Interfacing to C++ in Visual Studio 13

Covers: Interfacing DataRay Camera and Slit Scan Profilers to C++ in Visual Studio 2013 using the DataRay OCX.

Start in the standard software:

- As *Administrator*, install the DataRay software which came with your product.

- Attach the profiler product. Allow the drivers to install.

- Open the DataRay software and select your profiler in the **Device** pull-down menu.

- Learn to use your product in the DataRay software. Then close the software.

Add Visual Studio 13:

We do not claim to be Visual Studio 'experts', however we are able to create new projects in Visual Studio that can control DataRay products. Install Visual Studio 2013 on your computer (earlier versions should work, but exact details will change). You will also need to use MBCS a library you will need to download separately. You can find the download here. MBCS Library. To set up MBCS for compiling and running see (Fig. 5).Download the example from the DataRay website:

- Cameras: Download and unzip: DEAD_LINK.com

- BeamMap2, Beam'R2, ColliMate: TBA

Build and run example:

The example should build and run with no errors (Fig. 1). Not working? Email support@dataray.com or call +1(530)395-2500 with:

- Device name and serial number

- DataRay, Windows and Visual Studio versions which you are using. Only Visual Studio 2013 and later are fully supported. The DataRay OCX still works in VS2006, VS2008, and VS2010, but we are only able to provide limited support.

Overview of OCX:

Your interfacing code communicates with DataRay products through the DataRay OCX. The OCX is an ActiveX component that can be accessed from a variety of Windows based environments. The OCX is automatically generated and registered with the Windows operating system upon installing the DataRay software. Once initialized, the OCX is always running. This means that the camera is still running, even while editing GUI elements in Visual Studio. Do not be alarmed if DataRay OCX GUI elements are active while your program is not running. This is the expected behavior. **Some important notes:**

- Read through this entire document.

- Some prior experience with C++, Windows MFC programming, and Visual Studio is required.

- The OCX is only functional as part of a GUI-based program.

- In the Resource View of VS2013, elements may appear as white boxes or as the actual GUI element they represent.
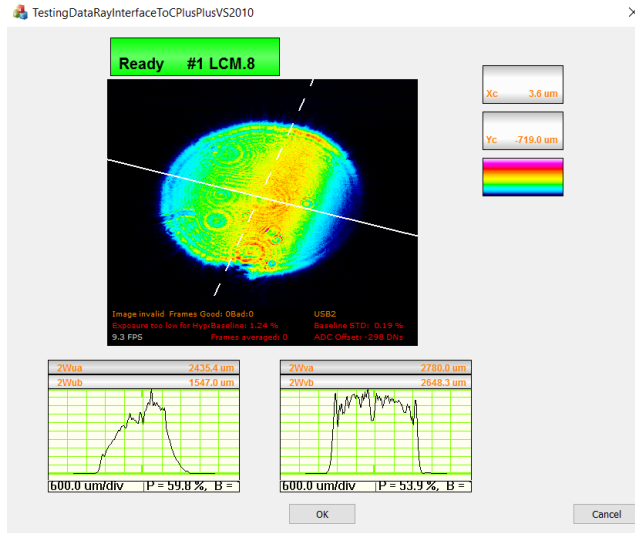
Figure 1: Example application built and running.

Tutorial:

We will show you step-by-step how the example program was created. Follow the instructions included in the series of figures listed below.
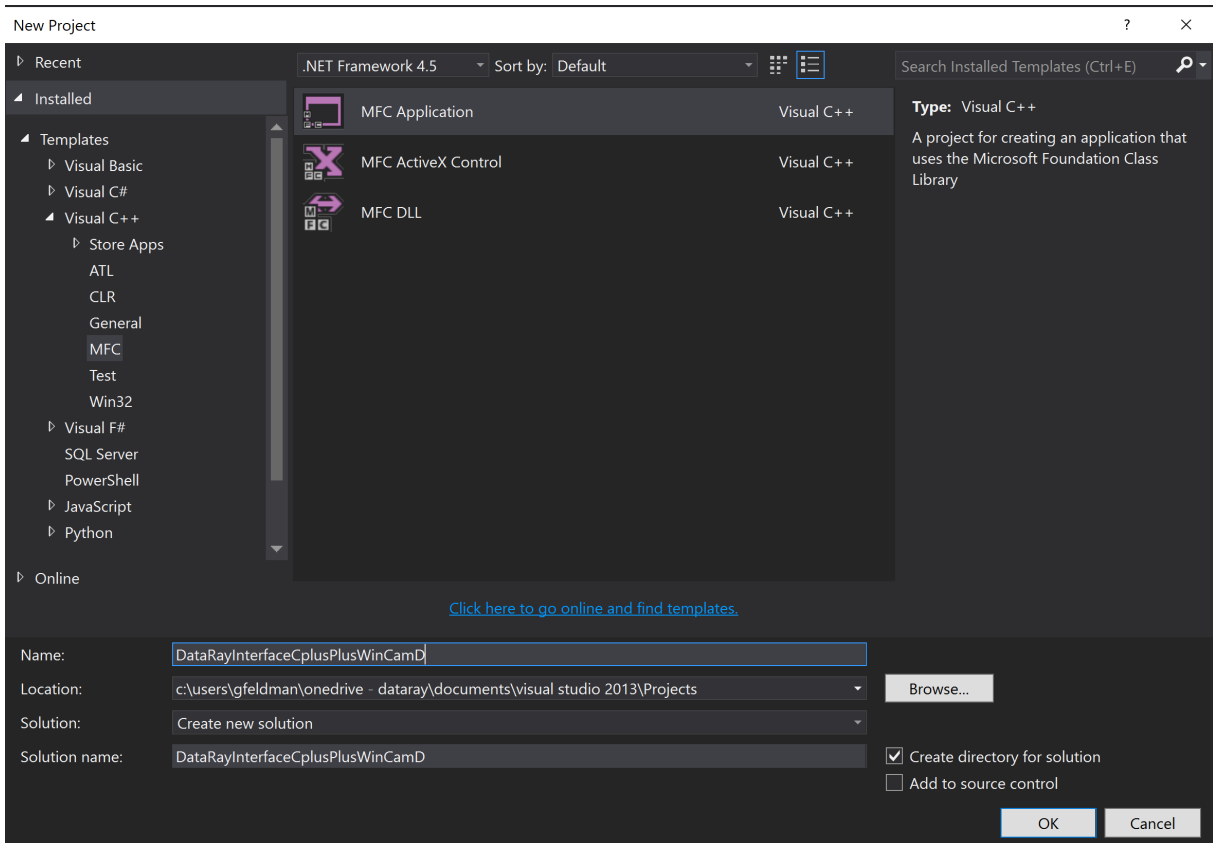
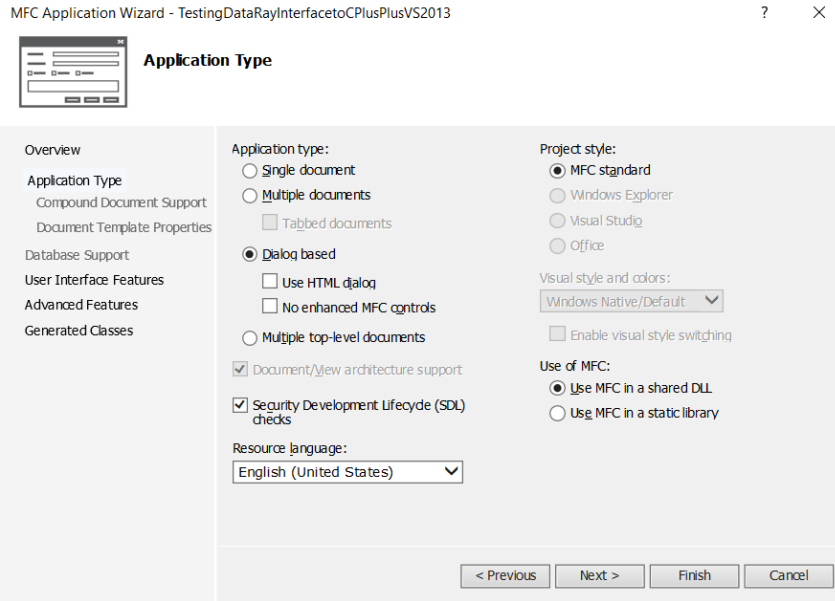Figure 2: First, create a new MFC Application in VS2013.

Figure 3: Select **Dialog based** to simplify things. This is not actually a requirement, but will allow the example to be simple.
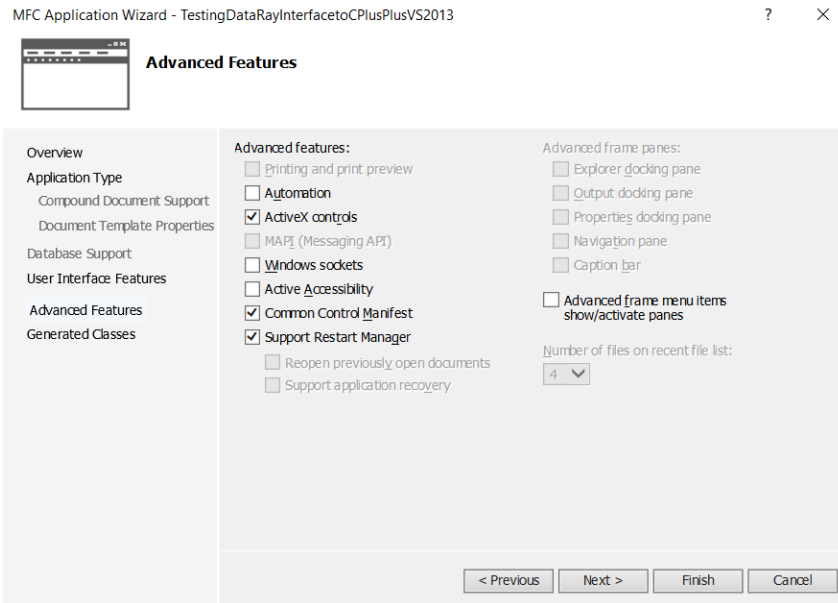

Figure 4: The default values in VS2013 are sufficient for this project. Under the **Advanced Features** tab, verify that **ActiveX controls** are enabled.
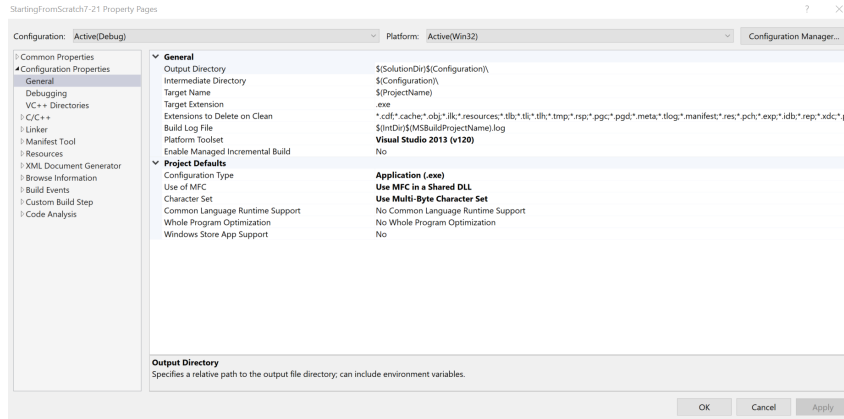
Figure 5: To change from the default Unicode to MBCS you will next go to **View**— ＞**Property Page**. Go to **Configuration Properties**— ＞**General**. Change the character set to **Use Multi-Byte Character Set** by clicking on the **Character Set** option to cycle through.
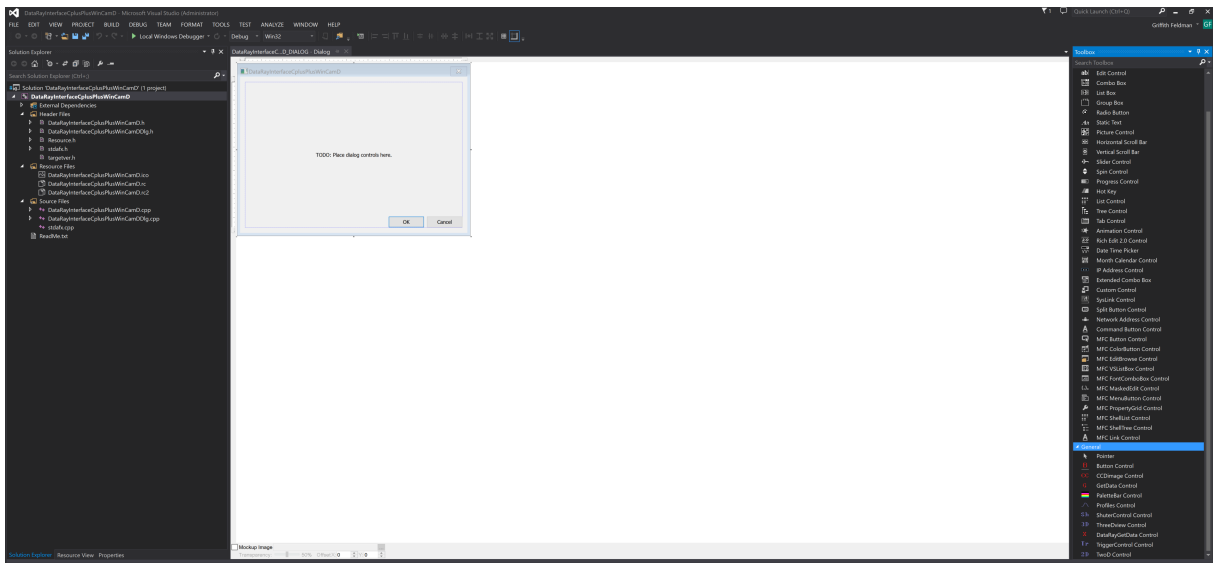

Figure 6: The empty project should look like this.

B	Button Control
CC	CCDimage Control
X	DataRayGetData Control
G	GetData Control
▬	PaletteBar Control
⌒	Profiles Control
Sh	ShuterControl Control
3D	ThreeDview Control
Tr	TriggerControl Control
2D	TwoD Control

Figure 7: Open the **Toolbox.** You should see the following DataRay components. If these components aren't visible, complete the following steps:

1. Select Tools — > Choose Toolbox Items

2. Select COM Components tab

3. Select Browse...

4. Navigate to the your DataRay install directory

5. Select DataRayOcx.ocx

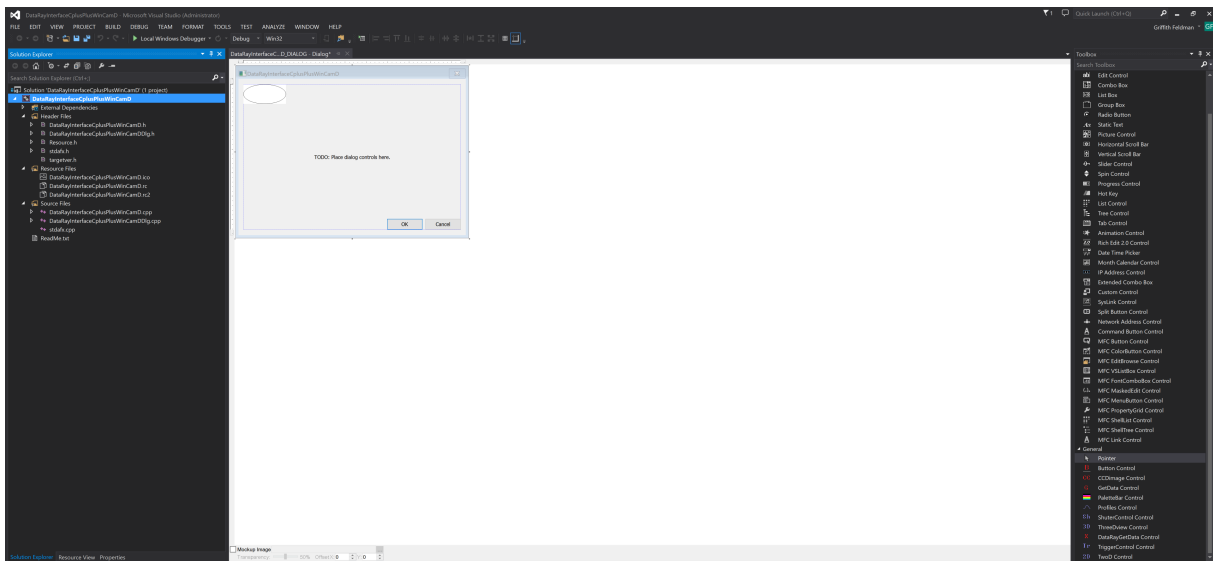6. Your toolbox should now be populated with DataRay Controls.

Figure 8: Now we can begin building the actual program. First drag a **GetData Control (not DataRayGetData Control)** onto the dialog box. This is the only OCX control class required for interfacing to DataRay cameras.
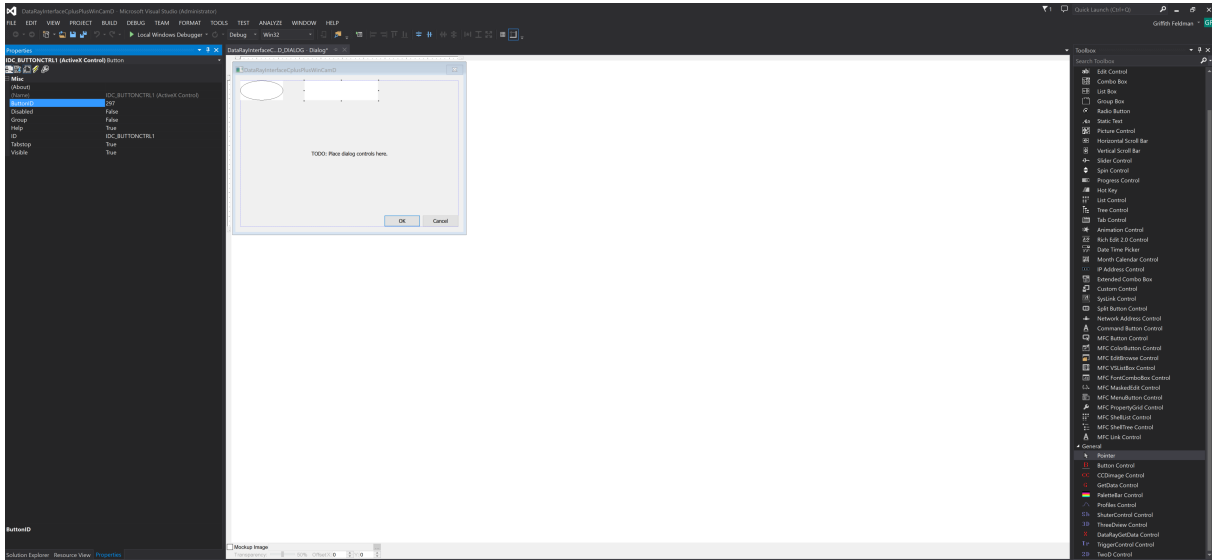
Figure 9: We will also create a **Ready** button and a display for the two-dimensional camera display (known as a **CCDImage**) Drag a **Button Control** to the dialog box. Right click on the button. Select **Edit Control** and then exit the **Edit Control** window. Right click on the button again. This time, select **Properties**. Change the **ButtonID** to **297**.
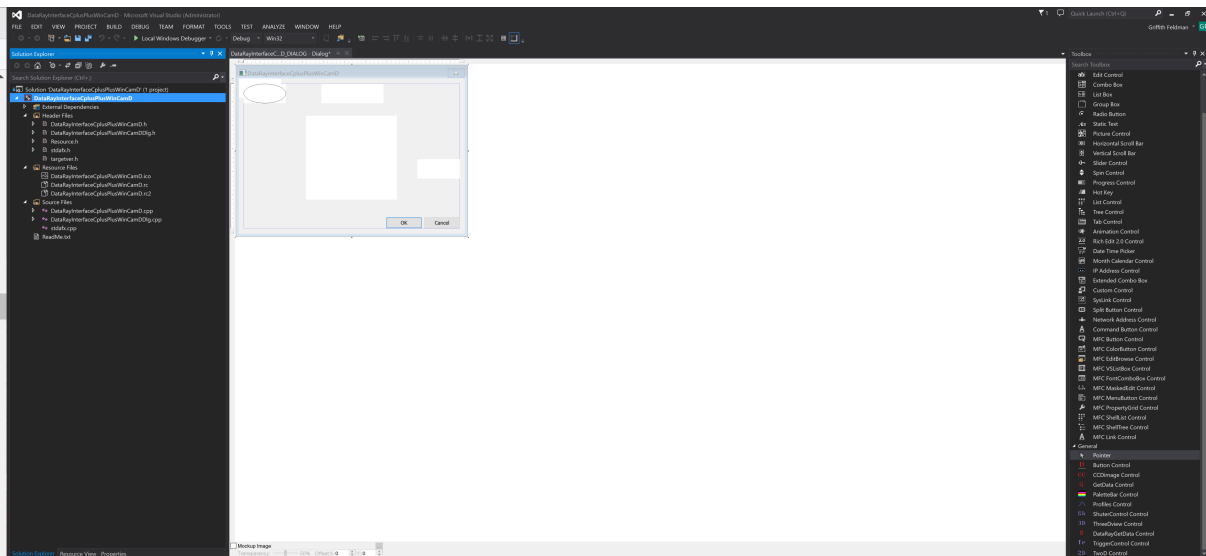

Figure 10: Now drag one **CCDimage Control** from the toolbox to the dialog box. This object will be used to view the image produced by the CCD, so make the object as big as you would like. Finally, drag one **PaletteBar Control** from the toolbox to the dialog box. This completes the basic layout of the dialog box.
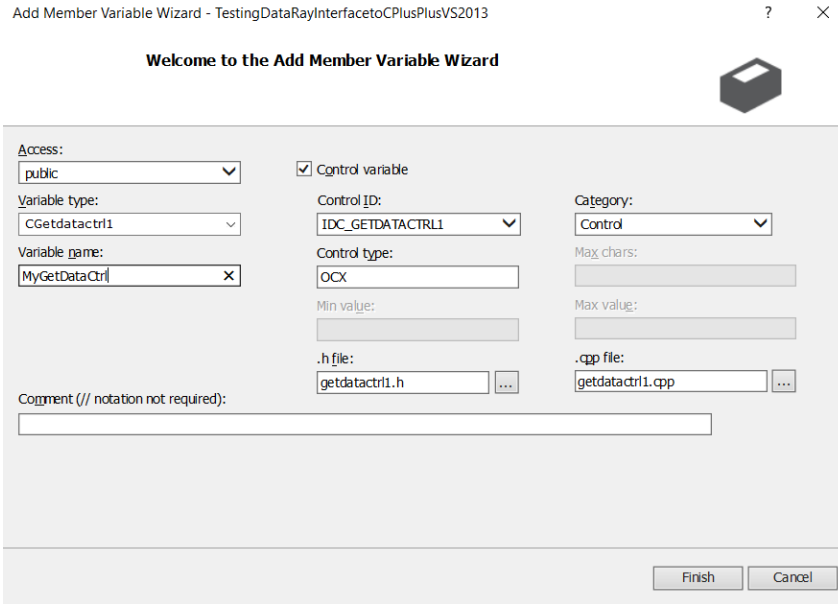
Figure 11: Now we need to add some code to the template. Right-click on the **GetData Control** and select **Add Variable**. Name the variable **MyGetDataCtrl** and make sure **Control variable** is checked. This creates a member control object in your dialog class named **MyGetDataCtrl**. The files getdatactrl1.h and getdatactrl1.cpp are automatically generated and do not need to be modified. This object contains the method StartDriver() that needs to be called to initialize the camera.
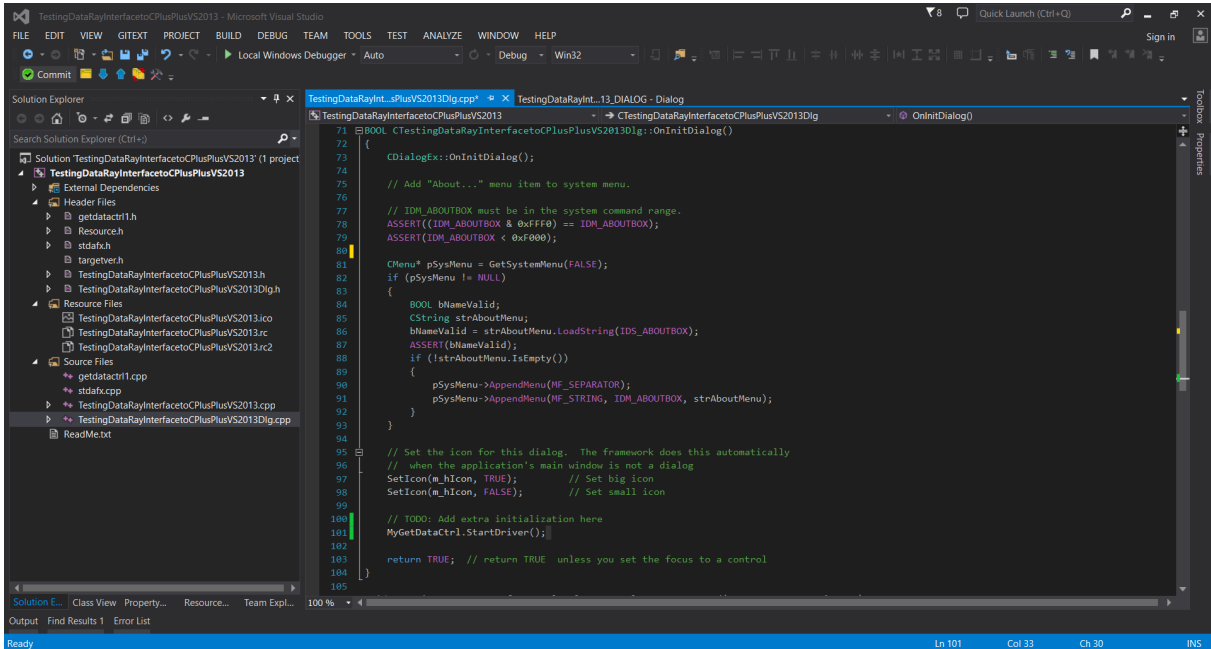
Figure 12: Add the following line: `MyGetDataCtrl.StartDriver();` then to your initialize dialog function `BOOL CTestingDataRayInterfaceToCPlusPlusVS2013Dlg::OnInitDialog()`. Then add the following lines `MyGetDataCtrl.SetResolutionAndROI(0,0,0,2048,2048);` This sets the resolution for our device to make it more consistent to parse data obtained from the Variant. The arguments are as follows, Resolution being 0, 1 indicating Fast Resolution and Full resolution respectively. Then Left, Top, Width, and Height. We want to start from the top left so we make those 0,0 and we are setting the Resolution to be 2048, 2048. Finally we want to start the device using the following command. `MyGetDataCtrl.StartDevice();`
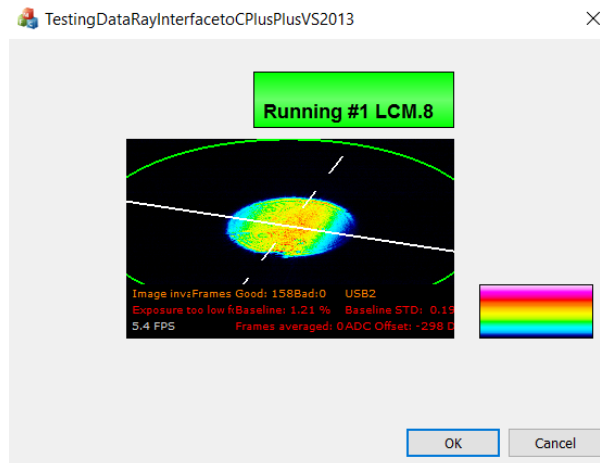

Figure 13: Now you are ready to build the project. Build and run your project. The green button is exactly the same **Ready** button as in the DataRay software. Click on the button to begin running your camera. You should see something similar to this, depending on your laser source. Congratulations, you are now interfacing with your DataRay device.
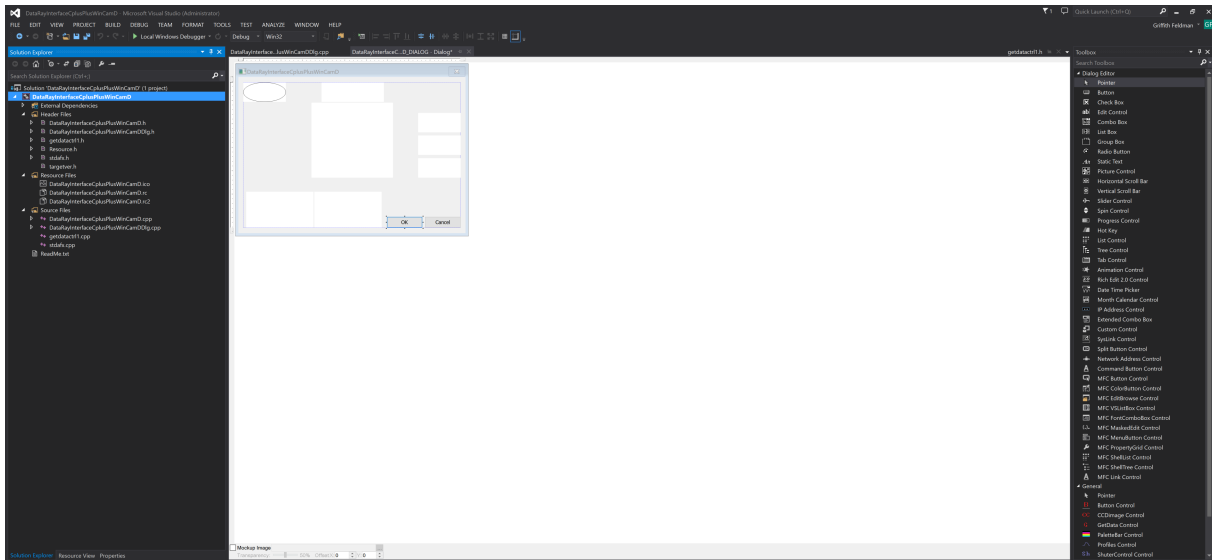
Figure 14: Now, we will add a few more objects. For this example, we want to display the X-axis profile, Y-axis profile, and the calculated centroid positions (**Xc** and **Yc**). Add two **Profile Controls** and two **Button Controls**.
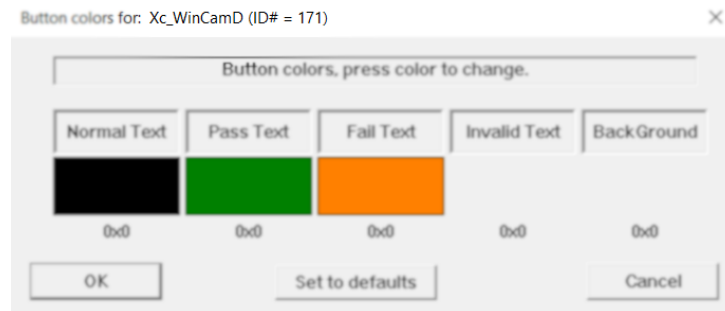

Figure 15: In order to find the correct **ButtonID** to use for each object in your custom interface, you need to:

1. Close VS2013 and open the DataRay software

2. Right click on any button, to see the dialog

3. Note the current Name and ID# for this result at the top of the dialog

4. Repeat for all the results of interest

5. Close the DataRay Software

Following these instructions, you will be able to tell that to see **Xc** and **Yc**, we should change the **ButtonIDs** to **171** and **172**. For the profiles, change the **ProfileIDs** to **22** and **23**.
A complete list of Button IDs:
Buttons
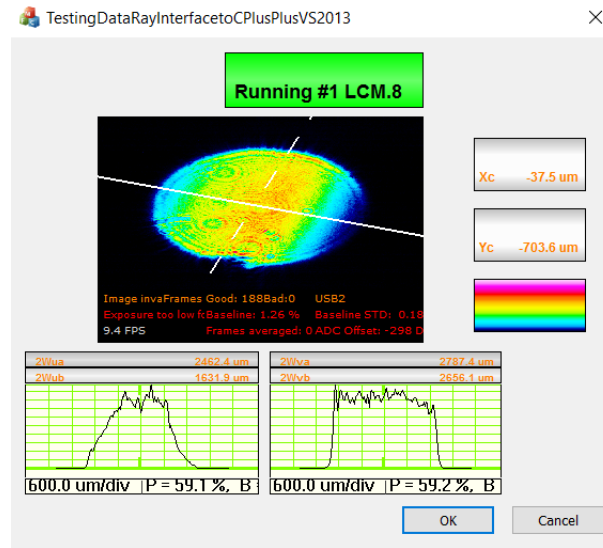A complete list of Profile IDs:
Profiles

Figure 16: Build and run your program and you should custom interface featuring **Xc button**, **Yc button**, **Xc profile** and **Yc profile**. This completes the basic tutorial! **Problems/Questions?** Contact us with the information listed on the first page of this document. Continue reading for additional tutorials regarding programmatically extracting data from the OCX and event handling.

Programmatically Extracting Data from the OCX:

There are two main methods for extracting data from the OCX in your program. One way is to create an instance of the control class (same steps as earlier for the **GetData Control**). For example, you could create a variable called **MyXcButton** for the Button with ID **171**. Then, the following line of code will give you the value from the button:

```
double XMyXcButton.GetParameter();
```

You can also query the **GetData Control** directly for parameters:

```
double Xc_FromOCXResult=MyGetDataCtrl.GetOcxResult(171);
```

where the argument for the **GetOcxResult** method is the same number used to ID the button.
The OCX also supports sending arrays of data via variants:

```
VARIANT MyXVar=MyXProfile.GetProfileDataAsVariant();
```

This is the preferred method for reading large amounts of data from the OCX. In this section of tutorial, we will create a button that will, when clicked read and retrieve the 2D image data via a avariant and store the information as an array of pixel data in a .csv file. The image data is obtained from the OCX through invocation of the FillVariantWithWinCamData() function of the GetData ActiveX control (see Fig.19 for sample code.
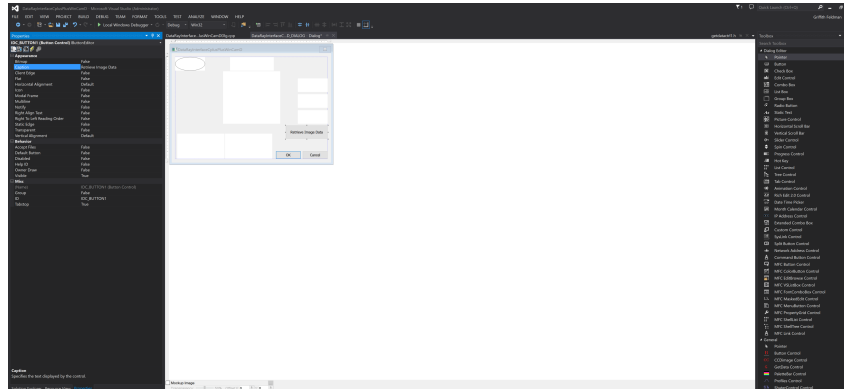
Figure 17: Using the toolbox, add an object that users can click by dragging a **Button** from the Dialog Editor section (not a Button Control from the Primitives section). Right click on the button and choose **Properties**. Change the caption to Retrieve Image Data. Next, right click on the button and choose **Add Event Handler**. Make sure the "Message type" is set to **Bn_Clicked** and set the Function handler name to **OnBnClickedRetrieveImageData**.

Figure 18: Adding the event handler to the Retrieve Image Data button will automatically create a function called OnBnClickedRetrieveImageData() in the Testing DataRayInterfacetoCPlusPlusVS2013.cpp source file. Within this function's definition, you will write your code that retrieves the 2D image data from the DataRay OCX and save the data as a .csv file. This sample code saves data retrieved from FillVariantWithWinCamData() in a file called ImageData.csv, which is created in the TestingDataRayInterfacetoCPlusPlusVS2013 folder when the button is clicked. If you plan on using the CComSafeArray to handle the variant data, be sure to include the atlsafe.h library in the header file. Code references can be found at the end of the tutorial is the image is difficult to read.

This is the preferred method for reading large amounts of data from the OCX. In this section of tutorial, we will create a button that will, when clicked, read and retrieve the 2D image data via a variant and store the information as an array of pixel data in a .csv file. The image data is obtained from the OCX through invocation of the GetWinCamDataAsVariant() function of the GetData ActiveX control (see Fig. 19 for sample code).

Event Handling:

Utilize event handling if you would like to produce code that will be executed only when a specified event occurs.
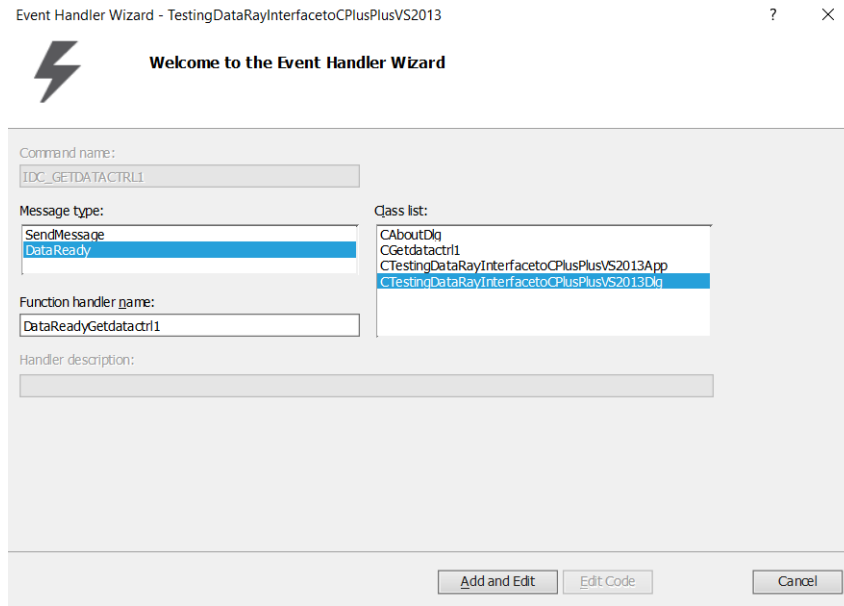

Figure 19: To begin event handling for GetData Control events, right click the GetData Control object in the dialog box and select **Add Event Handler**. Choose **DataReady** from the "Message type:" menu. The function handler name will be set to DataReadyGetdatactrl1.
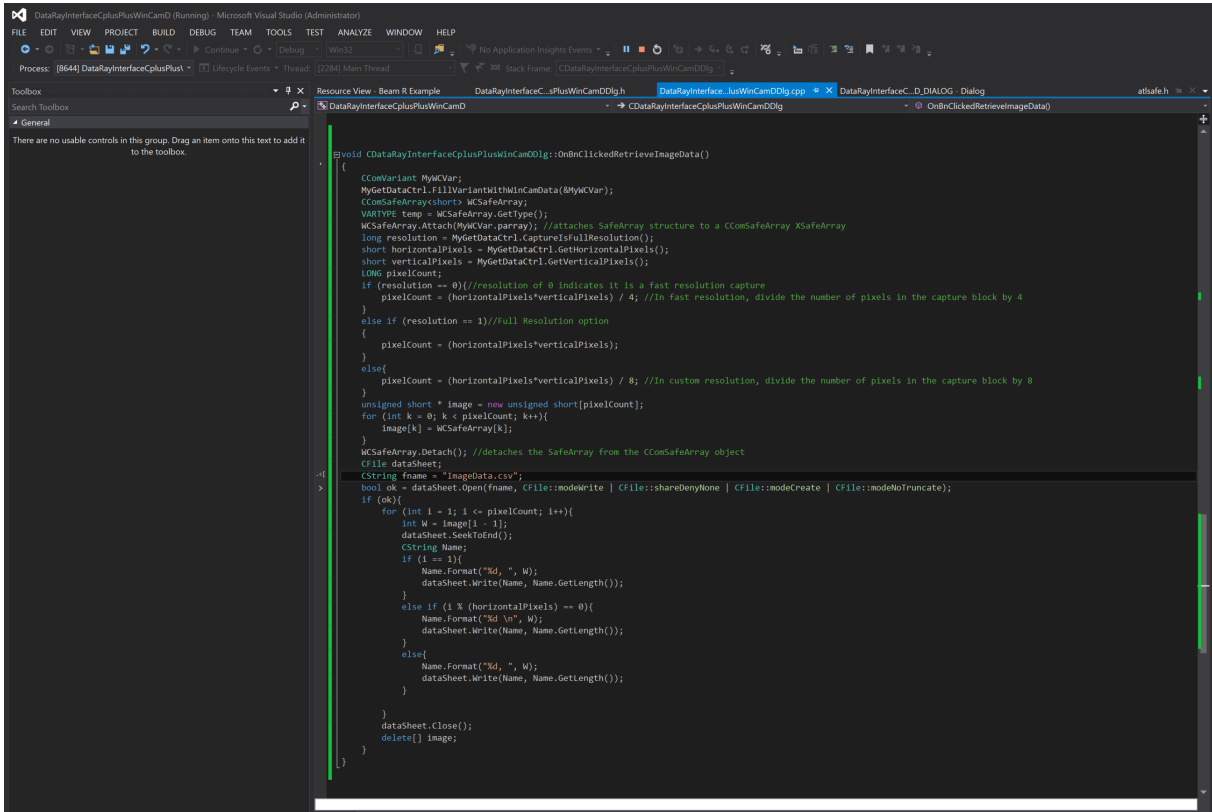
Figure 20: Locate the DataReadyGetdatactrl1() function in TestingDataRayInterfacetoCPlusPlusVS2013Dlg.cpp. This func-
tion will be trigger each time a GetData Control event occurs. Write your event handling code here.

Interfacing with Scanning Slit Beam Profilers

You can interface with the Scanning Slit Beam Profilers almost in exactly the same way with a few key differences.

- Button Id's may be specific to the type of device you are using. You can open the software and right click on the buttons to see what would be compatible with your product.

- Profile Id's similarly may be different. Check out the profile Id reference page to see what may apply to your product. Profiles. You can also look at our example code to see what Profile Id's we used.

- On the WincamD we used a CCD Image. This will not work with the Scanning Slit Beam Profilers. Instead we use a 2D Control. We can use it the same way we used the CCD Image as follows: `wx.lib.activex.ActiveXCtrl(self.⏎ frame,size=(400,400),axID='DATARAYOCX.TwoDCtrl.1')`

Reference Code

```
void CDataRayInterfaceCplusPlusWinCamDDlg::OnBnClickedRetrieveImageData()
{
        CComVariant MyWCVar;
        MyGetDataCtrl.FillVariantWithWinCamData(&MyWCVar);
        CComSafeArray<short> WCSafeArray;
```

```cpp
VARTYPE temp = WCSafeArray.GetType();
WCSafeArray.Attach(MyWCVar.parray); //attaches SafeArray structure to a CComSafeArray
↪   XSafeArray
long resolution = MyGetDataCtrl.CaptureIsFullResolution();
short horizontalPixels = MyGetDataCtrl.GetHorizontalPixels();
short verticalPixels = MyGetDataCtrl.GetVerticalPixels();
LONG pixelCount;
if (resolution == 0){//resolution of 0 indicates it is a fast resolution capture
        pixelCount = (horizontalPixels*verticalPixels) / 4; //In fast resolution, divide
        ↪   the number of pixels in the capture block by 4
}
else if (resolution == 1)//Full Resolution option
{
        pixelCount = (horizontalPixels*verticalPixels);
}
else{
        pixelCount = (horizontalPixels*verticalPixels) / 8; //In custom resolution, divide
        ↪   the number of pixels in the capture block by 8
}
unsigned short * image = new unsigned short[pixelCount];
for (int k = 0; k < pixelCount; k++){
        image[k] = WCSafeArray[k];
}
WCSafeArray.Detach(); //detaches the SafeArray from the CComSafeArray object
CFile dataSheet;
CString fname = "ImageData.csv";
bool ok = dataSheet.Open(fname, CFile::modeWrite | CFile::shareDenyNone | CFile::modeCreate
↪   | CFile::modeNoTruncate);
if (ok){
        for (int i = 1; i <= pixelCount; i++){
                int W = image[i - 1];
                dataSheet.SeekToEnd();
                CString Name;
                if (i == 1){
                        Name.Format("%d, ", W);
                        dataSheet.Write(Name, Name.GetLength());
                }
                else if (i % (horizontalPixels) == 0){
                        Name.Format("%d \n", W);
                        dataSheet.Write(Name, Name.GetLength());
                }
                else{
                        Name.Format("%d, ", W);
                        dataSheet.Write(Name, Name.GetLength());
                }

        }
        dataSheet.Close();
        delete[] image;
}
}
```