**Image processing \ Frame grabber control APP**

Author: Eli Arad

The architecture of that app will have:

Three C# Class API

One manager that handle all the work

One singleton log class

And a GUI that let’s configure the parameters

With an option to see also the log on screen.

Target architecture:

The CPP DLL determines the project format, x86 or x64.

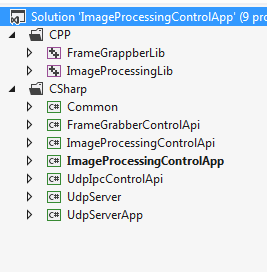
C# can have any CPU but CPP DLL should be deploying as x86 or 64.

Currently I set it to be x86.

Helper libraries:

I am using newtonsoft json .net 4.0 for .net 4

Project structure:



All CP simulation DLL are under CPP folder

All class libraries and application under CSharp.

**First class: Frame Grabber**

Design a CPP frame grabber code that simulate the work of the generating the row data.

The assumption is that the CPP Dll will use a callback and there for will call the C# code instead of the C# will do polling to see if its finished.

To help understand the CPP code , better to look at article:

<https://www.codeproject.com/Tips/318140/How-to-make-a-callback-to-Csharp-from-C-Cplusplus>

I taking the assumption that if we are using callback, this is what we will be see when the actual DLL will be.

The CPP DLL will accept two arguments and a start function.

When the row picture will be ready, it will call the C# API that works with that DLL.

The C# DLL will call that DLL with two parameters and will wait for the row data.

The simulation of the frame grabber in the CPP dll contains several functions:

DLL void DoWork(ProgressCallback progressCallback, int num1, int num2)

DLL void FGClose()

DLL void FGStop()

The main function is DoWork, it send the callback function and two numbers.

The CPP DLL FG simulator create a thread to let the function return immediately.

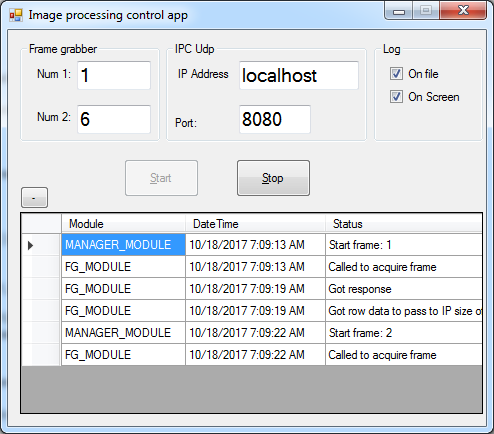
Then it sleep for num2 ( num1 is ignored) and then call back to the c#.

The simulator will eventually will return a buffer ( I will open a jpeg file to make sure I can read it correct in the c# side)

The initial Gui \*( subject to change) will let the user set the num1 and num2 for the frame grabber.

As we can see I am showing the log messages also on screen ( can be disable if one want)

We can see that the thread return immediately and after 6 seconds we got callback function.



As for now the function return just an integer, tomorrow I will return a full row buffer.

For that we can refer to:

<https://stackoverflow.com/questions/13313614/calling-c-dll-with-a-callback-function-that-contains-a-char-from-c-sharp>

Just for the syntax, how to pass a buffer.

The important code in the frame grabber callback is:

ProgressCallback callback =

(Buffer, length) =>

{

byte[] m\_fgBuffer = new byte[length];

Marshal.Copy(Buffer, m\_fgBuffer, 0, length);

return;

};

This code copies the unsigned char array from the CPP to the c# internal buffer.

These complete the first stage of getting the row data from the DLL.

Third class: UDP Server and client

First we need a UDP server to accept our structure.

I am referring to:

https://stackoverflow.com/questions/4844581/how-do-i-make-a-udp-server-in-c

As starting point.

The Server will be wrap as class and that class will be wrap in a separate windows form application,

The server is not part of the project it is just to simulate the UDP server for our client.

To open a port for server listening the application needs user control persmission.

So I added a manifest file with require Administrator.

<requestedExecutionLevel level="requireAdministrator" uiAccess="false" />

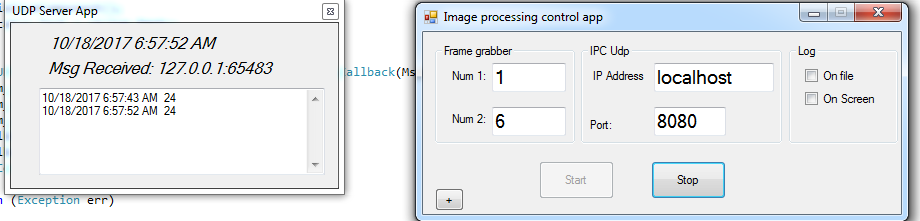
If the computer does not have admin we need to change it and when the app will run it will ask for permission.

The Udp server is only for debugging internally.

Udp Client:

<https://msdn.microsoft.com/en-us/library/system.net.sockets.udpclient(v=vs.110).aspx>

we can see here that the application of the same computer ( right side) is sending udp data structure to the server.



For simulation I create the following data structures:

public struct UDPMessageHeader

{

public ushort opcode;

public ushort size;

}

public struct UPayload

{

public UDPMessageHeader header;

public float data;

public ushort crc;

}

What should be changed in real system:

Interface functions names in FrameGrabberControl.cs

[UnmanagedFunctionPointer(CallingConvention.Cdecl)]

public delegate void ProgressCallback(IntPtr Buffer, int length);

[DllImport("FrameGrappberLib.dll", CallingConvention = CallingConvention.Cdecl)]

public static extern void DoWork([MarshalAs(UnmanagedType.FunctionPtr)] ProgressCallback callbackPointer, int num1, int num2);

[DllImport("FrameGrappberLib.dll")]

public static extern void FGClose();

DllImport("FrameGrappberLib.dll")]

public static extern void FGStop();

Hopefully the callback method in the frame grabber is the one that exists in the real DLL.

In case it is not, it should be easy to change.

Interface functions names in ImageProcessingControl.cs

[DllImport("ImageProcessingLib.dll", CallingConvention = CallingConvention.Cdecl)]

public static extern void IPStartProcess();

[DllImport("ImageProcessingLib.dll", CallingConvention = CallingConvention.Cdecl)]

public static extern int IPGetResult(out float result);

DllImport("ImageProcessingLib.dll", CallingConvention = CallingConvention.Cdecl)]

public static extern void IPClose();

[DllImport("ImageProcessingLib.dll", CallingConvention = CallingConvention.Cdecl)]

public static extern void IPStop();

[DllImport("ImageProcessingLib.dll", CallingConvention = CallingConvention.Cdecl)]

public static extern void IPSetRowData(byte [] data, int size);

Here for now I took an assumption of how that interface is going to be.

Changing should be easy.

In Udp Client\server I created a temporary structure

The structure contains header , payload and crc ( which not in use)

public struct UDPMessageHeader

{

public ushort opcode;

public ushort size;

}

public struct UPayload

{

public UDPMessageHeader header;

public float data;

public ushort crc;

}