

```
package image_processing;
```

```
/
```

```
*****
```

```
***
```

History:

5/24/2006 - Original Version by Janahan Thevaseelan

7/11/2006 - Updated by RoborealM to be more in sync with other extensions

Notes:

You should be able to run this just using

```
java Extension
```

The default socket extension should be added in RoboRealm. Pressing connect in that module should connect to this running program and swap the blue and red colors. You may want to disable the running image count that is printed to the console to increase processing speed. See *1* below.

Please send comments, suggestions, etc to contact@roborealM.com

Copyright 2006 Cyte.com. All rights reserved.

```
*****
```

```
***/
```

```
import java.net.*;
```

```
import java.io.*;
```

```
import java.io.BufferedInputStream;
```

```
//import java.math;
```

```
//////////////////////////////////// Data
```

```
Structures //////////////////////////////////////
```

```
//hold the image data received from RR
```

```
class ImageDataType
```

```
{
```

```
// hold the name used in RR to identify this module
```

```
public String name;
```

```
// image processed count ... sent back to RR
```

```
public int count;
```

```
// dimensions of the received image
```

```
public int width, height;
```

```
// holds the image data
```

```
public byte pixels[] = null;
```

```

// holds image data length to test if image size changes and buffer becomes
too small
int allocLen;
public int max_val=0;
public int max_x=0;
public int max_y=0;
public int pfc=0;
public double distance=0;
public int laser=0;
};

//////////////////////////////////// Image
processing //////////////////////////////////////

class Extension
{
    // buffer size used when reading in variable data
    public final static int DATA_BUFFER = 1024;

    // the maximum length of a variable name to read
    public final static int MAX_VARNAME_SIZE = 64;

    // the port number to listen on ... needs to match that used in RR
interface
    public final static int SERVER_PORTNUM = 5050;

    // buffer returned by intToByte
    private byte intToByteBuffer[] = new byte[4];

    // Performs the image conversion/processing/etc that you want to
perform
    void ProcessImage(ImageDataType imageData)
    {
        byte pixels[] = imageData.pixels;
        int width = imageData.width;
        int height = imageData.height;

        // sanity check
        if ((pixels==null)||(width==0)||(height==0)) return;

        // you could use imageData.getName() to perform different
analysis in the same
        // program ....

//*****IMAGE PROCESSING*****
        int buffer=0;

```

```

int vertical_gap=40;
int max_val=0;
int max_x=0;
int max_y=0;
int pfc=0;
double distance=0;

if (imageData.count%2==0)
{
    imageData.laser=1;
}
else
{
    imageData.laser=0;
}

//finds pixel with maximum brightness in red image array

for (int i=0; i<height; i++)
{
    for (int j=0; j<width; j++)
    {
        for (int k=0;k<3;k++)
        {
            //Area of image to search for the
            //brightest red pixel
            if (i<(height/2) && ((width/2)-
            (vertical_gap/2)< j && j< (width/2)+(vertical_gap/2)))
            {
                //pixel buffer
                buffer=pixels[j*3+i*3*width
+2];

                //converts from signed byte
                to unsigned byte

                if (buffer<0)
                {
                    buffer=buffer +
                    256;
                }

                //finds maximum values
                if (buffer>max_val)
                {
                    max_val=buffer;
                    max_x=j;
                }
            }
        }
    }
}

```



```

        }
    }
}

//*****

        // note this this is an inplace filter so you don't need an
additional image array
        // but you've now corrupted the original image ...
    }

    ////////////////////////////////// Data
Handling //////////////////////////////////

    int byteToInt(byte data[])
    {
        return (data[0]&255)|((data[1]&255)<<8)|
((data[2]&255)<<16)|((data[3]&255)<<24);
    }

    byte []intToByte(int num)
    {
        intToByteBuffer[0] = (byte)(num&255);
        intToByteBuffer[1] = (byte)((num>>8)&255);
        intToByteBuffer[2] = (byte)((num>>16)&255);
        intToByteBuffer[3] = (byte)((num>>24)&255);
        return intToByteBuffer;
    }

    // returns an error message to RR. This message is displayed in the
"messages" list within
    // the RR Pipe Program interface.
    void ReturnError(BufferedOutputStream writer, String txt) throws
IOException
    {
        writer.write(intToByte(5));
        writer.write("error".getBytes());
        writer.write(intToByte(txt.length()));
        writer.write(txt.getBytes(),0,txt.length());
    }

    // returns a byte string variable to RR. The returned variables can
be used
    // elsewhere in RR for continued processing.
    void ReturnBytesVariable(BufferedOutputStream writer, String name,
byte data[], int len) throws IOException

```

```

    {
        writer.write(intToByte(name.length()));
        writer.write(name.getBytes());

        writer.write(intToByte(len));
        writer.write(data,0,len);
    }

    // returns an int variable to RR
    void ReturnIntVariable(BufferedOutputStream writer, String name,
int num) throws IOException
    {
        String strTmp = new String(Integer.toString(num));

        writer.write(intToByte(name.length()));
        writer.write(name.getBytes());

        writer.write(intToByte(strTmp.length()));
        writer.write(strTmp.getBytes());
    }

    //returns a double variable to RR
    void ReturnDoubleVariable(BufferedOutputStream writer, String name,
double num) throws IOException
    {
        String strTmp = new String(Double.toString(num));

        writer.write(intToByte(name.length()));
        writer.write(name.getBytes());

        writer.write(intToByte(strTmp.length()));
        writer.write(strTmp.getBytes());
    }

    ////////////////////////////////////// Data
Processing //////////////////////////////////////

    int readBytes(byte data[], BufferedInputStream reader, int len)
throws IOException
    {
        int res;
        int index=0;

        while (len>0)
        {
            if ((res=reader.read(data, index, len))<0)

```

```

        return -1;

        index+=res;
        len-=res;
    }
    return index;
}

```

// Parses the variables sent by RR into the appropriate structure.

You can add

// your own processing routines here to handle other variables that may get sent.

```

    int ProcessVariable(BufferedOutputStream writer,
BufferedInputStream reader, ImageDataType imageData, String name, byte
data[], int len) throws IOException
{

```

```

    // determine what we've got
    if (name.equalsIgnoreCase("name"))
    {
        imageData.name = name;
    }
    else
    // determine what we've got
    if (name.equalsIgnoreCase("width"))
    {
        imageData.width = byteToInt(data);
    }
    else
    if (name.equalsIgnoreCase("height"))
    {
        imageData.height = byteToInt(data);
    }
    else
    if (name.equalsIgnoreCase("image"))
    {
        if ((imageData.width==0)||(imageData.height==0))
        {
            ReturnError(writer, "Error - missing image
dimensions before image data!");
            System.out.println("Error - missing image
dimensions before image data!");
            return -1;
        }

```

```

        if (len!=(imageData.width*imageData.height*3))
        {

```



```

        }
    }

    return 1;
}

////////////////////////////////////
Main //////////////////////////////////////

public void handler(String[] args)
{
    // holds the variable name being sent by RR
    String varName = new String();
    // holds the received and perhaps processed image data
    byte varData[] = new byte[DATA_BUFFER];
    // variables data length
    //int varLen;
    // byte array for incoming integer number
    byte number[] = new byte[4];

    ImageDataType imageData = new ImageDataType();

    if (args.length>1)
    {
        System.out.println("Started with \"");
        int i;
        for (i=1;i<args.length;i++)
        {
            if (i>1) System.out.println(" ");
            System.out.println(args[i]);
        }
        System.out.println("\\"\\n");
    }
    else
        System.out.println("Started.\\n");

    ServerSocket server = null;

    try
    {
        server = new ServerSocket(SERVER_PORTNUM);
        System.out.println("RoboRealm Java Socket Server
v0.0.1 Listening On Port "+SERVER_PORTNUM+"....\\n");
    }
    catch (Exception e)
    {

```

```

        System.out.println("Caught exception " +
e.toString() );
        e.printStackTrace( System.out );
        return;
    }

    // only handle one accept thread at a time ...
    while(true)
    {
        try
        {
            System.out.println("Waiting ...\\n");

            Socket client = server.accept();

            System.out.println("Connected.\\n");

            BufferedInputStream bufferedReader = new
BufferedInputStream(client.getInputStream());
            BufferedOutputStream bufferedWriter = new
BufferedOutputStream(client.getOutputStream());

            imageData.count=0;

            int len=0;

            while (true)
            {
                imageData.count++;

                // *1* Comment the next line out
                System.out.println("Processing
"+imageData.count+"\\r");

                while (true)
                {
                    // read in variable length
                    bufferedReader.read(number,
0, 4);

                    len = byteToInt(number);
                    // if length <=0 on the
                    variable name then we're done

                    if (len<=0) break;
                    // read in variable name
                    but if the name is longer than 64 characters

```

```

chars only
byte[len];

bufferedReader.read( varNameChar );
String(varNameChar);

data length
0, 4);

1024 read it in now ..

bufferedReader.read(varData, 0, len);

(ProcessVariable(bufferedWriter, bufferedReader, imageData, varName,
varData, len)<0)

attribute length

back to RoboRealm using stdout.

// then grab the first 64
byte varNameChar[] = new

varName = new

// read in the variable's
bufferedReader.read(number,

len = byteToInt(number);
// if the data is less than

if (len<DATA_BUFFER)
{

}

// handle this variable
if
{
    len=-1;
    break;
}

}

//Done collecting variables.

// termination signal -1 on

if (len==-1) break;

// process image
ProcessImage(imageData);

//Returning variables.

// Write out the processed image

// You can also write back any

```

```

other variables to use in
// other parts of the program.
// The format is the same as the
input.

ReturnBytesVariable(bufferedWriter,
"image", imageData.pixels, imageData.width*imageData.height*3);

//Returned image;

// Send back the count as an
example of how to feed back variables into RoboRealm
ReturnIntVariable(bufferedWriter,
"count", imageData.count);

//Send back information from image
processing algorithm
ReturnIntVariable(bufferedWriter,
"max_val", imageData.max_val);
ReturnIntVariable(bufferedWriter,
"max_x", imageData.max_x);
ReturnIntVariable(bufferedWriter,
"max_y", imageData.max_y);
ReturnIntVariable(bufferedWriter,
"pfc", imageData.pfc);
ReturnDoubleVariable(bufferedWriter, "distance", imageData.distance);
ReturnIntVariable(bufferedWriter,
"laser", imageData.laser);

// write out end of message
bufferedWriter.write(intToByte(0));

// flush all data back to RR
bufferedWriter.flush();
// continue by waiting for next
image request
}

System.out.println("\nDisconnected.\n");

bufferedReader.close();
bufferedWriter.close();
client.close();

if (len==-1) break;

```

```

        }
        catch (Exception e)
        {
            System.out.println("Caught exception " +
e.toString() );
            e.printStackTrace( System.out );
        }
    }

    try
    {
        server.close();
    }
    catch (Exception e)
    {
        System.out.println("Caught exception " +
e.toString() );
        e.printStackTrace( System.out );
        return;
    }

    System.out.println("Exiting\n");
}

// This is where the program first starts
public static void main(String[] args)
{
    Extension ext = new Extension();
    ext.handler(args);
}
}

```