

C-sharp windows example

Online version: https://wiki.advacam.cz/wiki/C-sharp_windows_example

Contents

| | |
|-----------------------------------|----|
| Overview & notes | 3 |
| Window | 3 |
| Code | 4 |
| Related | 11 |



Overview & notes

This is C# example of the Windows program with list devices, simple measurement and view of the output data. Timepix3 only.

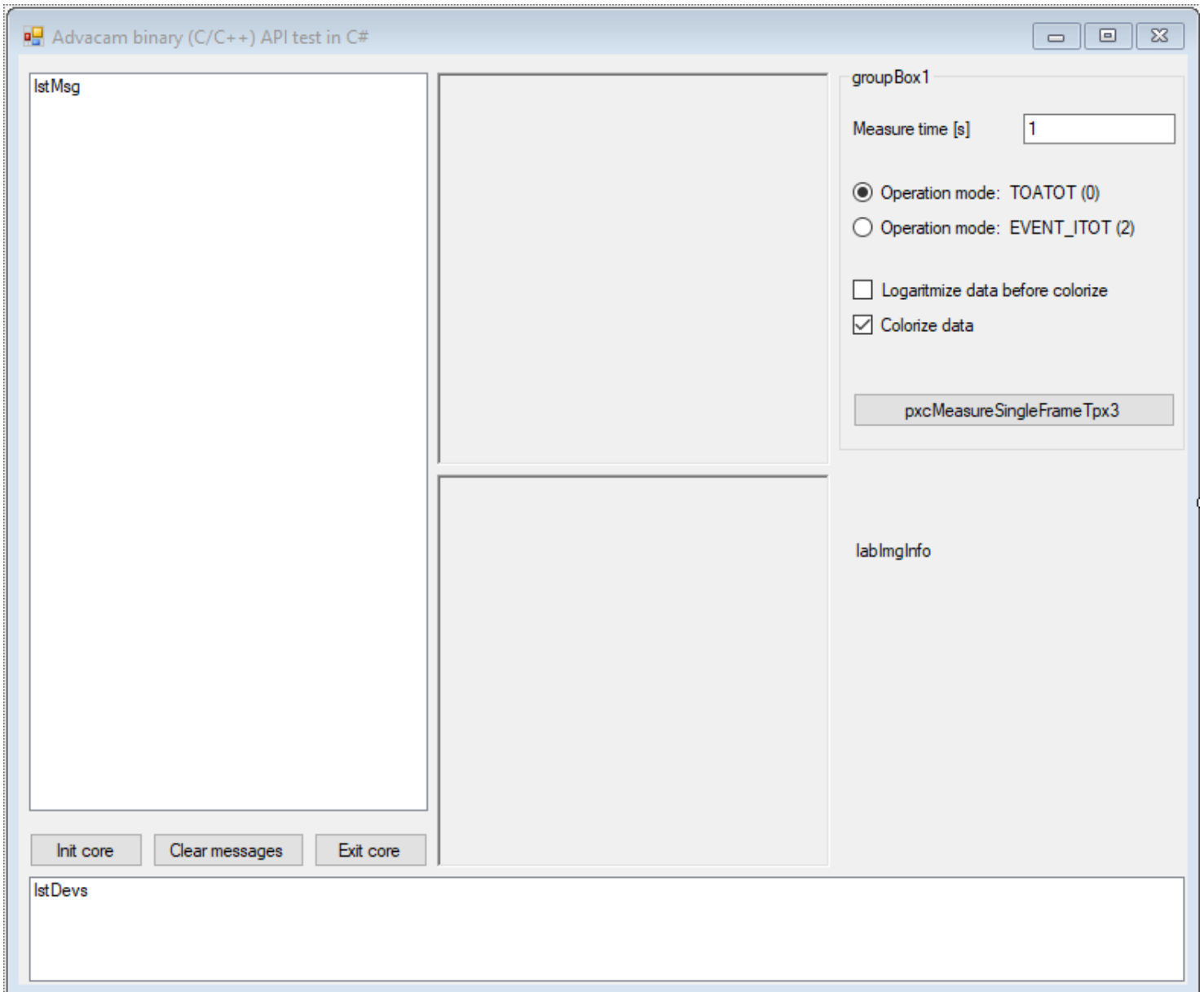
Notes:

- Use the release/64 bit configuration
- The working directory is directory with the exe file. Typically project\bin\Release. Copy pixet.ini and other auxilliary files here.
- In the MS Visual studio 2019, project first not working. You must click Properties, change .NET version to old, save it, change .NET version back to actual and save. Now project can work.

Window

Create window with components:

1. lstMsg - Listbox for messages
2. lstDevs - Listbox for devices list
3. btnInit - Button for init the Pixet core
4. btnClearMsg - Button for clear lstMsg
5. btnExit - Button for exit the Pixet Core
6. pbFrame1 - PictureBox for first output subframe, size equal 256x256 inside, Border Fixed3D
7. pbFrame2 - PictureBox for second output subframe, size equal 256x256 inside, Border Fixed3D
8. txtMeasTime - Textbox for measure time input, default text "1"
9. rbOPM0 - Radiobutton for operation mode 0
10. rbOPM2 - Radiobutton for operation mode 2
11. checkLogView - Checkbox to enable logaritimized view
12. checkColView - Checkbox to enable colored view
13. btnMeasSinTpx3 - Button to run the pxcMeasureSingleFrameTpx3 example measuring
14. lblMsgInfo - Label for informations about measured images



C-sharp windows example - window edit screenshot

Code

```
using System;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Runtime.InteropServices;

namespace WindowsFormsApp1 {
```

```

public partial class Form1 : Form {

    [DllImport("pxcore.dll", CallingConvention = CallingConvention.Cdecl)]
    public static extern int pxcInitialize(Int32 a, UInt64 b);

    [DllImport("pxcore.dll", CallingConvention = CallingConvention.Cdecl)]
    public static extern int pxcExit();

    [DllImport("pxcore.dll", CallingConvention = CallingConvention.Cdecl)]
    public static extern int pxcGetDevicesCount();

    [DllImport("pxcore.dll", CallingConvention = CallingConvention.Cdecl)]
    public static extern int pxcGetDeviceChipID(UInt32 deviceIndex, UInt32 chipIndex,
    StringBuilder chipIDBuffer, UInt32 size);

    [DllImport("pxcore.dll", CallingConvention = CallingConvention.Cdecl)]
    public static extern int pxcGetDeviceName(UInt32 deviceIndex, StringBuilder
    nameBuffer, UInt32 size);

    [DllImport("pxcore.dll", CallingConvention = CallingConvention.Cdecl)]
    public static extern int pxcGetDeviceChipCount(UInt32 deviceIndex);

    [DllImport("pxcore.dll", CallingConvention = CallingConvention.Cdecl)]
    public static extern int pxcGetDeviceInfo(UInt32 deviceIndex, UInt64 devInfo); //
    devInfo is struct, but not used here

    [DllImport("pxcore.dll", CallingConvention = CallingConvention.Cdecl)]
    public static extern int pxcMeasureSingleFrameTpx3(UInt32 deviceIndex, double
    frameTime, [Out] double[] frameToaITot, [Out] UInt16[] frameTotEvent, ref UInt32 size,
    UInt32 trgStg = 0);
    [DllImport("pxcore.dll", CallingConvention = CallingConvention.Cdecl)]
    public static extern int pxcGetDeviceDimensions(UInt32 deviceIndex, ref UInt32
    width, ref UInt32 height);

    [DllImport("pxcore.dll", CallingConvention = CallingConvention.Cdecl)]
    public static extern int pxcSetTimepix3Mode(UInt32 deviceIndex, int mode);

    int devicesCout = 0;
    int[] devTypesN = Enumerable.Repeat(-1, 1000).ToArray();
    UInt32 deviceIdx = uint.MaxValue;
    UInt32 devXsize = 256, devYsize =256;
    UInt32 devPixels = 65536;

    public Form1() {

```



```

        InitializeComponent();
    }

    void msgToList(String msg) { //
=====
        lstMsg.Items.Add(msg);
    }
    void msgToList(String msg, int rc) {
        lstMsg.Items.Add($"{msg:S} rc={rc:D} (0 is OK)");
    }
    void msgToList(String msg1, int n, String msg2) {
        lstMsg.Items.Add($"{msg1:S} {n:D} {msg2:S}");
    }
    void msgToList(String msg1, UInt32 n, String msg2) {
        lstMsg.Items.Add($"{msg1:S} {n:D} {msg2:S}");
    }

    int setPixelRGB(int r, int g, int b) { //
=====
        return b + (g << 8) + (r << 16);
    }

    // value 0-1280 to RGB color or 0-255 to grayscale
    int valToRGB(double val) { //
=====
        int v = (int)val;

        if (!checkColView.Checked) return setPixelRGB(v, v, v);

        if (v < 256) return setPixelRGB(0, 0, v);    // 0   black to blue
        v -= 256;
        if (v < 256) return setPixelRGB(0, v, 255); // 256  blue to blue-green
        v -= 256;
        if (v < 256) return setPixelRGB(0, 255, 255 - v); // 512  blue-green to green
        v -= 256;
        if (v < 256) return setPixelRGB(v, 255, 0); // 768  green to yellow
        v -= 256;
        if (v < 256) return setPixelRGB(255, 255 - v, 0); // 1024  yellow to red
        v -= 256;
        if (v < 256) return setPixelRGB(255, 0, v); // 1280  red to violet
        v -= 256;
        if (v < 256) return setPixelRGB(255, v, 255); // 1536  violet to white
        else return setPixelRGB(255, 255, 255);    // over 1792
    }

```



```

const double cColorizeIniMin = 1e30, cColorizeIniMax = -1e30;
double colorizeMin = cColorizeIniMin, colorizeMax = cColorizeIniMax;
int colorizePixCount = 0;
// normalize data (, logarithmize), convert to color RGB (or grayscale), count non-
zero, count values
// note: logarithmize not usable if data<=0
void colorizeData(double[] dataIn, int[] dataOut) { //
=====
    double colMin=0, colMax=0, pixVal, p;
    colorizeMin = cColorizeIniMin;
    colorizeMax = cColorizeIniMax;
    colorizePixCount = 0;

    for (int n = 0, i = 0; n < devPixels; n++) { // find min, max
        if (dataIn[n] != 0) {
            if (checkLogView.Checked) pixVal = Math.Log(dataIn[n]); else pixVal =
dataIn[n];
            if (dataIn[n] > colorizeMax) { colorizeMax = dataIn[n]; colMax =
pixVal; };
            if (dataIn[n] < colorizeMin) { colorizeMin = dataIn[n]; colMin =
pixVal; };
            colorizePixCount++;
        }
    }

    if (checkColView.Checked) p = 1792.0 / (colMax - colMin);
    else p = 255.0 / (colMax - colMin);
    if (p < 0) p = -p;

    for (int n = 0; n < devPixels; n++) { // colorize data
        if (dataIn[n] != 0) {
            if (checkLogView.Checked) pixVal = Math.Log(dataIn[n]); else pixVal =
dataIn[n];
            dataOut[n] = valToRGB((pixVal - colMin) * p);
        } else {
            dataOut[n] = setPixelRGB(128, 128, 128);
        }
    }
}
void colorizeData(UIInt16[] dataIn, int[] dataOut) { //
=====
    double[] tmpColData = new double[devPixels];
    for (int n = 0; n < devPixels; n++) tmpColData[n] = (double)dataIn[n];
    colorizeData(tmpColData, dataOut);
}

```



```

}

private void btnInit_Click(object sender, EventArgs e) { //
=====
    btnInit.Enabled = false;
    int rc = pxcInitialize(0, 0);
    msgToList("pxcInitialize", rc);
    rc = pxcGetDevicesCount();
    msgToList("pxcGetDevicesCount", rc, "(>=0 devCnt, <0 error)");
    lstDevs.Items.Clear();
    if (rc>0) {
        devicesCout = rc;
        for (UInt32 n=0; n<devicesCout; n++) {
            UInt32 bufLen = 100;
            StringBuilder buff = new StringBuilder((int)bufLen);

            rc = pxcGetDeviceName(n, buff, bufLen);
            String devName;
            if (rc == 0) { devName = buff.ToString(); }
            else { devName = "(failed)"; }

            rc = pxcGetDeviceChipID(n, 0, buff, bufLen);
            String chipID;
            if (rc == 0) { chipID = buff.ToString(); }
            else { chipID = "(failed)"; }

            int chipCnt = pxcGetDeviceChipCount(n);

            String[] devTypes = { "unknown", "TPX", "MPX3", "TPX3", "TPX2" };
            rc = pxcGetDeviceInfo(n, 0);
            String dt;
            if (rc >= 0 && rc < devTypes.Length) {
                dt = devTypes[rc];
                devTypesN[n] = rc;
            } else { dt = "(failed)"; }

            lstDevs.Items.Add($"{n:D} ID:{devName:S}, ID0:{chipID:S},
chipCnt:{chipCnt:D}, type:{dt:S}");
        }
        btnExit.Enabled = true;
    } else {
        msgToList("No devs found");
        btnInit.Enabled = true;
        btnExit.Enabled = false;
    }
}

```




```

        btnMeasSinTpx3.Enabled = false;
    }
}

private void btnClearMsg_Click(object sender, EventArgs e) { //
=====
    lstMsg.Items.Clear();
}

private void Form1_Load(object sender, EventArgs e) {
    lstMsg.Items.Add("Messages");
    lstDevs.Items.Add("Devices list - click the 'Init core' button");
}

private void Form1_FormClosing(object sender, FormClosingEventArgs e) {
    pxcExit();
}

private void btnExit_Click(object sender, EventArgs e) { //
=====
    int rc = pxcExit();
    msgToList("pxcExit", rc);
    lstDevs.Items.Clear();
    deviceIdx = uint.MaxValue;
    devicesCout = 0;

    btnInit.Enabled = true;
    btnExit.Enabled = false;
    btnMeasSinTpx3.Enabled = false;
}

private void lstDevs_SelectedIndexChanged(object sender, EventArgs e) { //
=====
    UInt32 i = (UInt32)lstDevs.SelectedIndex;
    if (devTypesN[i]==3) {
        deviceIdx = i;
        msgToList("Selected device:", deviceIdx, "");
        UInt32 width = 256, height = 256;
        int rc = pxcGetDeviceDimensions(deviceIdx, ref width, ref height);
        if (rc==0) {
            devXsize = width; devYsize = height;
            devPixels = devXsize * devYsize;
            msgToList($"Dev dimm: w:{devXsize:D}, h:{devYsize:D}");
        } else {

```



```

        msgToList("pxcGetDeviceDimensions", rc);
    }
    btnMeasSinTpx3.Enabled = true;
} else {
    deviceIdx = uint.MaxValue;
    btnMeasSinTpx3.Enabled = false;
}
}
}

```

```
private void btnMeasSinTpx3_Click(object sender, EventArgs e) { //
```

```
=====
```

```
    btnMeasSinTpx3.Enabled = false;
```

```
    int opm;
```

```
    if (rbOPM0.Checked) opm = 0; else opm = 2;
```

```
    int rc = pxcSetTimepix3Mode(deviceIdx, opm);
```

```
    msgToList("pxcSetTimepix3Mode", rc);
```

```
    double[] frameToaITot = new double[devPixels];
```

```
    UInt16[] frameTotEvent = new UInt16[devPixels];
```

```
    UInt32 size = devPixels;
```

```
    double t = Convert.ToDouble(txtMeasTime.Text);
```

```
    rc = pxcMeasureSingleFrameTpx3(deviceIdx, t, frameToaITot, frameTotEvent, ref
size, 0);
```

```
    msgToList("pxcMeasureSingleFrameTpx3", rc);
```

```
    btnMeasSinTpx3.Enabled = true;
```

```
    int[] imgData1 = new int[devPixels];
```

```
    int[] imgData2 = new int[devPixels];
```

```
    colorizeData(frameToaITot, imgData1);
```

```
    String colInfo1 = $"min:{colorizeMin:F}, max:{colorizeMax:F}, hit
pixels:{colorizePixCount:D}";
```

```
    colorizeData(frameTotEvent, imgData2);
```

```
    String colInfo2 = $"min:{colorizeMin:F}, max:{colorizeMax:F}, hit
pixels:{colorizePixCount:D}";
```

```
    pbFrame1.SizeMode = PictureBoxSizeMode.StretchImage;
```

```
    pbFrame2.SizeMode = PictureBoxSizeMode.StretchImage;
```

```
    IntPtr imgDataPtr1 = Marshal.AllocHGlobal(Marshal.SizeOf(typeof(int)) *
(int)devPixels);
```

```
    Marshal.Copy(imgData1, 0, imgDataPtr1, (int)devPixels);
```

```
    IntPtr imgDataPtr2 = Marshal.AllocHGlobal(Marshal.SizeOf(typeof(int)) *

```



```
(int)devPixels);
    Marshal.Copy(imgData1, 0, imgDataPtr2, (int)devPixels);

    Bitmap bmp1 = new Bitmap((int)devXsize, (int)devYsize, (int)devXsize * 4,
System.Drawing.Imaging.PixelFormat.Format32bppRgb, imgDataPtr1);
    Bitmap bmp2 = new Bitmap((int)devXsize, (int)devYsize, (int)devXsize * 4,
System.Drawing.Imaging.PixelFormat.Format32bppRgb, imgDataPtr2);

    pbFrame1.Image = bmp1;
    pbFrame2.Image = bmp2;

    labImgInfo.Text = $"Frame1 - ToA/IToT:\n{colInfo1:S}\n\nFrame2 -
ToT/Event:\n{colInfo2:S}";
    }
}
}
```

Related

- [Binary core API](#)
- [Pixet SDK](#)
- [Simple C# commandline example](#)

