

## 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.

### 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). Download an example below:

- Cameras: [Cameras](#)
- BeamMap2: [BeamMap2](#)
- Beam'R2: [BeamR2](#)

### Build and run example:

When building you will need to change the target from AnyCPU to x86 or x64 based off what version of the OCX you have installed. For 32 bit set the target to x86 for 64 bit set it to x64. When doing that the example should now build and run with no errors (Fig. 1). Not working? Email [support@dataray.com](mailto:support@dataray.com) 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 Form programming, and Visual Studio is required.

- The OCX is only functional as part of a GUI-based program.
- In the Design View of VS2013, elements may appear as white boxes or as the actual GUI element they represent.

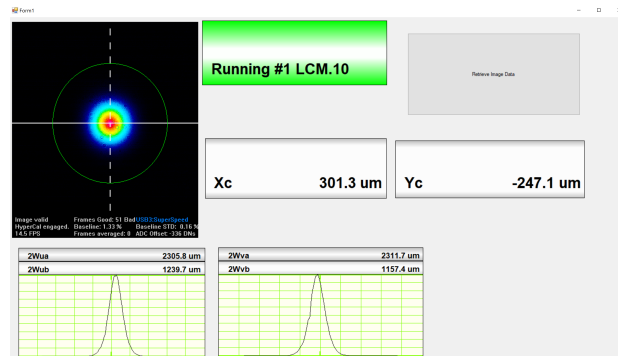


Figure 1: Example application built and running.

## 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.

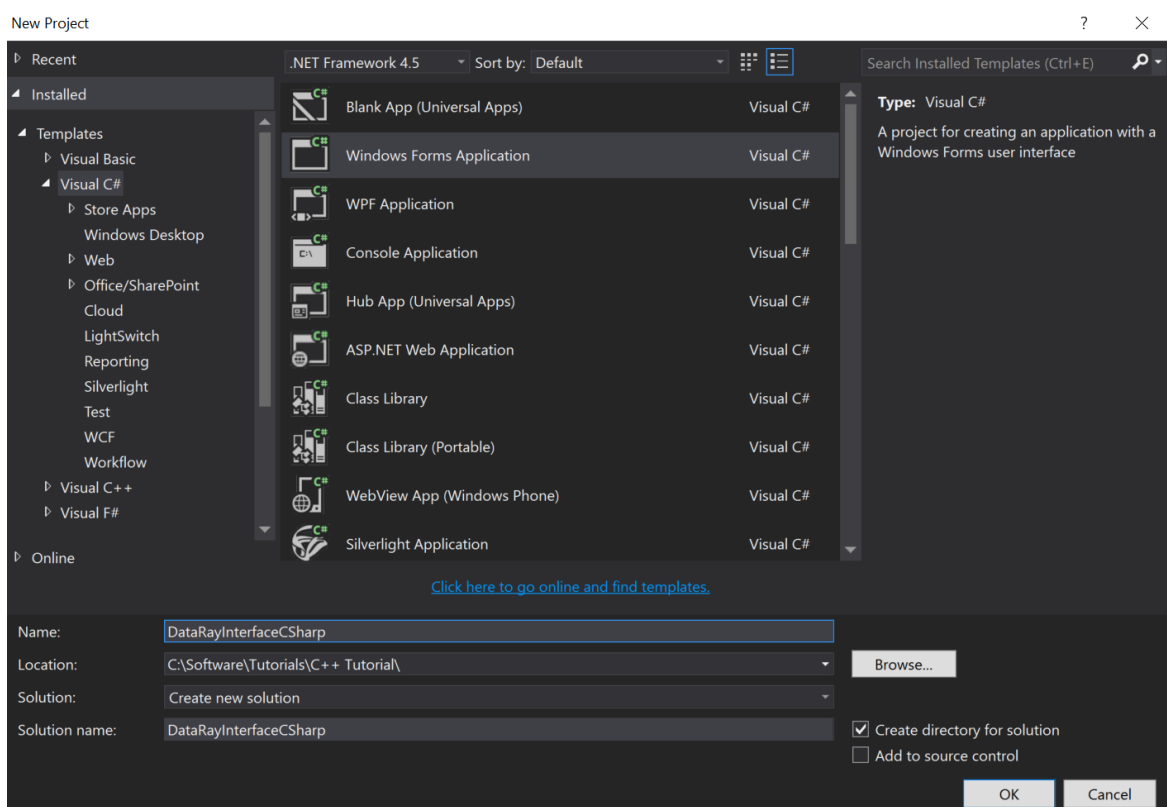


Figure 2: First, create a new Windows Form .

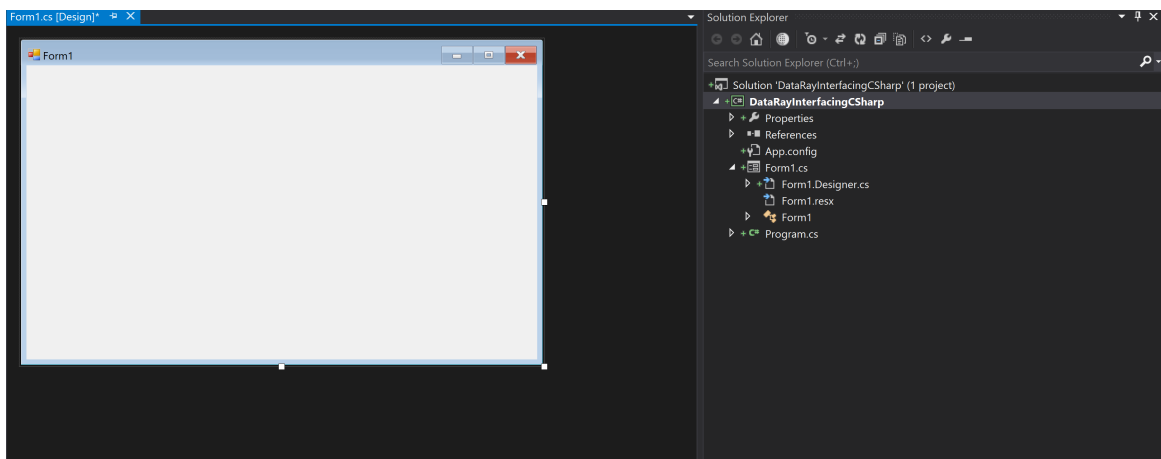


Figure 3: The empty project should look like this.



- 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 4: 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.

After this check the properties of your designer. Set AutoScaleMode to None.

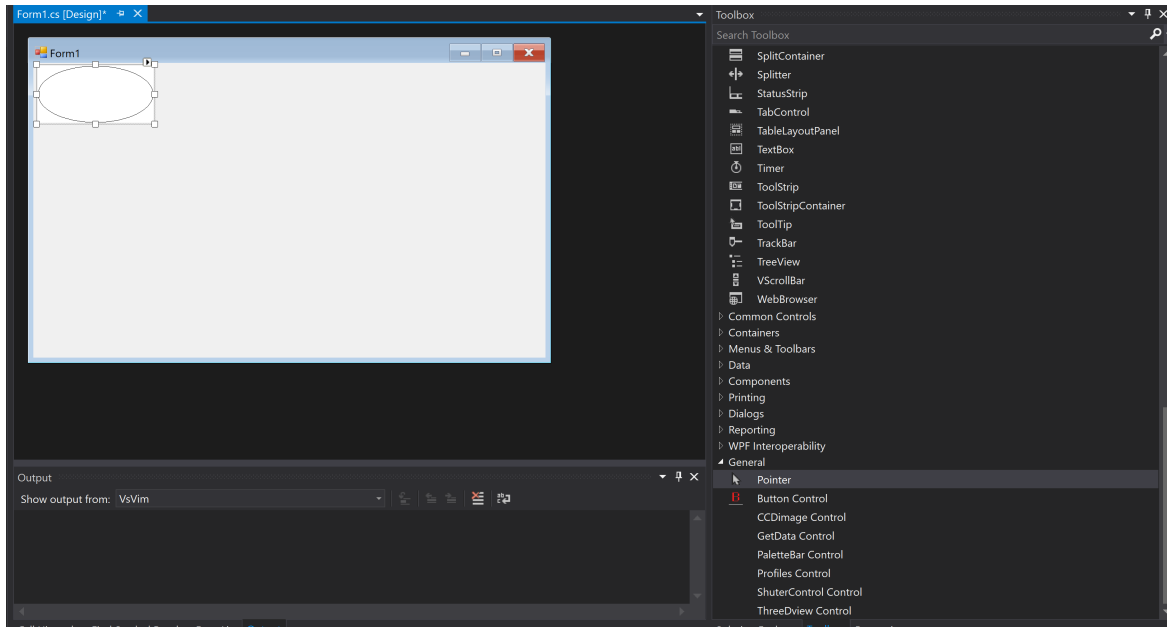


Figure 5: Now we can begin building the actual program. First select a **GetData Control** (**not DataRayGetData Control**) and create one on the dialog box. This is the only OCX control class required for interfacing to DataRay cameras.

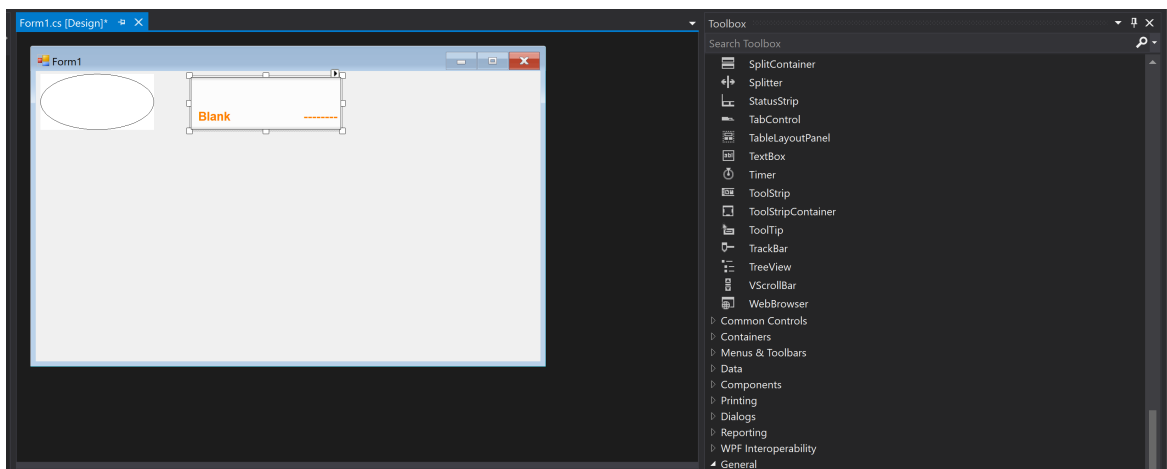


Figure 6: We will also create a **Ready** button and a display for the two-dimensional camera display (known as a **CCDImage**) Select and create a **Button Control** on the dialog box. Click on the top right arrow button. Select **ActiveX-Properties**. Change the **ButtonID** to **297**.

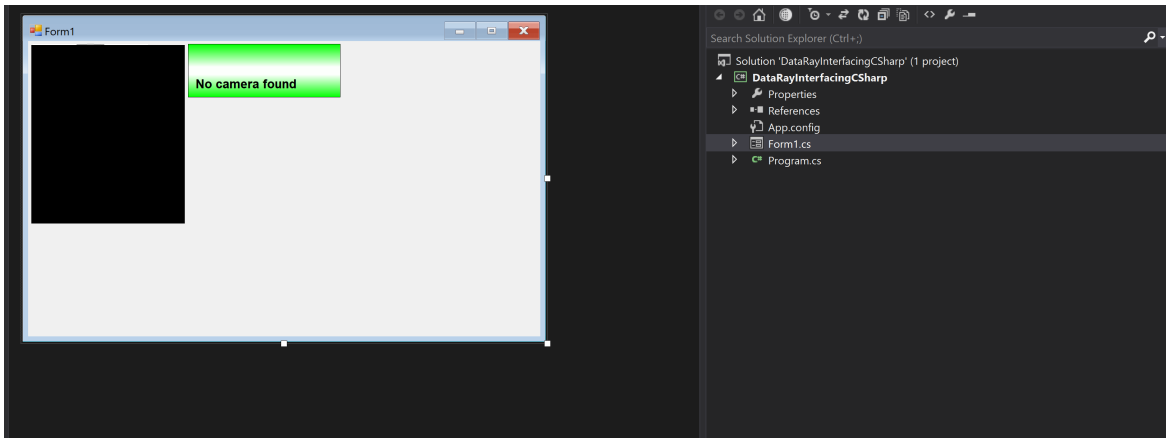


Figure 7: Now create one **CCDimage Control**. This object will be used to view the image produced by the CCD, so make the object as big as you would like. This completes the basic layout of the dialog box.



Figure 8: By default when you add the GetData Control it will create a member called **axGetData1**. This object contains the method `StartDriver()` that needs to be called to initialize the camera. You can change the name of this variable by highlighting the GetData Control editing the properties and changing the name.

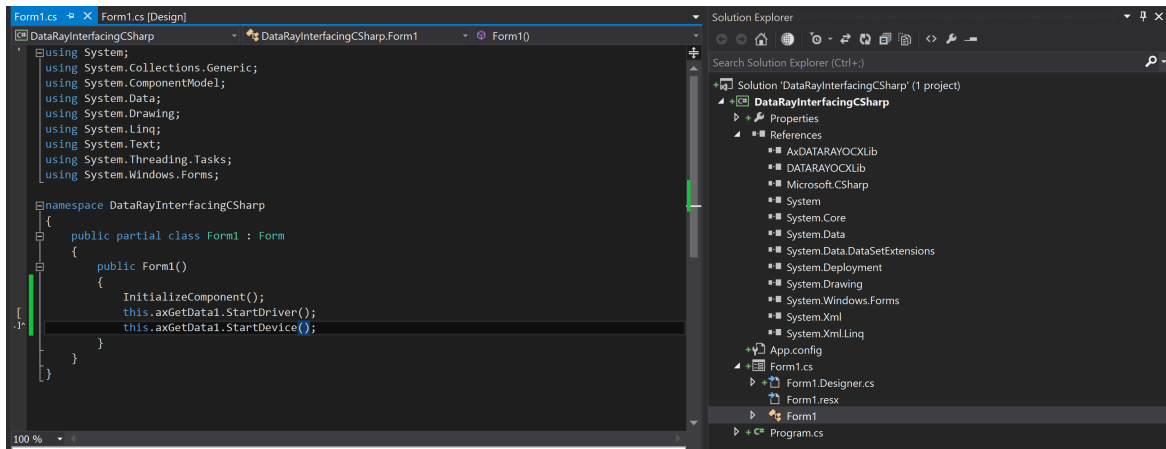


Figure 9: Add the following line: `this.axGetData1.StartDriver();` then to your public Form1 constructor in Form1.cs. Finally we want to start the device using the following command. `this.axGetData1.StartDevice();`

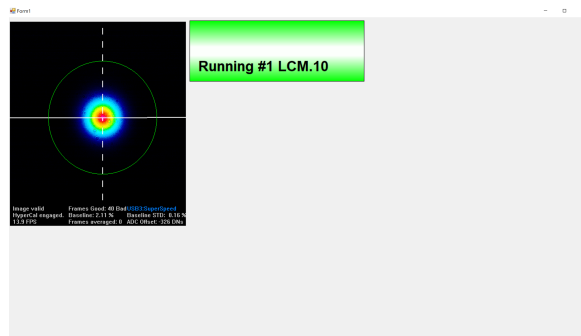


Figure 10: Now you are ready to build the project. Build and run your project. The green button is exactly the same **Ready** button as in the DataRay software. Click on the button to begin running your camera. You should see something similar to this, depending on your laser source. Congratulations, you are now interfacing with your DataRay device.

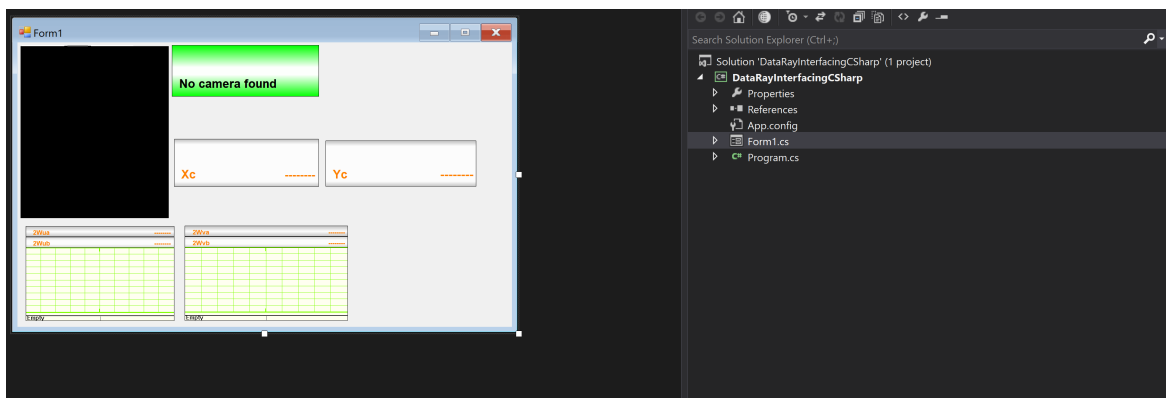


Figure 11: Now, we will add a few more objects. For this example, we want to display the X-axis profile, Y-axis profile, and the calculated centroid positions (**Xc** and **Yc**). Add two **Profile Controls** and two **Button Controls**.

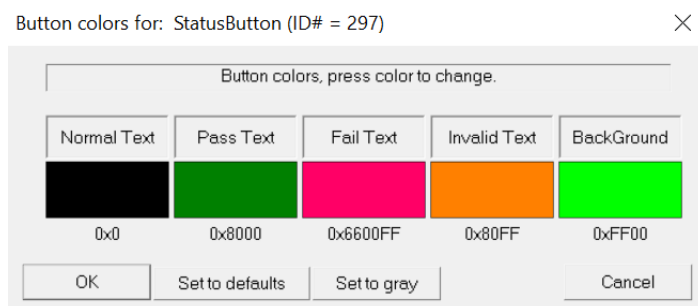


Figure 12: In order to find the correct **ButtonID** to use for each object in your custom interface, you need to:

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](#)



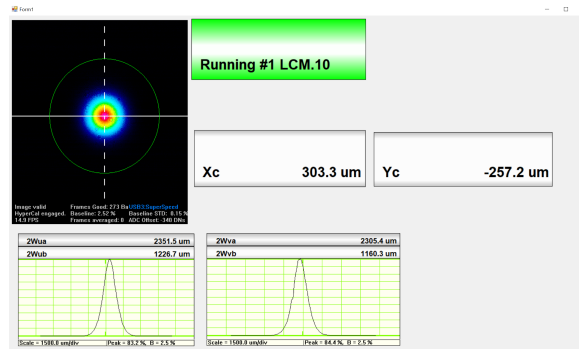


Figure 13: Build and run your program and you should have a custom interface featuring **Xc button**, **Yc button**, **Xc profile** and **Yc profile**. This completes the basic tutorial! **Problems/Questions?** Contact us with the information listed on the first page of this document. Continue reading 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=axGetData1.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:

```
object image;
short[] array;
image=MyXProfile.GetProfileDataAsVariant() as short[];
array = axGetData1.GetProfileDataAsVariant as short[];
```

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.15 for sample code).

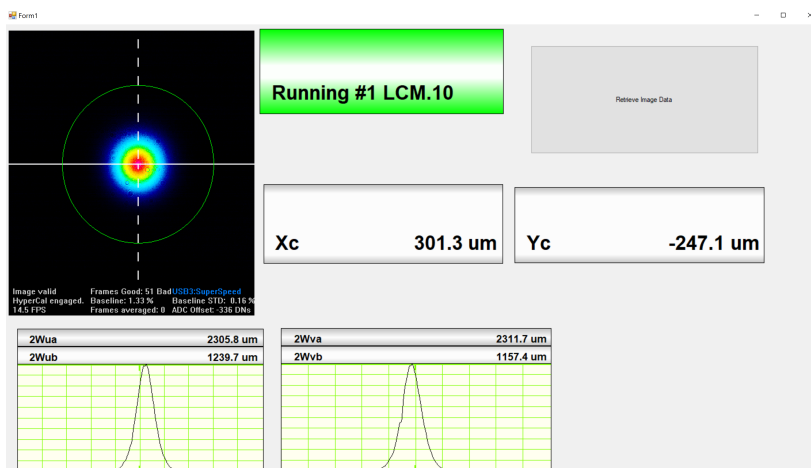


Figure 14: Using the toolbox, add an object that users can click by dragging a **Button** from the Dialog Editor section (not a Button Control from the Primitives section). Right click on the button and choose **Properties**. Change the caption to Retrieve Image Data. Next, change the name of the button to ImageButton. Next, click on the lightning bolt top right of the properties window. This is the event handlers section. We want to create an on click handler so we can type in our function name and then hitting enter and clicking on it again will take you to the generated function.

```
private void RetrieveImageData(object sender, EventArgs e)
{
    short[] image;
    short[] array;
    image = axGetData1.GetWinCamDataAsVariant() as short[];
    array = axGetData1.GetWinCamDataAsVariant() as short[];
    long resolution = axGetData1.CaptureIsFullResolution();
    short horizontalPixels = axGetData1.GetHorizontalPixels();
    short verticalPixels = axGetData1.GetVerticalPixels();
    long pixelCount;
    //resolution of 0 indicates it is a fast resolution capture
    if (resolution == 0)
    {
        pixelCount = (horizontalPixels * verticalPixels) / 4;
    }
    //Full Resolution
    else if (resolution == 1)
    {
        pixelCount = (horizontalPixels * verticalPixels);
    }
    else
    {
        pixelCount = (horizontalPixels * verticalPixels) / 8;
    }
    string exeFolder = System.IO.Path.GetDirectoryName(Application.ExecutablePath);
    string pathname = exeFolder + "\\ImageData.csv";
    StreamWriter sw = new StreamWriter(pathname);
    for (int i = 1; i <= array.Length; i++)
    {
        int W = image[i - 1];
        string Name;
        if (i == 1)
        {
            Name = W + ", ";
            sw.Write(Name);
        }
        else if (i % (horizontalPixels) == 0)
        {
            sw.Write(W + Environment.NewLine);
        }
        else
        {
            Name = W + ", ";
            sw.Write(Name);
        }
    }
    sw.Close();
}
```

Figure 15: 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 executable path folder when the button is clicked. Since we used StreamWriter you will need to include **using system.IO**; at the top of the page. Code references can be found at the end of the tutorial is the image is difficult to read.

## Event Handling:

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

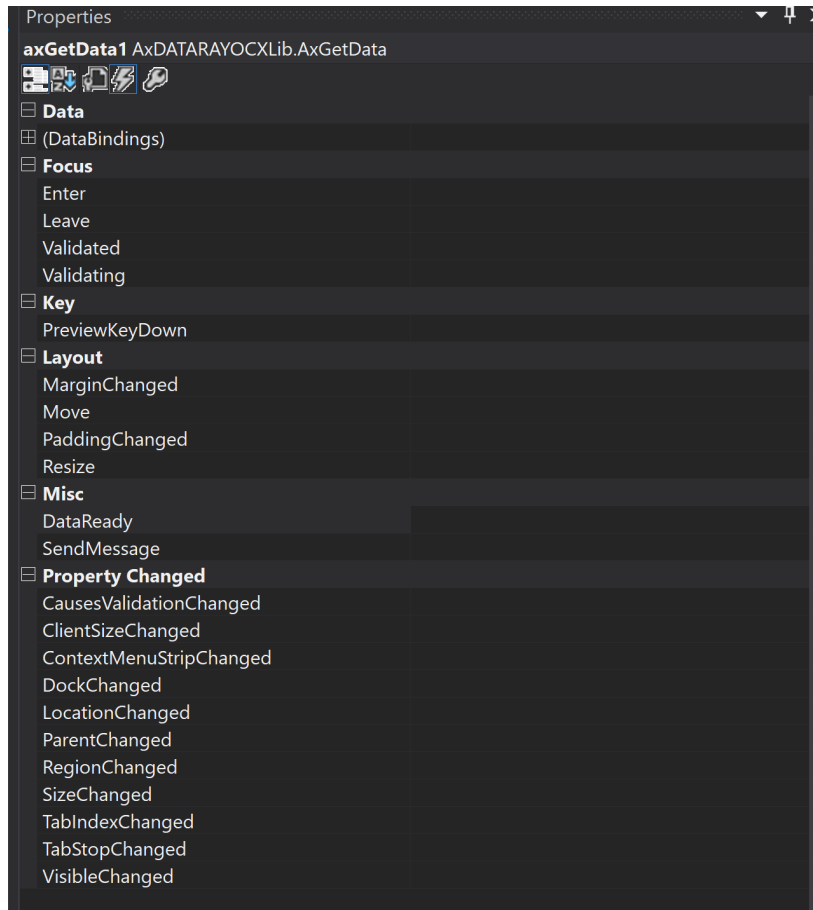


Figure 16: To begin event handling for GetData Control events, click the GetData Control object, go to its properties and click on the Event Handler lightning bolt. Choose **DataReady**. The function handler name will be set to axGetData1\_DataReady. Put whatever code you want to occur on a DataReady Event there.

```
private void axGetData1_DataReady(object sender, EventArgs e)
{
}
}
```

Figure 17: Locate the axGetDAta1\_DataReady(object sender, EventArgs e) function in Form1.cs. This function 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.

## Reference Code

```
private void ImageDataBn_Click(object sender, EventArgs e)
{
    short[] image;
    short[] array;
    image = axGetData1.GetWinCamDataAsVariant() as short[];
    array = axGetData1.GetWinCamDataAsVariant() as short[];
    long resolution = axGetData1.CaptureIsFullResolution();
    short horizontalPixels = axGetData1.GetHorizontalPixels();
    short verticalPixels = axGetData1.GetVerticalPixels();
    long pixelCount;
    //resolution of 0 indicates it is a fast resolution capture
    if (resolution == 0)
    {
        pixelCount = (horizontalPixels * verticalPixels)/4;
    }
    //Full Resolution
    else if(resolution==1){
        pixelCount = (horizontalPixels * verticalPixels);
    }
    else
    {
        pixelCount = (horizontalPixels * verticalPixels) / 8;
    }
    string exeFolder = System.IO.Path.GetDirectoryName(Application.ExecutablePath);
    string pathname = exeFolder + "\\ImageData.csv";
    StreamWriter sw = new StreamWriter(pathname);
    for(int i = 1; i<=array.Length;i++)
    {
        int W = image[i - 1];
        string Name;
        if (i == 1)
        {
            Name = W + ", ";
            sw.Write(Name);
        }
        else if (i % (horizontalPixels) == 0)
        {
            sw.Write(W+Environment.NewLine);
        }
        else
        {
            Name = W + ", ";
            sw.Write(Name);
        }
    }
    sw.Close();
}
```