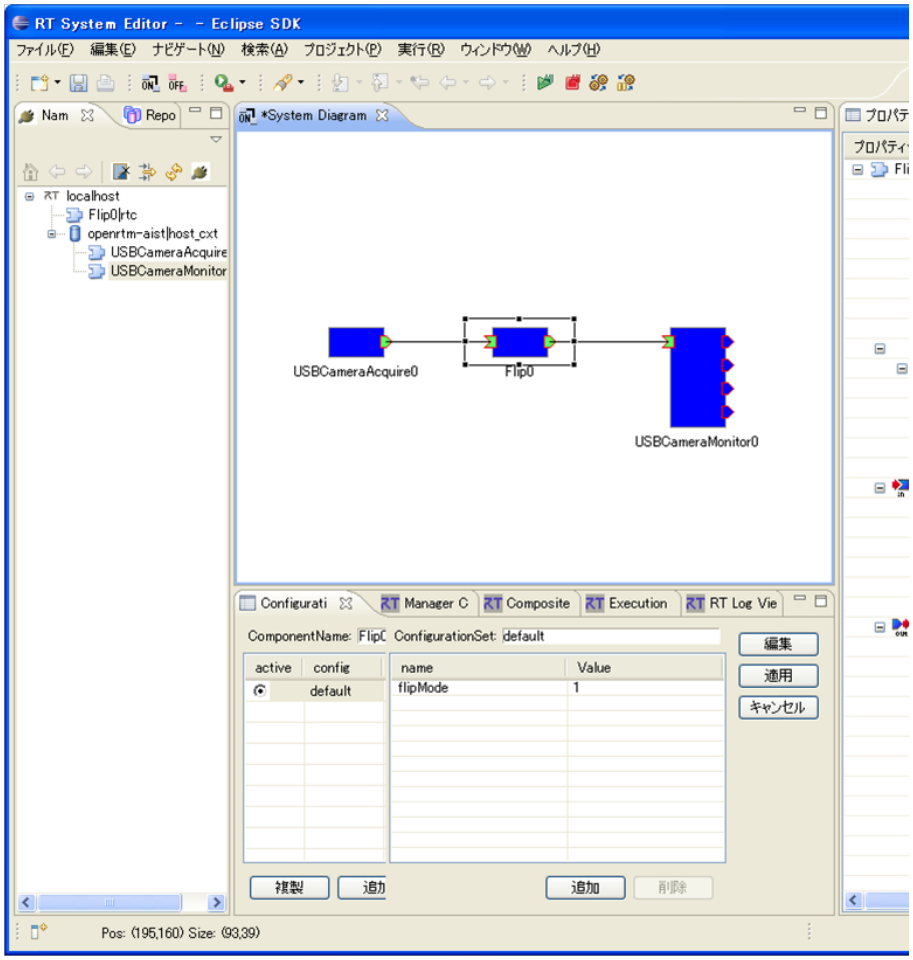

図**25.** コンポーネントの接続

コンポーネントの**Activate**

RTSystemEditorの上部にあります「ALL」というアイコンをクリックし、全てのコンポーネントをアクティブ化します。正常にアクティベートされ た場合、図26のように黄緑色でコンポーネントが表示されます。

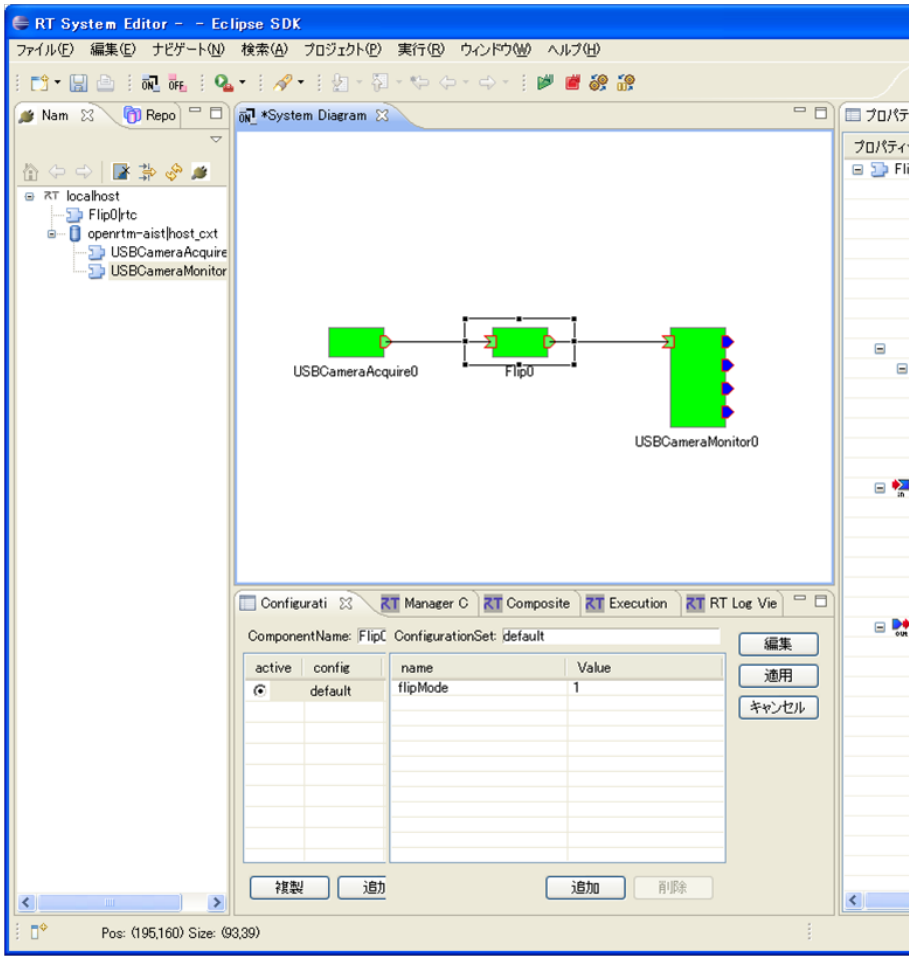

図**26.** コンポーネントのアクティブ化

動作確認

図27のようにコンフィギュレーションビューにてコンフィギュレーションを変更することができます。

Flipコンポーネントのコンフィギュレーションパラメータ「flipMode」を「0」や「-1」などに変更し、画像の反転が行われるかを確認して下さい。

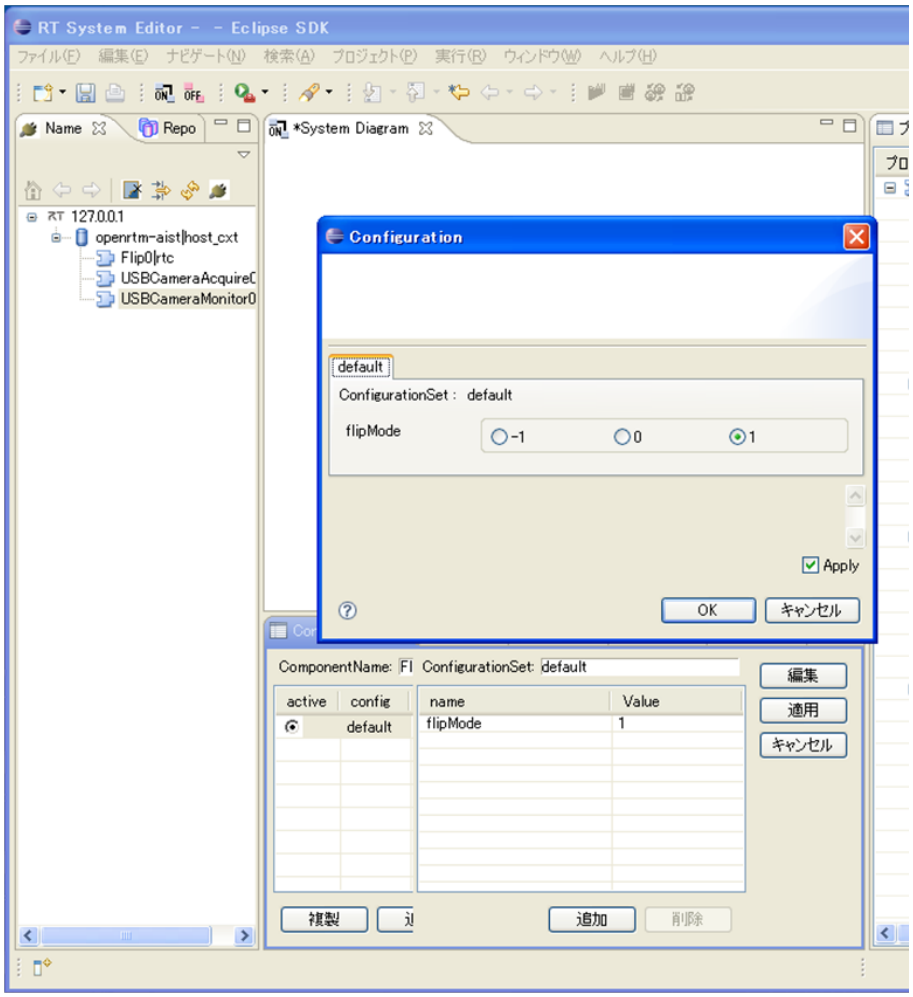

図27. コンフィギュレーションパラメータの変更

Flipコンポーネントの全ソース

Flipコンポーネントソースファイル (Flip.cpp)

```
// -*- C++ -*-
/*!
 * @file  Flip.cpp
 * @brief Flip image component
 * @date $Date$
 *
 * $Id$
 */

#include "Flip.h"

// Module specification
```

```
static const char* flip_spec[] =
  {
    "implementation_id", "Flip",
    "type_name",         "Flip",
    "description",       "Flip image component",
    "version",           "1.0.0",
    "vendor",            "AIST",
    "category",          "Category",
    "activity_type",     "PERIODIC",
    "kind",              "DataFlowComponent",
    "max_instance",      "1",
    "language",          "C++",
    "lang_type",         "compile",
    // Configuration variables
    "conf.default.flipMode", "1",
    // Widget
    "conf.__widget__.flipMode", "radio",
    // Constraints
    "conf.__constraints__.flip_mode", "(-1,0,1)",
    ""
  };

/*!
 * @brief constructor
 * @param manager Maneger Object
 */
Flip::Flip(RTC::Manager* manager)
  : RTC::DataFlowComponentBase(manager),
    m_originalImageIn("originalImage", m_originalImage
    m_flippedImageOut("flippedImage", m_flippedImage)
{
}

/*!
 * @brief destructor
 */
Flip::~Flip()
{
}


RTC::ReturnCode_t Flip::onInitialize()
{
  // Registration: InPort/OutPort/Service
  // Set InPort buffers
  addInPort("originalImage", m_originalImageIn);
```

```
  // Set OutPort buffer
  addOutPort("flippedImage", m_flippedImageOut);

  // Bind variables and configuration variable
  bindParameter("flipMode", m_flipMode, "1");

  return RTC::RTC_OK;
}


RTC::ReturnCode_t Flip::onActivated(RTC::UniqueId ec_i
{
  // イメージ用メモリの初期化
  m_imageBuff = NULL;
  m_flipImageBuff = NULL;

  // OutPortの画面サイズの初期化
  m_flippedImage.width = 0;
  m_flippedImage.height = 0;

  return RTC::RTC_OK;
}


RTC::ReturnCode_t Flip::onDeactivated(RTC::UniqueId ec
{
  if(m_imageBuff != NULL)
  {
    // イメージ用メモリの解放
    cvReleaseImage(&m_imageBuff);
    cvReleaseImage(&m_flipImageBuff);
  }

  return RTC::RTC_OK;
}


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
```

```
    if( m_originalImage.width != m_flippedImage.width
      {
    m_flippedImage.width = m_originalImage.width;
    m_flippedImage.height = m_originalImage.height;

    // InPortのイメージサイズが変更された場合
    if(m_imageBuff != NULL)
      {
        cvReleaseImage(&m_imageBuff);
        cvReleaseImage(&m_flipImageBuff);
      }

    // イメージ用メモリの確保
    m_imageBuff = cvCreateImage(cvSize(m_originalImage
    m_flipImageBuff = cvCreateImage(cvSize(m_originali
      }

    // InPortの画像データをIplImageのimageDataにコピー
    memcpy(m_imageBuff->imageData,(void *)&(m_original

    // InPortからの画像データを反転する。 m_flipMode 0: X軸
    cvFlip(m_imageBuff, m_flipImageBuff, m_flipMode);

    // 画像データのサイズ取得
    int len = m_flipImageBuff->nChannels * m_flipImage
    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_flipI

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

  return RTC::RTC_OK;
}


extern "C"
{

  void FlipInit(RTC::Manager* manager)
  {
    coil::Properties profile(flip_spec);
    manager->registerFactory(profile,
                             RTC::Create<Flip>,
```

```
                                    RTC::Delete<Flip>);
  }

};
```

Flipコンポーネントのヘッダファイル **(Flip.h)**

```
// -*- C++ -*-
/*!
 * @file  Flip.h
 * @brief Flip image component
 * @date  $Date$
 *
 * $Id$
 */

#ifndef FLIP_H
#define FLIP_H

#include <rtm/Manager.h>
#include <rtm/DataFlowComponentBase.h>
#include <rtm/CorbaPort.h>
#include <rtm/DataInPort.h>
#include <rtm/DataOutPort.h>
#include <rtm/idl/BasicDataTypeSkel.h>
#include <rtm/idl/ExtendedDataTypesSkel.h>
#include <rtm/idl/InterfaceDataTypesSkel.h>

//OpenCV用インクルードファイルのインクルード
#include<cv.h>
#include<cxcore.h>
#include<highgui.h>

using namespace RTC;

/*!
 * @class Flip
 * @brief Flip image component
 *
 */
class Flip
  : public RTC::DataFlowComponentBase
{
 public:
  /*!
```

```
   * @brief constructor
   * @param manager Maneger Object
   */
  Flip(RTC::Manager* manager);

  /*!
   * @brief destructor
   */
  ~Flip();

  /***
   *
   * The initialize action (on CREATED->ALIVE transiti
   * formaer rtc_init_entry()
   *
   * @return RTC::ReturnCode_t
   *
   *
   */
   virtual RTC::ReturnCode_t onInitialize();

  /***
   *
   * The activated action (Active state entry action)
   * former rtc_active_entry()
   *
   * @param ec_id target ExecutionContext Id
   *
   * @return RTC::ReturnCode_t
   *
   *
   */
   virtual RTC::ReturnCode_t onActivated(RTC::UniqueId

  /***
   *
   * The deactivated action (Active state exit action)
   * former rtc_active_exit()
   *
   * @param ec_id target ExecutionContext Id
   *
   * @return RTC::ReturnCode_t
   *
   *
   */
   virtual RTC::ReturnCode_t onDeactivated(RTC::Unique
```

```
  /***
   *
   * The execution action that is invoked periodically
   * former rtc_active_do()
   *
   * @param ec_id target ExecutionContext Id
   *
   * @return RTC::ReturnCode_t
   *
   *
   */
   virtual RTC::ReturnCode_t onExecute(RTC::UniqueId e

 protected:
  // Configuration variable declaration
  /*!
   *
   * - Name:  flipMode
   * - DefaultValue: 1
   */
  int m_flipMode;

  // DataInPort declaration
  CameraImage m_originalImage;

  /*!
   */
  InPort<CameraImage> m_originalImageIn;

  // DataOutPort declaration
  CameraImage m_flippedImage;

  /*!
   */
  OutPort<CameraImage> m_flippedImageOut;

 private:
  // 処理画像用バッファ
  IplImage* m_imageBuff;
  IplImage* m_flipImageBuff;
};


extern "C"
{
```

```
  DLL_EXPORT void FlipInit(RTC::Manager* manager);
};

#endif // FLIP_H
```

Flipコンポーネントの全ソースコード

Flipコンポーネントの全ソースコードを以下に添付します。

Flip.zip