Sherlock 7 Technical Resource

Teledyne DALSA Industrial Products (IPD)

Ben Dawson
Document Revision: 13 August 2012

Using Images in Sherlock Formulas

Introduction

This document describes how to write a Sherlock Formula that processes images. While this is possible, it is not recommended. We first review what a Formula does, then why you shouldn't process images with a Formula, and then tell you how to do it, if you must. For details and background, see the "plug-in manual" in the Sherlock distribution: *CreatingSherlockProcessingPlugins.pdf*

What is a Formula?

A Sherlock processor uses information macros to "tell" Sherlock's Image Processing Engine (IPE) the type of processor:

- Preprocessors have image description (PLUGIN_PEEKS and PLUGIN_IMGTYPE) macros but do not have Input or Output macros. Preprocessors may have Parameters.
- Algorithms have an image description ((PLUGIN_PEEKS and PLUGIN_IMGTYPE) macros and must have at least one Output macro. Algorithms may have Parameter macros but not Input macros.
- Formulas must have no image descriptions – the definition macros PLUGIN_PEEKS and PLUGIN_IMGTYPE **must not** be used. Formulas may have Parameter, Input, and Output macros

This table summarizes the rules for what preprocessors, algorithms and formulas can and can't have for image description, Parameter, Input, and Output macros.

	Image Description	**Parameter Macros**	**Input Macros**	**Output Macros**
Preprocessors	YES	May have	NO	NO
Algorithms	YES	May have	NO	YES
Formulas	NO	May have	May have	May have

Where NO means the processor must not have that kind of information macro, YES means it must have that kind of macro, and "May Have" indicates an optional kind of macro.

Sherlock's Formulas are meant to do calculations on Algorithm results, such as adding an array of numbers, or for input-output (I/O) operations such as opening a disk file.

Why You Shouldn't Process Images with a Formula

Formulas can take Input images and Output processed images. However, we did not create the Formula type of processor to do this, so we do not recommend doing this.

Because Formulas have no image description macros:

- There is no concept of an ROI type or extracted or masked images inside a Formula. The Input and Output macros can only pass rectangular images of class CIpeImage.
- There is no built-in protection against using an image type that the Formula can't handle. For example, if a Formula processor only handles 8-bit, monochrome images (MONO8 type), passing it a color image, say an RGB32 image, may cause it to malfunction or crash. Preprocessors and Algorithms only handle the image types you specify, thus preventing this kind of malfunction.

Processing Images with a Formula

If you must use images as inputs or outputs to a Formula, here are two possible ways.

First, you can access "global images" in a Formula using (see plug-in manual). You pass the index of the input and / or output global image(s) through INPUT_NUMBER Input macros and then access the global image through that index. The function:

```
CIpeImage* GetImage(int nIndex);
```

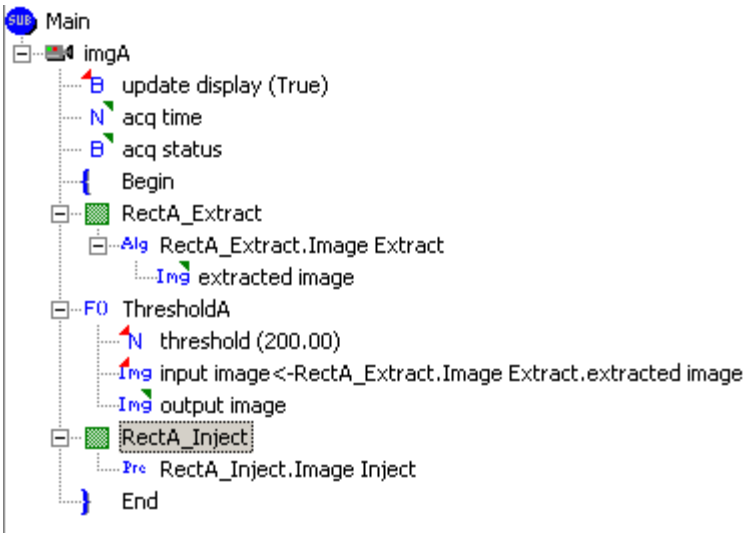
Returns a pointer to the global image specified by index nIndex as a CIpeImage type. You then have to access the member elements of the CIpeImage to determine the image's size, type, location in memory, etc.

The second method is to pass images as arguments to the Formula, and use the Input and Output macros for images. The rest of this document shows you how to do this.

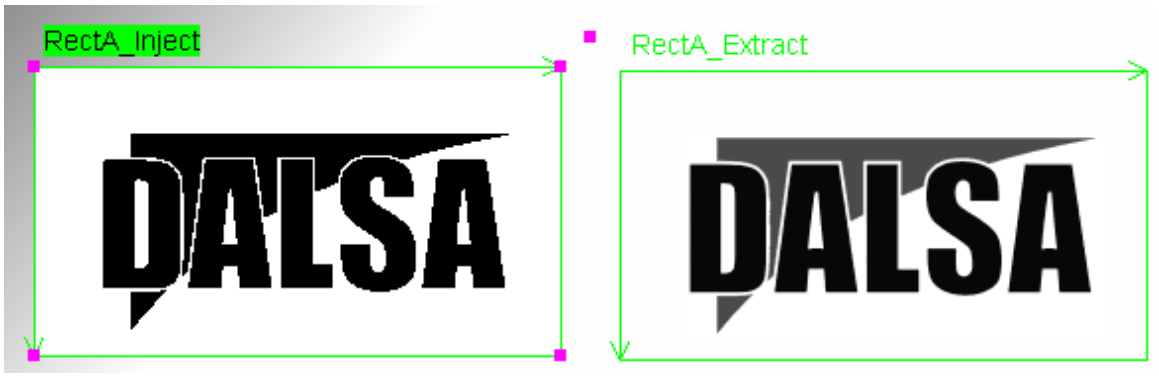
Before you start, try out the 'threshold' formula in the Image folder of the Instruction window. This Formula takes an input image, thresholds it (output pixel values are 0 below threshold and 255 at or above the threshold), and outputs the resulting image. Here is how to do try this function out:

Create a Program (Investigation) with a rectangular ROI named RectA_Extract. Then add the Threshold formula below this ROI. It will be called ThresholdA. Note that the image input to ThresholdA is marked as undefined. Copy and past RectA_Extract, put this copy below the "threshold" formula and name it RectA_Inject. Move RectA_Inject so that the ROI is within the Image Window.

In RectA_Extract, add the "Image Extract" Algorithm. This gets the pixels from the ROI into a Sherlock image. Drag the "extracted image" into ThresholdA's "input image". This connects the image extracted from RectA_Extract to ThresholdA's "input image". In RectA_Inject, add the "Image Inject" Preprocessor. Open the parameters for "image inject" and select ThresholdA's "output image". This will show as "Handle" in the parameters, indicating that we are passing an image by a handle, rather than copying images. Your investigation should look like:



Run this investigation. RectA_Extract extracts the image from the ROI, ThresholdA threshold it and RectA_Inject takes the "output image" from ThresholdA and injects it into its ROI's processing pipeline:



The extracted (input) image is on the right, and the injected (output) result of ThresholdA is on the left.

The next pages show the code for the 'threshold' Formula. Use this code to learn about passing images in and out of Formulas and as a "template" for your own code.

This is the include file:

```
// Image.h = Formulas that process images (not common or recommended!)
// Copyright (C) 2012, Teledyne DALSA, Inc., ALL RIGHTS RESERVED
//-----------------------------------------------------------------------
#pragma once

#include "..\interfaces\IXInstr.h"
#include "..\IpeImage\IpeImage.h"

////////////////////////////////////////////////////////////////////////
// CThresholdImage = Formula-based image threshold

class CThresholdImage : public IXInstrPlugin
{
IPE_PLUGIN_IMPLEMENT(CThresholdImage)

public:
      CThresholdImage();
      ~CThresholdImage();
      virtual IPE_EXE_RESULT Execute(IXPluginArgs* pArgs);

private:
    CIpeImage* m_pImg;  // This class "owns" the output image
};
```

This is the code file:

```
// Image.cpp =Formulas that process images (not common or recommended!)
// Copyright (C) 2012, Teledyne DALSA, Inc., ALL RIGHTS RESERVED
//-----------------------------------------------------------------------
#include "stdafx.h"

#include "Image.h"
#include "../IpeOs/IpeOs.h"

////////////////////////////////////////////////////////////////////////
// CThreshold Image (Formula)
IPE_PLUGIN_INFO_BEGIN(CThresholdImage)
      // Description macros -- No ROI (PLUGIN_PEEKS) or image
      // type (PLUGIN_IMGTYPE) macros are allowed in a Formula.
      // Formulas can have Parameters, Inputs and / or Outputs.
      PLUGIN_TOOLBOX_NAME("Image")
      PLUGIN_VISIBLE_NAME("Threshold")
      PLUGIN_INTERNAL_NAME("_CThresholdImage_")
      PLUGIN_HELP(460108, NULL)
      PLUGIN_DESCRIPTION("Threshold image")

      // Inputs
      INPUTS_BEGIN
        INPUT_NUMBER(0, _T("threshold"), 128, _T("Threshold, 0..255"))
        INPUT_IMAGE(1, _T("input image"), _T("Monochrome, 8-bit input
image"))
      INPUTS_END

      // Output
      OUTPUTS_BEGIN
        OUTPUT_ENTRY(0, IPE_VAL_IMAGE, _T("output image"))
      OUTPUTS_END
IPE_PLUGIN_INFO_END

CThresholdImage::CThresholdImage()
: m_pImg(0) // Initialize this data member before construction
{
}

CThresholdImage::~CThresholdImage()
{
    IPE_DELETE(m_pImg); // Delete the member image
}

IPE_EXE_RESULT CThresholdImage::Execute(IXPluginArgs* pArgs)
{
    // Inputs:
    CIpeObject** ppInput = pArgs->GetInputs(); // pointer to inputs

    int val = (int) (double) *(ppInput[0]);    // Threshold value
    // Limit val to 0..255
    if (val < 0) val = 0;
    if (val > 255) val = 255;

    const CIpeHandle& hIn = *(ppInput[1]);      // Get handle to image
```

```
    CIpeImage* pInputImg = (CIpeImage*)hIn.handle;// into image pointer

    // If the input image pointer is NULL (0) or has not been
    // created, return OK.
    // This means you haven't connected an image to the input yet.
    if ((pInputImg == 0) || (!pInputImg->IsCreated())) {
        return IPE_EXE_OK;
    }

      // Get the input image information
    SIpeImageInfo InfoIn;                   // Info structure
    pInputImg->GetInfo(&InfoIn);    // Get the image information

    // This Formula works only on a MON08 image
    if (CIpeImage::MONO8 != InfoIn.sImg.eAttr) {
        return IPE_EXE_ERROR; // Only MONO8 images allowed!
    }

    // Output:
    CIpeObject** ppOutputs = pArgs->GetOutputs(); // Get pointer to
outputs
    CIpeHandle& h = *(ppOutputs[0]);// h is the handle for output image

    // Info structure for output image
    SIpeImageInfo InfoOut;

    // If no output image, create it with the same size and type
    // as input image
    if (NULL == m_pImg) {
        m_pImg = new CIpeImage();           // New image structure
        if (NULL == m_pImg) {
            return IPE_EXE_ERROR;           // Out of memory
        }
        m_pImg->Create(pInputImg, true);  // Create pixel storage
    }
    else {  // We have an output image.  Is it the right size and type?
        m_pImg->GetInfo(&InfoOut);          // Get output image info

        // Make output image the same type and size as input image
        if (InfoOut.sImg.eAttr != InfoIn.sImg.eAttr) {
            IPE_DELETE(m_pImg);
            h.handle = NULL;                // In case the 'new' fails
            m_pImg = new CIpeImage();
            if (NULL == m_pImg) {
                return IPE_EXE_ERROR;       // Out of memory
            }
            m_pImg->Create(pInputImg, true);
    }
    // Make output image the same size as input, if the input and
    // output images were the same type to begin with.
        m_pImg->Resize(InfoIn.sImg.nWidth, InfoIn.sImg.nHeight);
    }

    // Always set the output image handle to the internal image pointer
    h.handle = m_pImg;

    // Get the output image information again (might have changed)
```

```
    m_pImg->GetInfo(&InfoOut);

    // Build a threshold LUT (faster on image than an if-else test)
    BYTE lut[256];
    for (int n = 0 ; n < 256 ; n++) {
        lut[n] = (n < val) ? 0 : 255;
    }

    for (int y = 0 ; y < InfoOut.sImg.nHeight ; y++) {
        BYTE* pRowIn  = ( (BYTE*) (InfoIn.sBuf.ppRat[y]) );
        BYTE* pRowOut = ( (BYTE*) (InfoOut.sBuf.ppRat[y]) );
        for (int x = 0 ; x < InfoOut.sImg.nWidth ; x++) {
            pRowOut[x] = lut[pRowIn[x]];
        }
    }

      return IPE_EXE_OK;
}
```