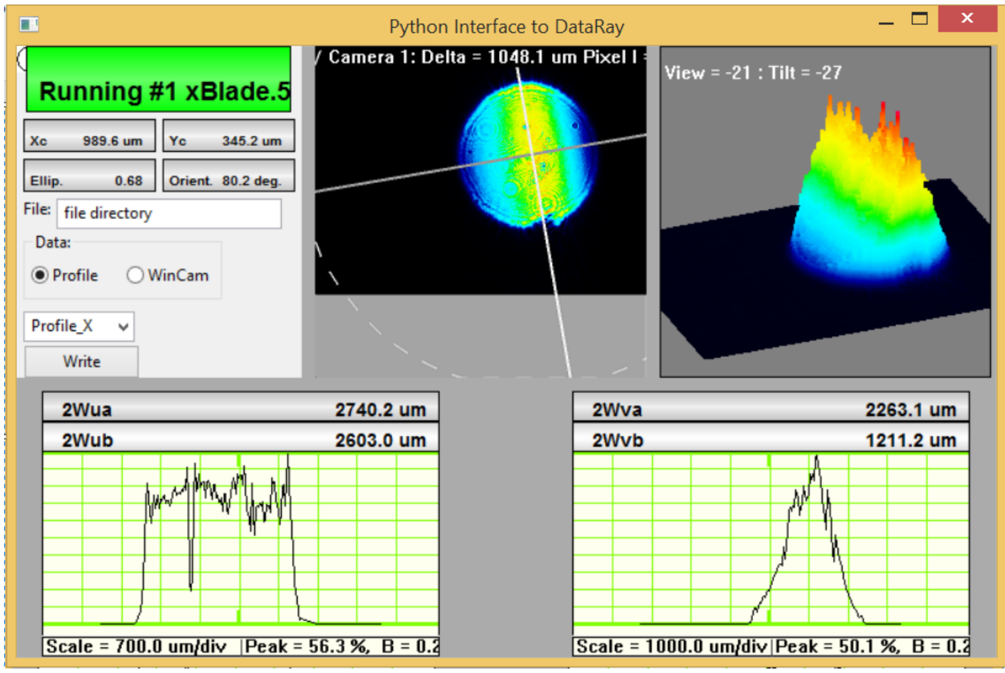
support@dataray.com

1675 Market Street
Redding, CA 96001

+1 530 395 2500

Interfacing to Python

OVERVIEW:



Python Interface to DataRay
Running #1 xBlade.5
Camera 1: Delta = 1048.1 um Pixel I
View = -21 : Tilt = -27
Write

Xc 989.6 um
Yc 345.2 um
Ellip. 0.68
Orient. 80.2 deg.
File: file directory
Data: [✓ Profile] WinCam
Profile_X

2Wua 2740.2 um
2Wub 2603.0 um
Scale = 700.0 um/div Peak = 56.3 %, B = 0.2

2Wva 2263.1 um
2Wvb 1211.2 um
Scale = 1000.0 um/div Peak = 50.1 %, B = 0.2

GETTING STARTED:

INTERFACING WITH 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. This tutorial is for a WinCam; if you are using a different device, some of the ID's and ActiveX controls will be different.

SOME IMPORTANT NOTES:

- The OCX is functional only as part of a *GUI-based program*. In this tutorial, we use the wxPython library.
- Since the OCX is 32-bit, you will need associated 32-bit Python and libraries
- 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.
- 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.

Second we need to install the associated Python software:

- Install a 32-bit version of Python which supports both wxPython and its library of widgets (2.6 or 2.7); 2.7 is recommended
- Install wxPython from http://wxpython.org/download.php
- Install the comtypes library with pip or setuptools; most versions of Python now install pip for you. Using the command line, go to the "Scripts" directory of your Python installation and type "pip install comtypes".

You can download the interface developed in this tutorial. It exists as a solution file in Visual Studio 2013, but it has no dependencies from Visual Studio. You could work on the Python file in any editor.

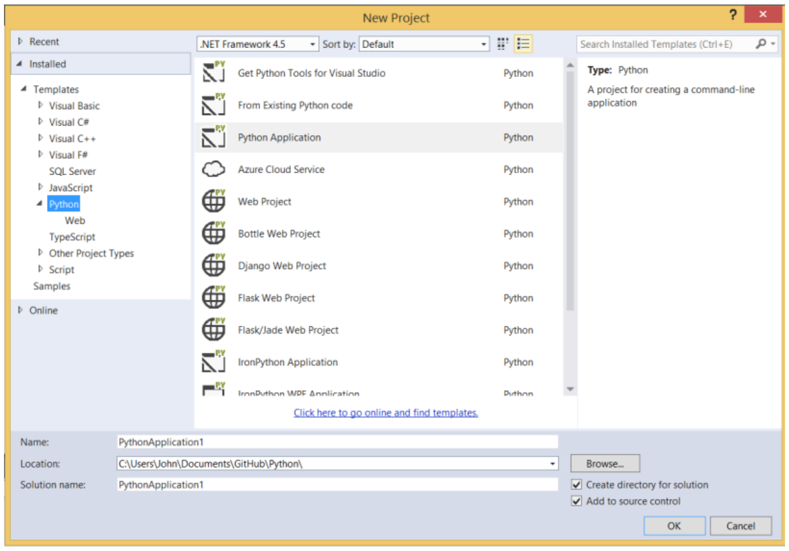
- **Cameras**: Download & unzip: http://www.dataray.com/assets/software/WinCamDPython.zip
- **BeamR2:** Download & unzip: http://www.dataray.com/assets/software/BeamRPython.zip
- **BeamMap2:** Download & unzip: http://www.dataray.com/assets/software/BeamMapPython.zip

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 which you are using.

BASIC TUTORIAL:

We will show you step-by-step how the example program was created. If you are using Visual Studio, you should install Python tools. After doing so, you can select to create a new Python Application from the project creation menu.


New Project
Recent
Installed
Templates
Visual Basic
Visual C#
Visual C++
Visual F#
SQL Server
JavaScript
Python
Web
TypeScript
Other Project Types
Script
Samples
Online
.NET Framework 4.5
Sort by: Default
Search Installed Templates (Ctrl+E)
Get Python Tools for Visual Studio — Python
From Existing Python code — Python
Python Application — Python
Azure Cloud Service — Python
Web Project — Python
Bottle Web Project — Python
Django Web Project — Python
Flask Web Project — Python
Flask/Jade Web Project — Python
IronPython Application — Python
IronPython WPF Application — Python
Click here to go online and find templates.
Browse...
OK
Cancel

Type: Python

A project for creating a command-line application

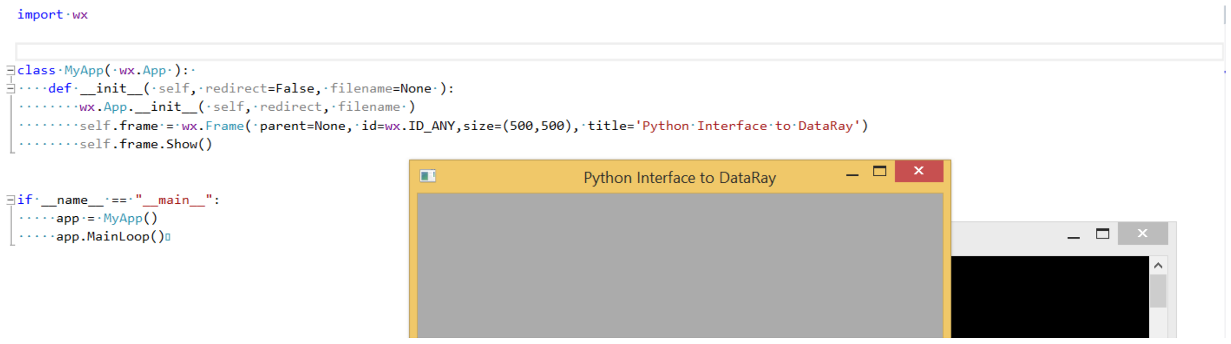
Name: PythonApplication1
Location: C:\Users\John\Documents\GitHub\Python\
Solution name: PythonApplication1
[✓] Create directory for solution
[✓] Add to source control

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:

```
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=(500,500), title='Python Interface to DataRay')
        self.frame.Show()

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

support@dataray.com
1675 Market Street
Redding, CA 96001
+1 530 395 2500



```
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=(500,500), title='Python Interface to DataRay')
        self.frame.Show()

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

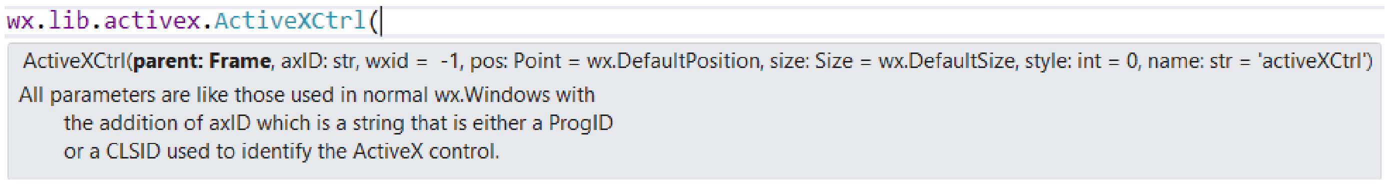
Without using the frame's "Show" method, the window will not be visible. The black window created in the background is an instance of the Python interpreter. Any print statements will output to that window.

FIRST BUTTON AND IMAGE

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

```
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.

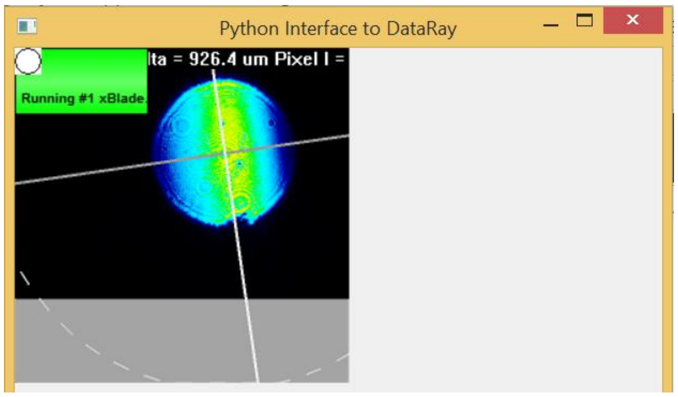
As a minimum, the ActiveXCtrl class needs the parent and axID (ActiveXID) arguments to initialize, but it can also take arguments for size and position. The parent argument must either be a frame or be a wx item which has a frame as its parent, grandparent, etc…

```
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=(500,500), 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.gd.ctrl.StartDriver()
```

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'). With newer cameras, you must add a palette bar as well ('DATARAYOCX.PaletteBarCtrl.1'). We will add both of them to our panel. The size argument takes a tuple of the form (horizontal pixels, vertical pixels):

```
#Button
b1 = wx.lib.activex.ActiveXCtrl(parent=p,size=(100,50),axID='DATARAYOCX.ButtonCtrl.1')
b1.ctrl.ButtonID =297 #Id's for some ActiveX controls must be set
#CCDImage
wx.lib.activex.ActiveXCtrl(parent=p,size=(250,250),axID='DATARAYOCX.CCDimageCtrl.1')
wx.lib.activex.ActiveXCtrl(parent=self.frame,size=(10,250), axID='DATARAYOCX.PaletteBarCtrl.1')
```

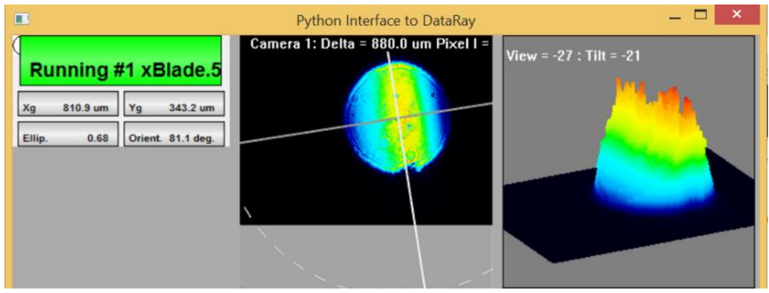


Since both items were added into the same panel without position arguments, they will overlap because they both start from the upper left corner, position (0,0) of the panel. This is obviously problematic for interfaces. The wxPython library supplies many ways to make formatting interfaces easier.

LAYOUT WITH wxPYTHON

Entire tutorials dedicated to formatting wxPython interfaces are available on the library's wiki and in books. This section will explain the basics. First, we will create a separate panel for buttons with manually specified sizes and positions. We will also move the start button (#297) from the panel it shares with GetData ("p") to our new, separate button panel ("bp"):

```
#Button Panel
bp = wx.Panel(parent=self.frame, id=wx.ID_ANY, size=(215, 110))
b1 = wx.lib.activex.ActiveXCtrl(parent=bp,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=bp,size=(100,25), pos=(5, 55),axID='DATARAYOCX.ButtonCtrl.1')
b2.ctrl.ButtonID =171
b3 = wx.lib.activex.ActiveXCtrl(parent=bp,size=(100,25), pos=(110,55),axID='DATARAYOCX.ButtonCtrl.1')
b3.ctrl.ButtonID =172
b4 = wx.lib.activex.ActiveXCtrl(parent=bp,size=(100,25), pos=(5, 85),axID='DATARAYOCX.ButtonCtrl.1')
b4.ctrl.ButtonID =177
b4 = wx.lib.activex.ActiveXCtrl(parent=bp,size=(100,25), pos=(110, 85),axID='DATARAYOCX.ButtonCtrl.1')
b4.ctrl.ButtonID =179
```



For emphasis, we made the start button twice the width and height of the other buttons which display values. To allow all the smaller buttons to fit with 5 pixel borders, 15 horizontal pixels and 60 pixels (25+25+5+5) were added on to start button size to determine the panel size.

In addition to manually setting positions, wxPython provides Sizers. Sizers let us add items to an interface without calculating exact positions. We will be using BoxSizers instead of GridSizers. BoxSizers can either be vertical or horizontal. When they are initialized, their orientation can be set as follows:

`row1 = wx.BoxSizer(wx.HORIZONTAL)` **or** `col1 = wx.BoxSizer(wx.VERTICAL)`

BoxSizers add objects to a horizontal or vertical space with borders and alignments. We will start with adding two pictures with 10 pixel borders to the same row as the button panel. The pictures will be 250x250, so the window needs to be set to new horizontal size of about 760 pixels wide:

```
#Pictures
pic = wx.lib.activex.ActiveXCtrl(parent=self.frame,size=(250,250),axID='DATARAYOCX.CCDimageCtrl.1')
tpic = wx.lib.activex.ActiveXCtrl(parent=self.frame,size=(250,250), axID='DATARAYOCX.ThreeDviewCtrl.1')
palette=wx.lib.activex.ActiveXCtrl(parent=self.frame,size=(10,250), axID='DATARAYOCX.PaletteBarCtrl.1')
#Formatting
row1 = wx.BoxSizer(wx.HORIZONTAL)
row1.Add(item=bp,flag=wx.RIGHT, border=10)
row1.Add(pic)
row1.Add(item=tpic, flag=wx.LEFT, border=10)
self.frame.SetSizer( row1 )
```

Along with the item to be added, the BoxSizer takes flags which specify the borders or alignments and the border in pixels. It also takes a proportion argument, but that is beyond the scope of this tutorial. After adding all the items to the BoxSizer, we need to set the sizer to the frame. More border types can be specified with wx.TOP, wx.BOTTOM, and wx.ALL. Flags can be combined by adding the OR operator (|). To add another row to display profiles, we will need another horizontal BoxSizer and a vertical one to combine the two.

First we create the profiles and set their ID's. They are set as properties because we will want to access them later.

```
#Profiles
self.px = wx.lib.activex.ActiveXCtrl(parent=self.frame,size=(300,200),axID='DATARAYOCX.ProfilesCtrl.1')
self.px.ctrl.ProfileID=22
self.py = wx.lib.activex.ActiveXCtrl(parent=self.frame,size=(300,200),axID='DATARAYOCX.ProfilesCtrl.1')
self.py.ctrl.ProfileID = 23
```

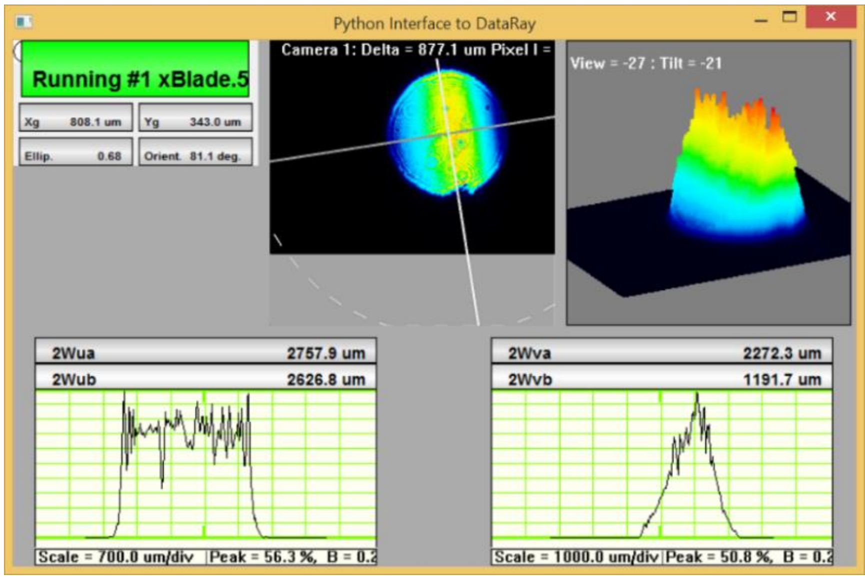
Next, we will add them to a second horizontal BoxSizer with a 100 pixel space between the two.

```
row2 = wx.BoxSizer(wx.HORIZONTAL)
row2.Add(self.px, 0, wx.RIGHT, 100)# Arguments: item, proportion, flags, border
row2.Add(self.py)
```

Finally, we add both rows to the vertical BoxSizer and set it to the frame.

```
col1 = wx.BoxSizer(wx.VERTICAL)
col1.Add(item=row1, flag=wx.BOTTOM, border=10)
col1.Add(item=row2, flag=wx.ALIGN_CENTER_HORIZONTAL)
self.frame.SetSizer(col1)
```

support@dataray.com
1675 Market Street
Redding, CA 96001
+1 530 395 2500

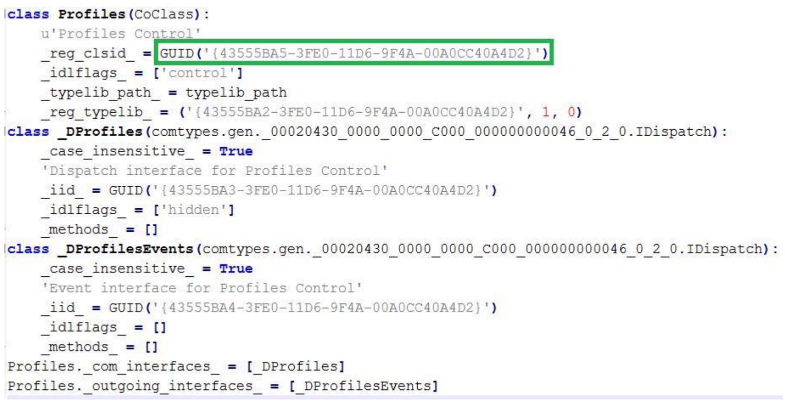

Python Interface to DataRay
Running #1 xBlade.5
Xg 808.1 um
Yg 343.0 um
Ellip. 0.68
Orient. 81.1 deg.
Camera 1: Delta = 877.1 um Pixel I =
View = -27 : Tilt = -21
2Wua 2757.9 um
2Wub 2626.8 um
Scale = 700.0 um/div Peak = 56.3 %, B = 0.2
2Wva 2272.3 um
2Wvb 1191.7 um
Scale = 1000.0 um/div Peak = 50.8 %, B = 0.2

Even though col1's orientation is vertical, it will still let us use the positioning flags to place our profiles in the horizontal center of the frame. There are many position flags: wx.ALIGN_LEFT, wx.ALIGN_RIGHT, wx.ALIGN_BOTTOM, wx.ALIGN_CENTER_VERTICAL, wx.ALIGN_CENTER_HORIZONTAL, and wx.ALIGN_CENTER. The end result is a fairly compact GUI of just the essential elements.

.

EXTENDING THE GUI

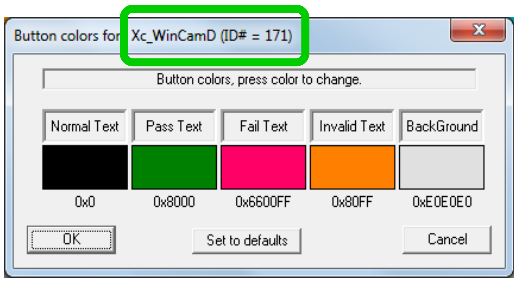
There are additional ActiveX controls available. When wxPython's activex library is used for the first time, comtypes is used to generate a file with interfaces for all of the ActiveX controls. You can find the generated file inside your Python directory under "\Lib\site-packages\comtypes\gen". As of 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: "DATARAYOCX.{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_".



```
class Profiles(CoClass):
    u'Profiles Control'
    _reg_clsid_ = GUID('{43555BA5-3FE0-11D6-9F4A-00A0CC40A4D2}')
    _idlflags_ = ['control']
    _typelib_path_ = typelib_path
    _reg_typelib_ = ('{43555BA2-3FE0-11D6-9F4A-00A0CC40A4D2}', 1, 0)
class _DProfiles(comtypes.gen._00020430_0000_0000_C000_000000000046_0_2_0.IDispatch):
    _case_insensitive_ = True
    'Dispatch interface for Profiles Control'
    _iid_ = GUID('{43555BA3-3FE0-11D6-9F4A-00A0CC40A4D2}')
    _idlflags_ = ['hidden']
    _methods_ = []
class _DProfilesEvents(comtypes.gen._00020430_0000_0000_C000_000000000046_0_2_0.IDispatch):
    _case_insensitive_ = True
    'Event interface for Profiles Control'
    _iid_ = GUID('{43555BA4-3FE0-11D6-9F4A-00A0CC40A4D2}')
    _idlflags_ = []
    _methods_ = []
Profiles._com_interfaces_ = [_DProfiles]
Profiles._outgoing_interfaces_ = [_DProfilesEvents]
```

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



Button colors for Xc_WinCamD (ID# = 171)

Button colors, press color to change.

Normal Text	Pass Text	Fail Text	Invalid Text	BackGround
⬛	🟩	🟥	🟧	⬜
0x0	0x8000	0x6600FF	0x80FF	0xE0E0E0

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

Buttons: http://www.dataray.com/assets/pdf/IndexToTestParametersEnum.pdf

Profiles: http://www.dataray.com/assets/pdf/ProfilesEnum.pdf

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

ADVANCED TUTORIAL:

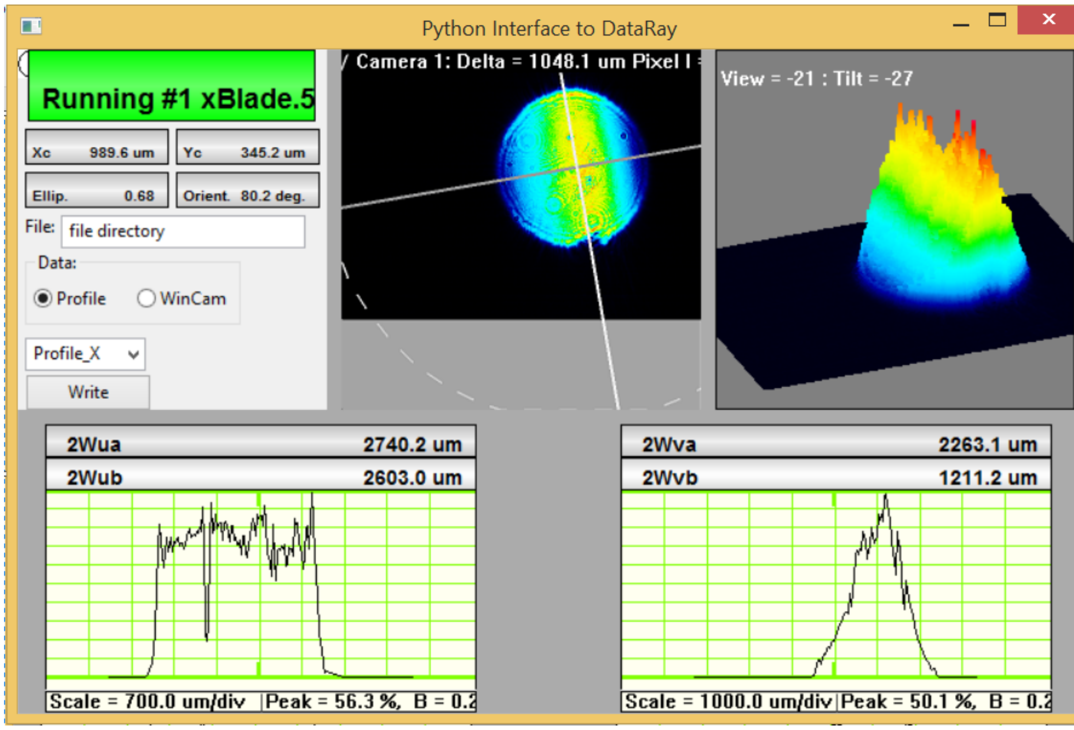
wxPYTHON CONTROLS

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.

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


Python Interface to DataRay
Running #1 xBlade.5
Camera 1: Delta = 1048.1 um Pixel I
View = -21 : Tilt = -27
Write
Xc 989.6 um
Yc 345.2 um
Ellip. 0.68
Orient. 80.2 deg.
File: file directory
Data: [✓ Profile] WinCam
Profile_X
2Wua 2740.2 um
2Wub 2603.0 um
2Wva 2263.1 um
2Wvb 1211.2 um
Scale = 700.0 um/div |Peak = 56.3 %, B = 0.2
Scale = 1000.0 um/div |Peak = 50.1 %, B = 0.2

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 all we have written so far 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:

```
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.

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:

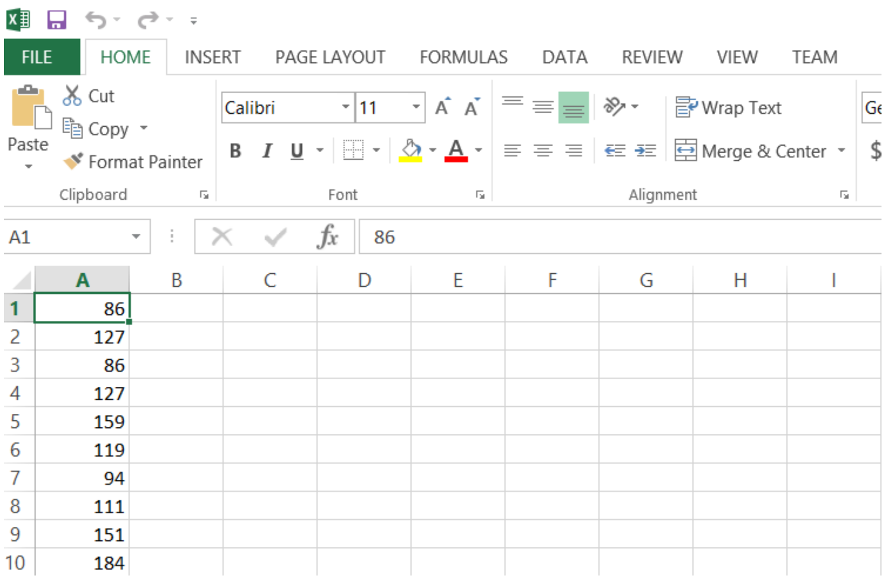
```
import csv
```

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.

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

support@dataray.com
1675 Market Street
Redding, CA 96001
+1 530 395 2500

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.



	A
1	86
2	127
3	86
4	127
5	159
6	119
7	94
8	111
9	151
10	184

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.

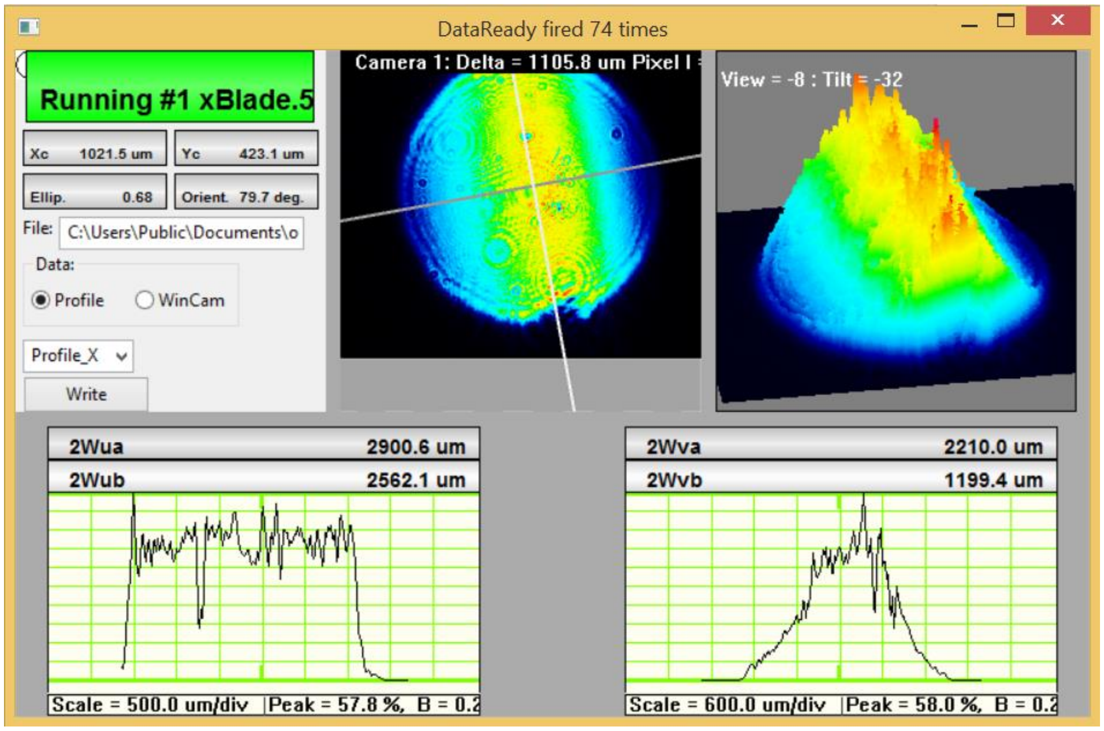
```
class EventSink(object):

    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)
```

support@dataray.com
1675 Market Street
Redding, CA 96001
+1 530 395 2500

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:



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