# Interfacing to Python

## Contents

## Getting Started

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. This tutorial is for a WinCamD; if you are using a different device, some of the ID's and ActiveX controls will be different. In this tutorial we will go over how to do this using Anaconda3.

## Some Important Notes:

- The OCX is functional only as part of a GUI-based program. In this tutorial, we use the wxPython library

- If the installed OCX is 32-bit, you will need associated 32-bit Python and libraries. Similarly, if the installed OCX is 64-bit, you will need associated 64-bit Python.

- This tutorial assumes familiarity with Python

## Installation

First we need to install the DataRay Software

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

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

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

Second we will set up our environment

- Install Anaconda at `https://www.anaconda.com/`. We used the Individual Edition for this but any edition should work. No special arguments for the installation are needed

- Open *Anaconda Prompt* in the Windows search bar.

  – We will be setting up our environment from this prompt. First, we want to create our environment which we will be working in.

  – If you are using a 32-bit version of the OCX, this command we will be force a download of 32-bit Python. The command is:

    Set CONDA_FORCE_32BIT=1

    Alternatively, for a 64-bit environment:

    Set CONDA_FORCE_64BIT=1

    This sets the flag for our install.

  – Next we will create an create the environment using the conda package manager and install all the necessary commands. Run:

    conda create --name myGui python=3.7.1 wxpython=4.0.4 comtypes=1.1.7.

  – We called ours myGui but any name works. Just make sure you are activating the correct environment when interfacing.

  – To enter the environment to install packages and run programs we will be using the command

    conda activate myGui

    similarly to exit the environment use

    conda deactivate

  – Once in the environment you may want to check your Python version using the command

    python −version

  – You can also verify if you have the correct sized Python by running the command *python*.

  – You can install any packages that you may need with the command *conda install <package name>*.

  – The numbers at the end of the command indicate the version number. At writing this tutorial these are the most current and what work. It is possible that there will be more current versions that may not function as intended.

  – In this environment we can run the program using *python <path to file><file name>.*

You can download the interface developed in this tutorial. It exists as a .py file. You can work on the Python file in any editor. We used a combination of notepad++ and spyder.

- WinCamD Example

This example should build and run with no errors. Not working? Email support@dataray.com or call 530-472-1717 with:

- Device name and serial number

- DataRay and Windows versions you are using.
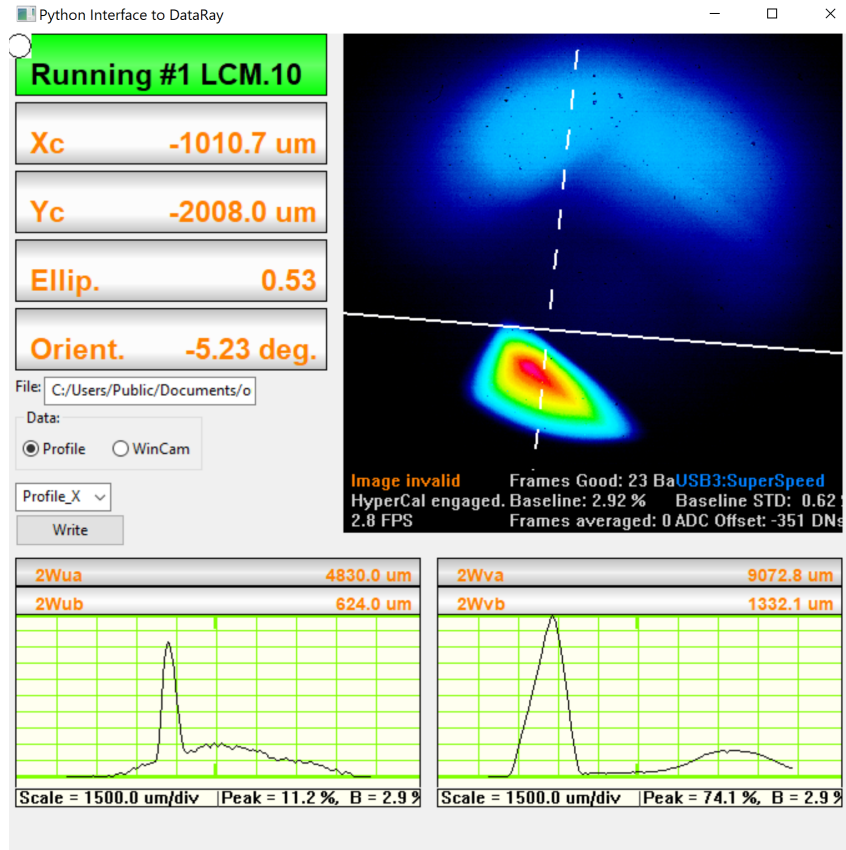
## BASIC TUTORIAL:

We will show you step-by-step how the example program was created.

### GUI WITH wxPYTHON

First, we will make a basic GUI with wxPython. The wxPython library is a wrapper around the wxWidgets library for cross-platform GUI's. It supports all the items you would expect from a GUI library. The following code will create a window and run a program:

```python
import wx

class MyApp( wx.App ):
    def __init__( self, redirect=False, filename=None ):
        wx.App.__init__( self, redirect, filename )
        self.frame = wx.Frame( parent=None, id=wx.ID_ANY,size=(900,900),
                               title='Python Interface to DataRay')
        self.frame.Show()

if __name__ == "__main__":
    app = MyApp()
    app.MainLoop()
```
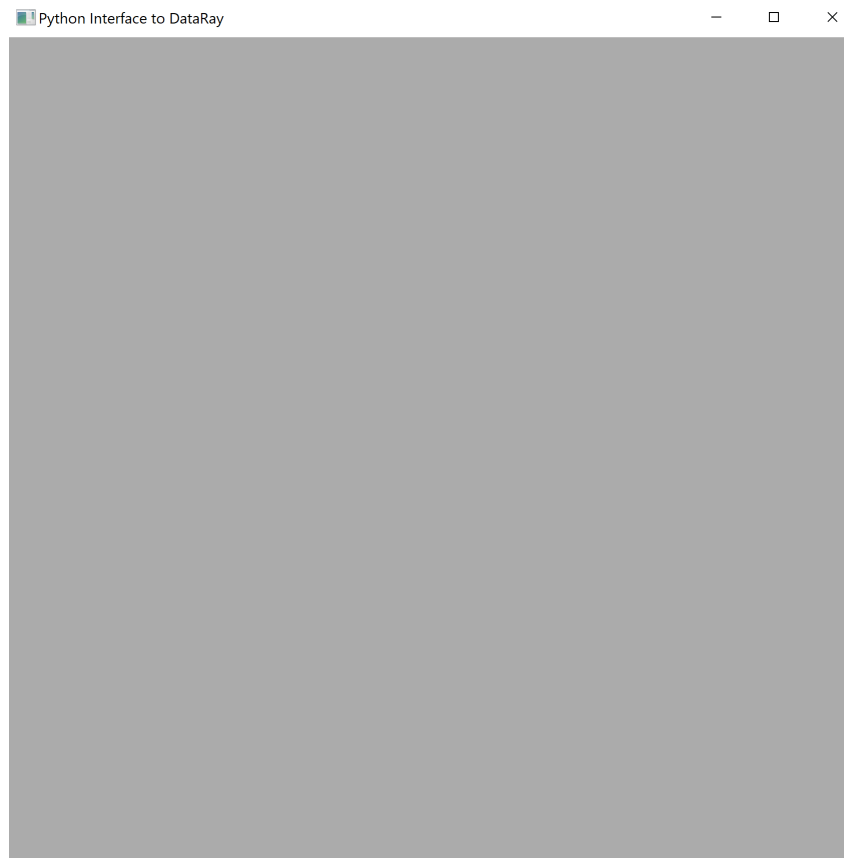
Without using the frame's "Show" method, the window will not be visible. The black window created in the background is an instance of the Python interpreter. Any print statements will output to that window.

## FIRST BUTTON AND IMAGE

To add ActiveX components, we need to import the ActiveX library of wxPython:

```python
import wx.lib.activex
```

ActiveX components are added with the following class:

```
wx.lib.activex.ActiveXCtrl(
```
ActiveXCtrl(**parent: Frame**, axID: str, wxid = -1, pos: Point = wx.DefaultPosition, size: Size = wx.DefaultSize, style: int = 0, name: str = 'activeXCtrl')
All parameters are like those used in normal wx.Windows with
the addition of axID which is a string that is either a ProgID
or a CLSID used to identify the ActiveX control.

As a minimum, the ActiveXCtrl class needs the parent and axID (ActiveXID) arguments to initialize, but it can also take arguments for size and position. The parent argument must either be a frame or be a wx item which has a frame as its parent, grandparent, etc…

```python
import wx
import wx.lib.activex

class MyApp( wx.App ):
    def __init__( self, redirect=False, filename=None ):
        wx.App.__init__( self, redirect, filename )
        self.frame = wx.Frame( parent=None, id=wx.ID_ANY,size=(900,900),
                               title='Python Interface to DataRay')
        #Panel
        p = wx.Panel(self.frame,wx.ID_ANY)
        #Get Data
        self.gd = wx.lib.activex.ActiveXCtrl(p, 'DATARAYOCX.GetDataCtrl.1')
        self.frame.Show()
        self.gd.ctrl.StartDriver()

if __name__ == "__main__":
    app = MyApp()
    app.MainLoop()
```
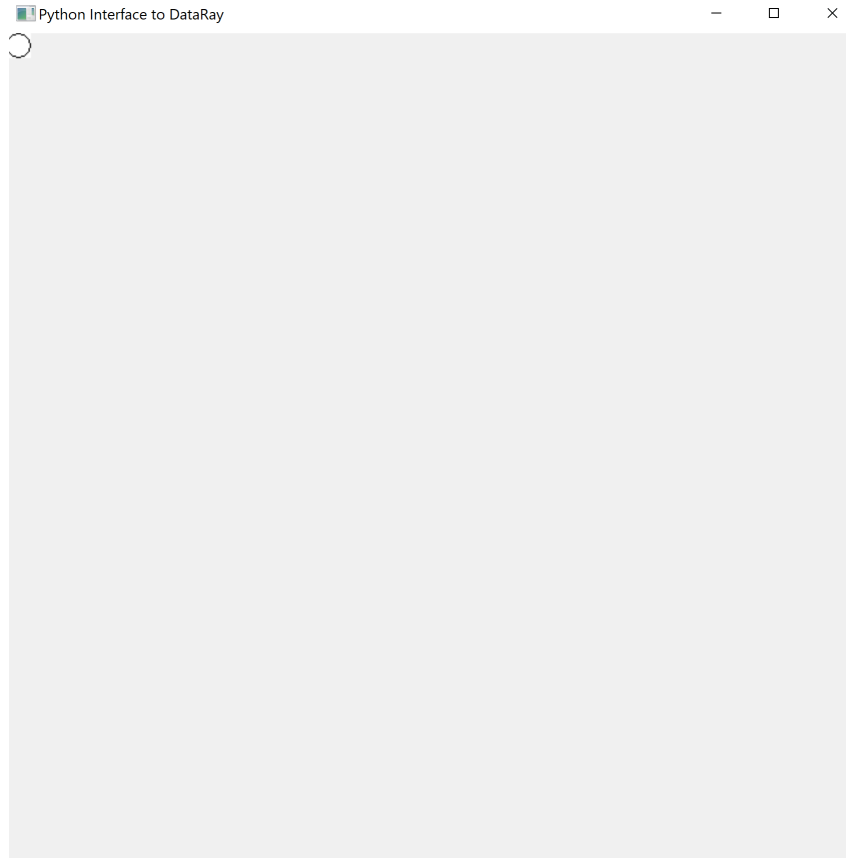
The GetDataCtrl control must be present for the OCX to start with the "StartDriver" method. To make this control accessible by other methods of our "MyApp" class, we make it a property with "self.gd =" instead of "gd =". The reason for doing this will become clear in the Advanced Tutorial. All methods and properties of ActiveX components are available through the "ctrl" property of each instance of the ActiveXCtrl class. Next we will create a Button ('DATARAYOCX.ButtonCtrl.1') and the two-dimensional camera display CCDImage ('DATARAYOCX.CCDimageCtrl.1'). The size argument takes a tuple of the form (horizontal pixels, vertical pixels):

```python
import wx
import wx.lib.activex

class MyApp( wx.App ):
    def __init__( self, redirect=False, filename=None ):
        wx.App.__init__( self, redirect, filename )
        self.frame = wx.Frame( parent=None, id=wx.ID_ANY,size=(900,900),
                               title='Python Interface to DataRay')
        #Panel
        p = wx.Panel(self.frame,wx.ID_ANY)
        #Get Data
        self.gd = wx.lib.activex.ActiveXCtrl(p, 'DATARAYOCX.GetDataCtrl.1')
        self.frame.Show()
        #Button
        b1 = wx.lib.activex.ActiveXCtrl(parent=p,size=(200,50), pos=(7, 0),
                                        axID='DATARAYOCX.ButtonCtrl.1')
        b1.ctrl.ButtonID = 297
```
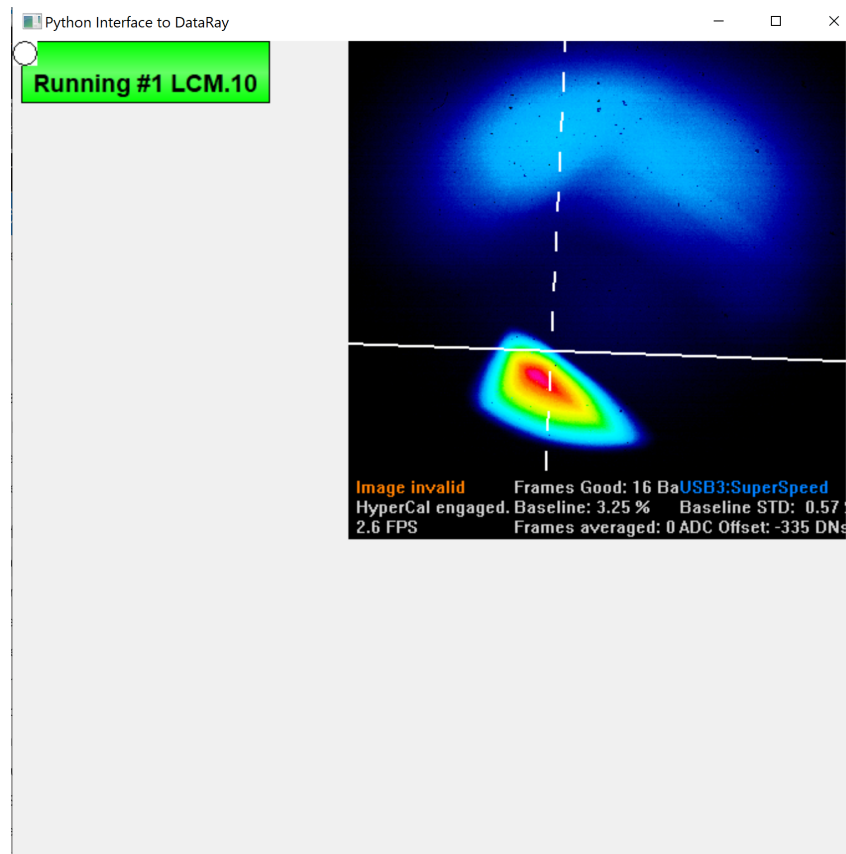
```
#CCDImage
wx.lib.activex.ActiveXCtrl(parent=p,axID='DATARAYOCX.CCDimageCtrl.1',
                            size=(250,250),pos=(250,250))
self.gd.ctrl.StartDriver()

if __name__ == "__main__":
    app = MyApp()
    app.MainLoop()
```



The position argument determines where on the screen these will be located. In this tutorial we have provided some that make it easy to test but they are arbitrary. Feel free to reloacte any element to anywhere you would like on the screen. The wxPython library supplies many ways to make formatting interfaces easier.

## MORE BUTTONS

Next we will add 4 more buttons to display more to the GUI options. We can relocate these to anywhere on the screen but we set them to be in the top right corner for convenience. Later in the tutorial we will go over how to look up and add different types of buttons and controls

```
import wx
import wx.lib.activex
```

```python
class MyApp( wx.App ):
    def __init__( self, redirect=False, filename=None ):
        wx.App.__init__( self, redirect, filename )
        self.frame = wx.Frame( parent=None, id=wx.ID_ANY,size=(900,900), title='Python Interface
        ↪  to DataRay')
        #Panel
        p = wx.Panel(self.frame,wx.ID_ANY)
        #Get Data
        self.gd = wx.lib.activex.ActiveXCtrl(p, 'DATARAYOCX.GetDataCtrl.1')
        self.frame.Show()
        #Button
        b1 = wx.lib.activex.ActiveXCtrl(parent=p,size=(200,50), pos=(7, 0),
                                        axID='DATARAYOCX.ButtonCtrl.1')
        b1.ctrl.ButtonID =297 #Id's for some ActiveX controls must be set

        b2 = wx.lib.activex.ActiveXCtrl(parent=p,size=(100,25), pos=(5, 55),
                                        axID='DATARAYOCX.ButtonCtrl.1')
        b2.ctrl.ButtonID =171
        b3 = wx.lib.activex.ActiveXCtrl(parent=p,size=(100,25), pos=(110,55),
                                        axID='DATARAYOCX.ButtonCtrl.1')
        b3.ctrl.ButtonID =172
        b4 = wx.lib.activex.ActiveXCtrl(parent=p,size=(100,25), pos=(5, 85),
                                        axID='DATARAYOCX.ButtonCtrl.1')
        b4.ctrl.ButtonID =177
        b4 = wx.lib.activex.ActiveXCtrl(parent=p,size=(100,25), pos=(110, 85),
                                        axID='DATARAYOCX.ButtonCtrl.1')
        b4.ctrl.ButtonID =179


        #CCDImage
        wx.lib.activex.ActiveXCtrl(parent=p,axID='DATARAYOCX.CCDimageCtrl.1',
                                   size=(250,250),pos=(250,250))
        self.gd.ctrl.StartDriver()

if __name__ == "__main__":
    app = MyApp()
    app.MainLoop()
```

## EXTENDING THE GUI

There are additional ActiveX controls available. When wxPython's activex library is used for the first time, comtypes is used to generate a file with interfaces for all of the ActiveX controls. You can find the generated file inside your Python directory under

`\Lib\site-packages\comtypes\gen`

If you are using the anaconda environment we go over creating in the tutorial you will find it under

`\Users\<Username>\Anaconda3\envs\<enviornmentName>\Lib\site-packages\comtypes\gen`

As of this writing, the file containing the interfaces is called

`"\_43555BA2\_3FE0\_11D6\_9F4A\_00A0CC40A4D2\_0\_1\_0.py"`.

The generated file creates three classes "_D{name}" and "_D{ name}Events" and one that is just simply the ActiveX control name. You can either use the GUID under the class named after the control or infer the ActiveX ID of the class: "DATARAY-OCX.{name}Ctrl.1". The methods and properties of the interfaces are specified in another section of the file which follows

this pattern: "_D{name}._disp_methods_".

Besides the names of the ActiveX controls, you will need to know the ID's for specific button and profiles. In order to find the correct Button ID# to use for the buttons, you need to

1.  Close your GUI and open the DataRay software

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

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

4.  Repeat for all the results of interest. Close the DataRay Software
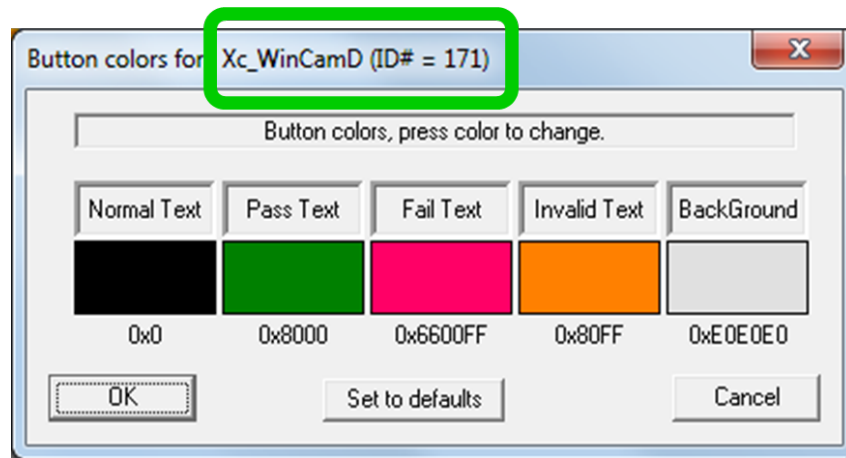


There are complete lists of ID's for profiles and buttons available in interface section of the DataRay website:

       Buttons: Buttons

       Profiles: Profiles

This completes the basic tutorial! **Problems/Questions?** Please contact us with the information listed above.

## ADVANCED TUTORIAL

### wxPYTHON CONTROLS

The wxPython library has its own controls and input methods. These can be used to provide custom functionality to your GUI. In this case, we will be using them to select data to write to a file.

First, we will add one text label, one text control (text input), one radio button box, one combo box (a dropdown list) and one button to our button panel. As with the GetData control, we will prepend "self" to the items because we want them to be accessible in other methods of our MyApp class. A default value is provided for the path and name of the file. Make sure to run the program as administrator and/or write to a directory for which you have privileges.

```python
#Custom controls
t = wx.StaticText(p, label="File:", pos=(5, 115))
self.ti = wx.TextCtrl(p, value="C:/Users/Public/Documents/output.csv", pos=(30, 115), size=(170, -1))
self.rb = wx.RadioBox(p, label="Data:", pos=(5, 140), choices=["Profile", "WinCam"])
self.cb = wx.ComboBox(p, pos=(5,200), choices=[ "Profile_X", "Profile_Y", "Both"])
self.cb.SetSelection(0)
myb = wx.Button(p, label="Write", pos=(5,225))
myb.Bind(wx.EVT_BUTTON, self.OnClick)
```

The text input will take the entire path and file name together. The radio box designates either the WinCam data or profile data to be written. The combo box designates the various profiles which can be selected, and to prevent an error due to an empty value, it has been set to a default value of "Profile_X" with its "SetSelection" method. Finally, we put a button at the bottom of the panel and bind a method to it which we will write for our "MyApp" class.

Entire tutorials have been written about events and binding in wxPython. We will stick to the basics. The Bind method takes an event type and a method to be run. Because this method needs to be defined before we bind it in our initialization, we need to put the method above our class constructor. Stylistically this may be frowned upon. Programmers writing bigger wxPython applications write everything here in a separate method to initialize their user interface, another for the bindings and call the two inside their initialization. For now, we will keep it simple and start with this definition of our "OnClick" method and put it above our initialization:

```python
class MyApp( wx.App ):
  def OnClick(self,e):
    data = self.gd.ctrl.GetWinCamDataAsVariant()
    print type(data), data
  def __init__( self, redirect=False, filename=None ):
```

This will print the results to the instance of the Python interpreter running in the background. If we hit the "Write" button before we start the camera, we will get a tuple with one zero in it. If we hit the button after we have started the camera, it will return a tuple with a length equal to the size of the number of pixels captured by the camera's current settings. Here is all of our code at the end of adding this.

```python
import wx
import wx.lib.activex

class MyApp( wx.App ):
    def __init__( self, redirect=False, filename=None ):
        wx.App.__init__( self, redirect, filename )
        self.frame = wx.Frame( parent=None, id=wx.ID_ANY,size=(900,900),
                               title='Python Interface to DataRay')
        #Panel
        p = wx.Panel(self.frame,wx.ID_ANY)
        #Get Data
        self.gd = wx.lib.activex.ActiveXCtrl(p, 'DATARAYOCX.GetDataCtrl.1')
        self.frame.Show()
        #Button
        b1 = wx.lib.activex.ActiveXCtrl(parent=p,size=(200,50), pos=(7, 0),
                                        axID='DATARAYOCX.ButtonCtrl.1')
        b1.ctrl.ButtonID =297 #Id's for some ActiveX controls must be set

        b2 = wx.lib.activex.ActiveXCtrl(parent=p,size=(100,25), pos=(5, 55),
                                        axID='DATARAYOCX.ButtonCtrl.1')
        b2.ctrl.ButtonID =171
        b3 = wx.lib.activex.ActiveXCtrl(parent=p,size=(100,25), pos=(110,55),
                                        axID='DATARAYOCX.ButtonCtrl.1')
        b3.ctrl.ButtonID =172
        b4 = wx.lib.activex.ActiveXCtrl(parent=p,size=(100,25), pos=(5, 85),
                                        axID='DATARAYOCX.ButtonCtrl.1')
        b4.ctrl.ButtonID =177
        b4 = wx.lib.activex.ActiveXCtrl(parent=p,size=(100,25), pos=(110, 85),
                                        axID='DATARAYOCX.ButtonCtrl.1')
        b4.ctrl.ButtonID =179

        #Pictures
```
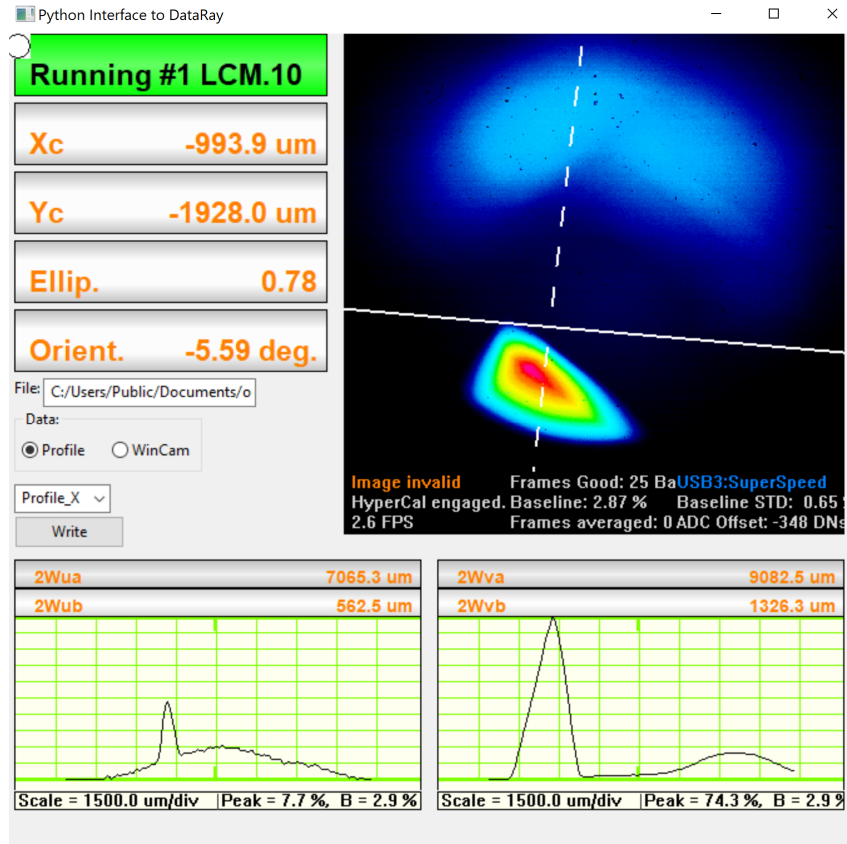
```python
        tpic = wx.lib.activex.ActiveXCtrl(parent=p,size=(250,250),
                        axID='DATARAYOCX.ThreeDviewCtrl.1',pos=(500,250))
        #Profiles
        self.px = wx.lib.activex.ActiveXCtrl(parent=p,size=(300,200),
                        axID='DATARAYOCX.ProfilesCtrl.1',pos=(0,600))
        self.px.ctrl.ProfileID=22
        self.py = wx.lib.activex.ActiveXCtrl(parent=p,size=(300,200),
                        axID='DATARAYOCX.ProfilesCtrl.1',pos=(600,600))
        self.py.ctrl.ProfileID = 23

        #CCDImage
        wx.lib.activex.ActiveXCtrl(parent=p,axID='DATARAYOCX.CCDimageCtrl.1',
                            size=(250,250),pos=(250,250))

        #Custom controls
        t = wx.StaticText(p, label="File:", pos=(5, 115))
        self.ti = wx.TextCtrl(p, value="C:/Users/Public/Documents/output.csv",
                            pos=(30, 115), size=(170, -1))
        self.rb = wx.RadioBox(p, label="Data:",
                            pos=(5, 140), choices=["Profile", "WinCam"])
        self.cb = wx.ComboBox(p, pos=(5,200),
                            choices=[ "Profile_X", "Profile_Y", "Both"])
        self.cb.SetSelection(0)
        myb = wx.Button(p, label="Write", pos=(5,225))
        myb.Bind(wx.EVT_BUTTON, self.OnClick)
        self.gd.ctrl.StartDriver()

if __name__ == "__main__":
    app = MyApp()
    app.MainLoop()
```

## GETTING DATA

Now that we have an understanding of how wxPython's controls can be set up, we will rewrite the OnClick method to output data to a comma separated value file. In order to write to a .csv file, we will import the appropriate library by adding this import statement to the top of our file:

```python
import csv
```

Next, we must change the "OnClick" method by adding control statements and writing. To export to a .csv file, you will need to give the "writerows" method a list of lists. Each list inside the main list represents a row. Data with two columns would be of the form [["Row 1 value 1", "Row 1 value 2"], ["Row 2 value 1", "Row 2 value 2"]]. Each of the four possible branches has a step to format the data variable properly.

```python
class MyApp(wx.App):
  def OnClick(self,e):
    rb_selection = self.rb.GetStringSelection()
    if rb_selection == "WinCam":
      data = self.gd.ctrl.GetWinCamDataAsVariant()
      data = [[x] for x in data]
    else:
      p_selection = self.cb.GetStringSelection()
      if p_selection == "Profile_X":
        data = self.px.ctrl.GetProfileDataAsVariant()
        data = [[x] for x in data]
      elif p_selection == "Profile_Y":
        data = self.py.ctrl.GetProfileDataAsVariant()
        data = [[x] for x in data]
      else:
        datax = self.px.ctrl.GetProfileDataAsVariant()
        datay = self.py.ctrl.GetProfileDataAsVariant()
        data = [list(row) for row in zip(datax,datay)]
        #Makes a listof lists; X1 with Y1 in a list, etc...
    filename = self.ti.Value
    with open(filename,'w')as fp:
      w = csv.writer(fp, delimiter=',')
      w.writerows(data)
```

You can get the value of most wxPython controls with their "GetStringSelection" method except for the text control. To get the value from the text control we named "ti," you should use its "Value" property. With the data exported as a .csv file, you can open it a variety of applications including Excel. Here is our code after we add this function.

```python
import wx
import wx.lib.activex
import csv

class MyApp( wx.App ):
    def OnClick(self,e):
        rb_selection = self.rb.GetStringSelection()
        if rb_selection == "WinCam":
            data = self.gd.ctrl.GetWinCamDataAsVariant()
            data = [[x] for x in data]
        else:
            p_selection = self.cb.GetStringSelection()
            if p_selection == "Profile_X":
```

```python
            data = self.px.ctrl.GetProfileDataAsVariant()
            data = [[x] for x in data]
        elif p_selection == "PRofile_Y":
            data = self.py.ctrl.GetProfileDataAsVariant()
            data = [[x] for x in data]
        else:
            datax = self.px.ctrl.GetProfileDataAsVariant()
            datay = self.py.ctrl.GetPRofileDataAsVariant()
            data = [list(row) for row in zip(datax,datay)]
            #Makes a listof lists; X1 with Y1 in a list, etc...
        filename = self.ti.Value
        with open(filename,'w')as fp:
            w = csv.writer(fp, delimiter=',')
            w.writerows(data)

def __init__( self, redirect=False, filename=None ):
    wx.App.__init__( self, redirect, filename )
    self.frame = wx.Frame( parent=None, id=wx.ID_ANY,size=(900,900),
                           title='Python Interface to DataRay')
    #Panel
    p = wx.Panel(self.frame,wx.ID_ANY)
    #Get Data
    self.gd = wx.lib.activex.ActiveXCtrl(p, 'DATARAYOCX.GetDataCtrl.1')
    self.frame.Show()
    #Button
    b1 = wx.lib.activex.ActiveXCtrl(parent=p,size=(200,50), pos=(7, 0),
                                    axID='DATARAYOCX.ButtonCtrl.1')
    b1.ctrl.ButtonID =297 #Id's for some ActiveX controls must be set

    b2 = wx.lib.activex.ActiveXCtrl(parent=p,size=(100,25), pos=(5, 55),
                                    axID='DATARAYOCX.ButtonCtrl.1')
    b2.ctrl.ButtonID =171
    b3 = wx.lib.activex.ActiveXCtrl(parent=p,size=(100,25), pos=(110,55),
                                    axID='DATARAYOCX.ButtonCtrl.1')
    b3.ctrl.ButtonID =172
    b4 = wx.lib.activex.ActiveXCtrl(parent=p,size=(100,25), pos=(5, 85),
                                    axID='DATARAYOCX.ButtonCtrl.1')
    b4.ctrl.ButtonID =177
    b4 = wx.lib.activex.ActiveXCtrl(parent=p,size=(100,25), pos=(110, 85),
                                    axID='DATARAYOCX.ButtonCtrl.1')
    b4.ctrl.ButtonID =179
    #Pictures
    tpic = wx.lib.activex.ActiveXCtrl(parent=p,size=(250,250),
                    axID='DATARAYOCX.ThreeDviewCtrl.1',pos=(500,250))
    #Profiles
    self.px = wx.lib.activex.ActiveXCtrl(parent=p,size=(300,200),
                    axID='DATARAYOCX.ProfilesCtrl.1',pos=(0,600))
    self.px.ctrl.ProfileID=22
    self.py = wx.lib.activex.ActiveXCtrl(parent=p,size=(300,200),
                    axID='DATARAYOCX.ProfilesCtrl.1',pos=(600,600))
    self.py.ctrl.ProfileID = 23

    #CCDImage
    wx.lib.activex.ActiveXCtrl(parent=p,axID='DATARAYOCX.CCDimageCtrl.1',
```
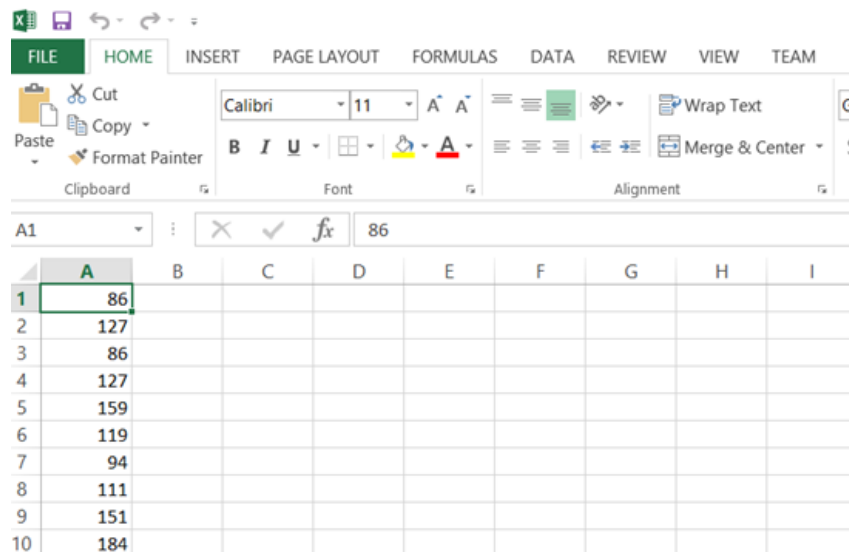
```
                                    size=(250,250),pos=(250,250))

        #Custom controls
        t = wx.StaticText(p, label="File:", pos=(5, 115))
        self.ti = wx.TextCtrl(p, value="C:/Users/Public/Documents/output.csv",
                              pos=(30, 115), size=(170, -1))
        self.rb = wx.RadioBox(p, label="Data:", pos=(5, 140),
                              choices=["Profile", "WinCam"])
        self.cb = wx.ComboBox(p, pos=(5,200),
                              choices=[ "Profile_X", "Profile_Y", "Both"])
        self.cb.SetSelection(0)
        myb = wx.Button(p, label="Write", pos=(5,225))
        myb.Bind(wx.EVT_BUTTON, self.OnClick)
        self.gd.ctrl.StartDriver()

if __name__ == "__main__":
    app = MyApp()
    app.MainLoop()
```

**EVENTS**



Besides communicating through interfaces to the DataRayOCX, there is a system of events which allow the DataRayOCX to communicate back. One of the most useful events is the GetData control's DataReady event. The comtypes library makes it fairly convenient to listen to events. First we need to import the relevant section of the comtypes library into our program:

```
        import comtypes.client
```

The object which listens for events is called a sink. To have custom behavior, we need to write our own. We will make our program listen for the DataReady event which fires whenever the program has a new frame. It signals that new data is available.

```
class EventSink(object):
  def __init__(self,frame):
    self.counter=0
```

```
        self.frame = frame
    def DataReady(self):
        self.counter +=1
        self.frame.Title= "DataReady fired {0} times".format(self.counter)
```

For comtypes to make the connection between the sink's methods and the OCX events, the names of the sink's methods must be similar to the corresponding events.  In this case, the name needs to include "DataReady."  It is not necessary to give the sink access to any of the app's properties, but in order to give our sink access to the title property of our app's frame, we have added a frame argument to its initialization. It is necessary, however, to set the connection object created by comtypes as a property of our app; we do this with "self.sink =". We can add it to MyApp's initialization anywhere after the definition of the frame and the GetData ActiveX control:

```
self.gd = wx.lib.activex.ActiveXCtrl(p, 'DATARAYOCX.GetDataCtrl.1')
 self.gd.ctrl.StartDriver()
#The methods of the object are available through the ctrl property of the item
sink = EventSink(self.frame)
 self.sink = comtypes.client.GetEvents(self.gd.ctrl, sink)
```

We pass the component we wish to listen to and the sink into comtypes.client.GetEvents function and attach it to our MyApp class.  Now, every time the DataReady event fires, the title of our program's window will change: Finally here is the end result of our gui.

```
import wx
import wx.lib.activex
import csv
import comtypes.client



#Class that handles the event handling
class EventSink(object):
    def __init__(self,frame):
        self.counter=0
        self.frame = frame
    def DataReady(self):
        self.counter +=1
        self.frame.Title= "DataReady fired {0} times".format(self.counter)
class MyApp( wx.App ):
    def OnClick(self,e):
        rb_selection = self.rb.GetStringSelection()
        if rb_selection == "WinCam":
            data = self.gd.ctrl.GetWinCamDataAsVariant()
            data = [[x] for x in data]
        else:
            p_selection = self.cb.GetStringSelection()
            if p_selection == "Profile_X":
                data = self.px.ctrl.GetProfileDataAsVariant()
                data = [[x] for x in data]
            elif p_selection == "PRofile_Y":
                data = self.py.ctrl.GetProfileDataAsVariant()
                data = [[x] for x in data]
            else:
                datax = self.px.ctrl.GetProfileDataAsVariant()
                datay = self.py.ctrl.GetPRofileDataAsVariant()
                data = [list(row) for row in zip(datax,datay)]
```

```python
        #Makes a listof lists; X1 with Y1 in a list, etc...
        filename = self.ti.Value
        with open(filename,'w')as fp:
            w = csv.writer(fp, delimiter=',')
            w.writerows(data)

def __init__( self, redirect=False, filename=None ):
    wx.App.__init__( self, redirect, filename )
    self.frame = wx.Frame( parent=None, id=wx.ID_ANY,size=(900,900),
                          title='Python Interface to DataRay')
    #Panel
    p = wx.Panel(self.frame,wx.ID_ANY)
    #Get Data
    self.gd = wx.lib.activex.ActiveXCtrl(p, 'DATARAYOCX.GetDataCtrl.1')
    self.frame.Show()

    #EventSink
    sink = EventSink(self.frame)
    self.sink = comtypes.client.GetEvents(self.gd.ctrl,sink)
    #Button
    b1 = wx.lib.activex.ActiveXCtrl(parent=p,size=(200,50), pos=(7, 0),
                              axID='DATARAYOCX.ButtonCtrl.1')
    b1.ctrl.ButtonID =297 #Id's for some ActiveX controls must be set

    b2 = wx.lib.activex.ActiveXCtrl(parent=p,size=(100,25), pos=(5, 55),
                              axID='DATARAYOCX.ButtonCtrl.1')
    b2.ctrl.ButtonID =171
    b3 = wx.lib.activex.ActiveXCtrl(parent=p,size=(100,25), pos=(110,55),
                              axID='DATARAYOCX.ButtonCtrl.1')
    b3.ctrl.ButtonID =172
    b4 = wx.lib.activex.ActiveXCtrl(parent=p,size=(100,25), pos=(5, 85),
                              axID='DATARAYOCX.ButtonCtrl.1')
    b4.ctrl.ButtonID =177
    b4 = wx.lib.activex.ActiveXCtrl(parent=p,size=(100,25), pos=(110, 85),
                              axID='DATARAYOCX.ButtonCtrl.1')
    b4.ctrl.ButtonID =179

    #Pictures
    tpic = wx.lib.activex.ActiveXCtrl(parent=p,size=(250,250),
                    axID='DATARAYOCX.ThreeDviewCtrl.1',pos=(500,250))
    #Profiles
    self.px = wx.lib.activex.ActiveXCtrl(parent=p,size=(300,200),
                    axID='DATARAYOCX.ProfilesCtrl.1',pos=(0,600))
    self.px.ctrl.ProfileID=22
    self.py = wx.lib.activex.ActiveXCtrl(parent=p,size=(300,200),
                    axID='DATARAYOCX.ProfilesCtrl.1',pos=(600,600))
    self.py.ctrl.ProfileID = 23

    #CCDImage
    wx.lib.activex.ActiveXCtrl(parent=p,axID='DATARAYOCX.CCDimageCtrl.1',
                        size=(250,250),pos=(250,250))

    #Custom controls
    t = wx.StaticText(p, label="File:", pos=(5, 115))
```

```python
        self.ti = wx.TextCtrl(p, value="C:/Users/Public/Documents/output.csv",
                            pos=(30, 115), size=(170, -1))
        self.rb = wx.RadioBox(p, label="Data:", pos=(5, 140),
                            choices=["Profile", "WinCam"])
        self.cb = wx.ComboBox(p, pos=(5,200),
                            choices=[ "Profile_X", "Profile_Y", "Both"])
        self.cb.SetSelection(0)
        myb = wx.Button(p, label="Write", pos=(5,225))
        myb.Bind(wx.EVT_BUTTON, self.OnClick)
        self.gd.ctrl.StartDriver()

if __name__ == "__main__":
    app = MyApp()
    app.MainLoop()
```

## 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:

```python
wx.lib.activex.ActiveXCtrl(self.frame,size=(400,400),axID='DATARAYOCX.TwoDCtrl.1')
```
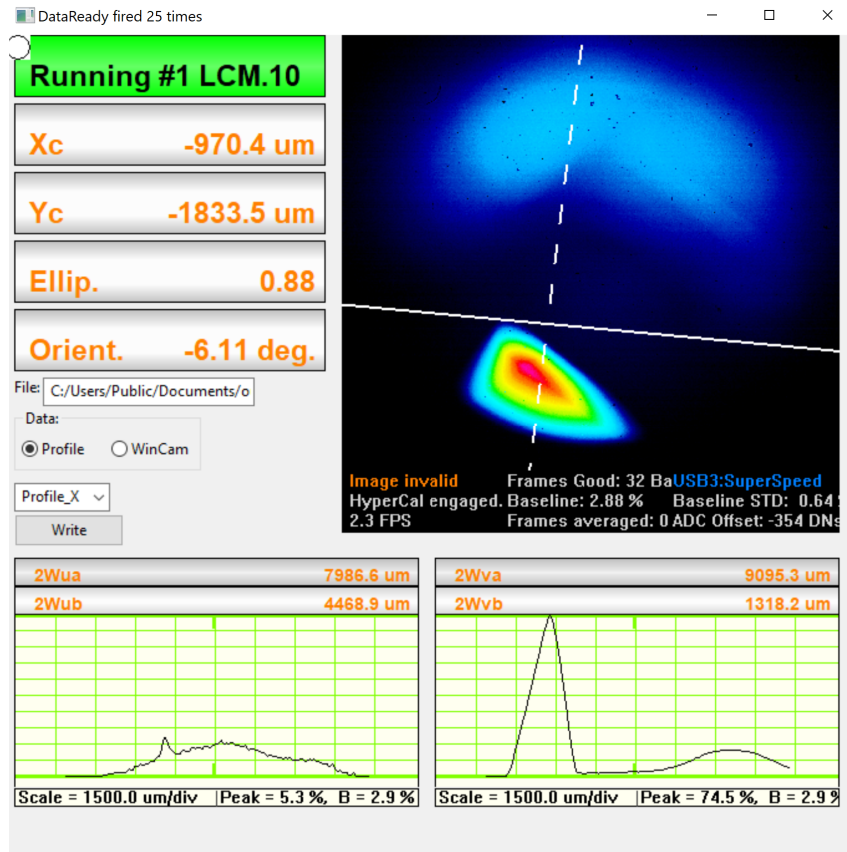
This completes the advanced tutorial! **Problems/Questions?** Please contact us with the information listed above.