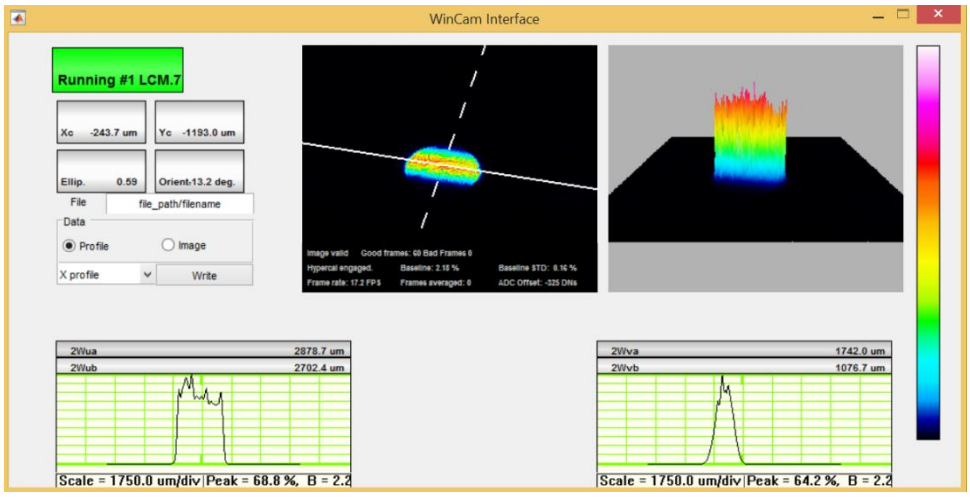
support@dataray.com
1675 Market Street
Redding, CA 96001
+1 530 395 2500

Interfacing to MATLAB



OVERVIEW:

- Getting Started
  - Interfacing with OCX
  - Installation
- Basic Tutorial
  - GUI with MATLAB's GUIDE
  - First Button & Image
  - More ActiveX Controls
  - Extending the GUI
- Advanced Tutorial
  - MATLAB Controls
  - Getting Data
  - Events

GETTING STARTED:

INTERFACING WITH OCX

Your interfacing code communicates with DataRay products through the DataRay OCX. The OCX has ActiveX controls 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 or the GUI in MATLAB's GUIDE. Unfortunately, while this characteristic of the OCX is useful for creating interfaces in some languages, it causes MATLAB to crash, and as a result, ActiveX components must be added to a GUI interface programmatically. Also, for reasons unknown, an interface can only be run once per MATLAB session.

SOME IMPORTANT NOTES:

- The OCX is functional only as part of a *GUI-based program*. In this tutorial, we use MATLAB's .fig and GUIDE.
- Since the OCX is 32-bit, you will need associated 32-bit MATLAB and libraries
- The OCX and DataRay program cannot be used at the same time

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.

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

You can download the interface developed in this tutorial. It exists as a collection of 3 MATLAB files.

- **Cameras**: Download & unzip WinCamD
- **BeamMap2:** Download & unzip BeamMap2

This example should build and run with no errors. Not working? Email support@dataray.com or call 530-395-2500 with:

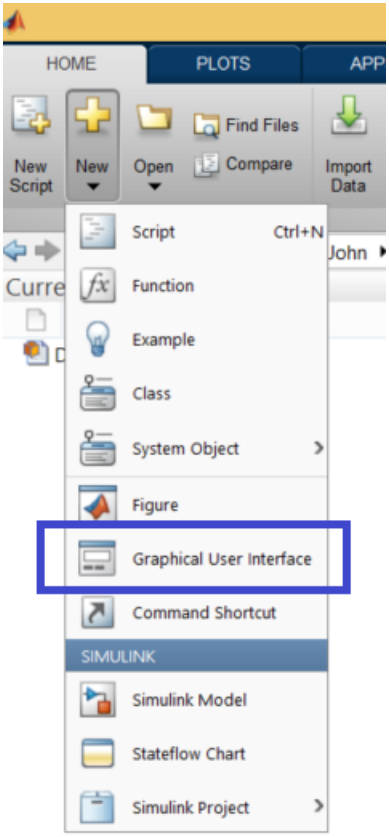
- Device name and serial number
- DataRay and Windows versions which you are using.

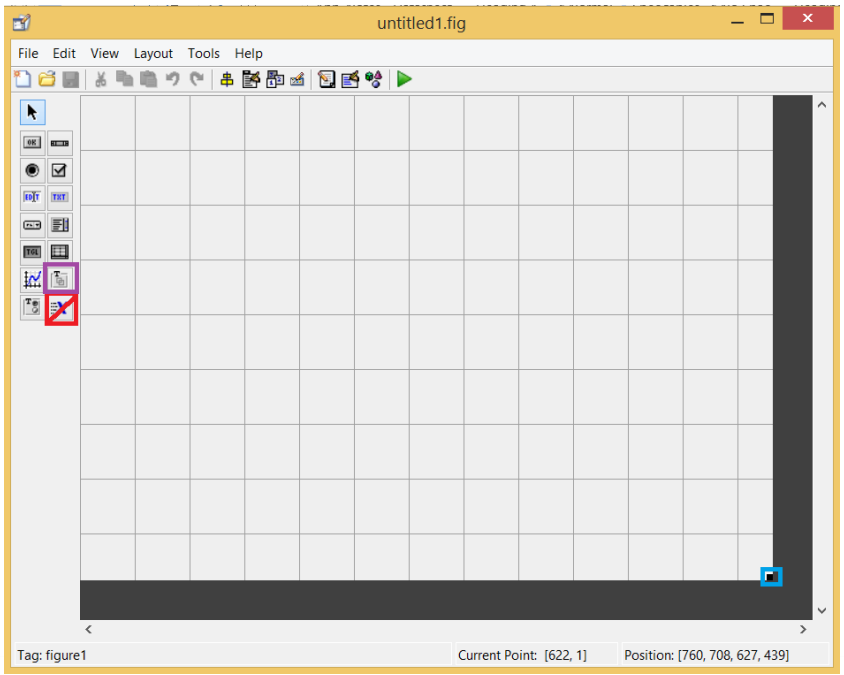
BASIC TUTORIAL:

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

GUI WITH MATLAB'S GUIDE

First, we will make a basic GUI with GUIDE. It supports all the items you would expect from a GUI library. In a new folder for the 3 files which will compose the interface, under the **HOME** tab click **Graphical User Interface** under the **New** button.

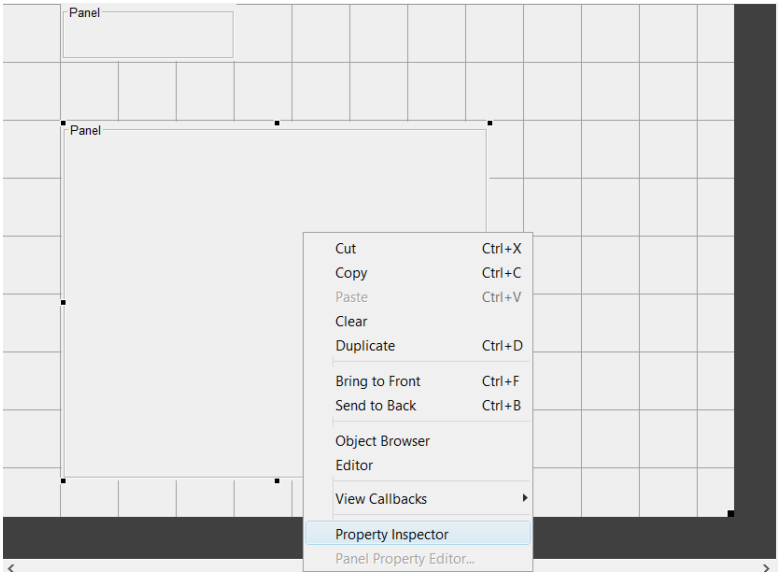




For our basic interface, we will only require adding **Panels**. If you need more space, you can click the bottom corner and drag to increase the size of your interface. Although we are using ActiveX controls, do not attempt to use **ActiveX Control**. The other GUI interface components are fine to use, and we will cover their use in the Advanced Tutorial section. To create the most basic interface, we will add 3 panels for the start button, the 2D image of the beam and the Palette. The Dataray class will create the fourth required GetData control.

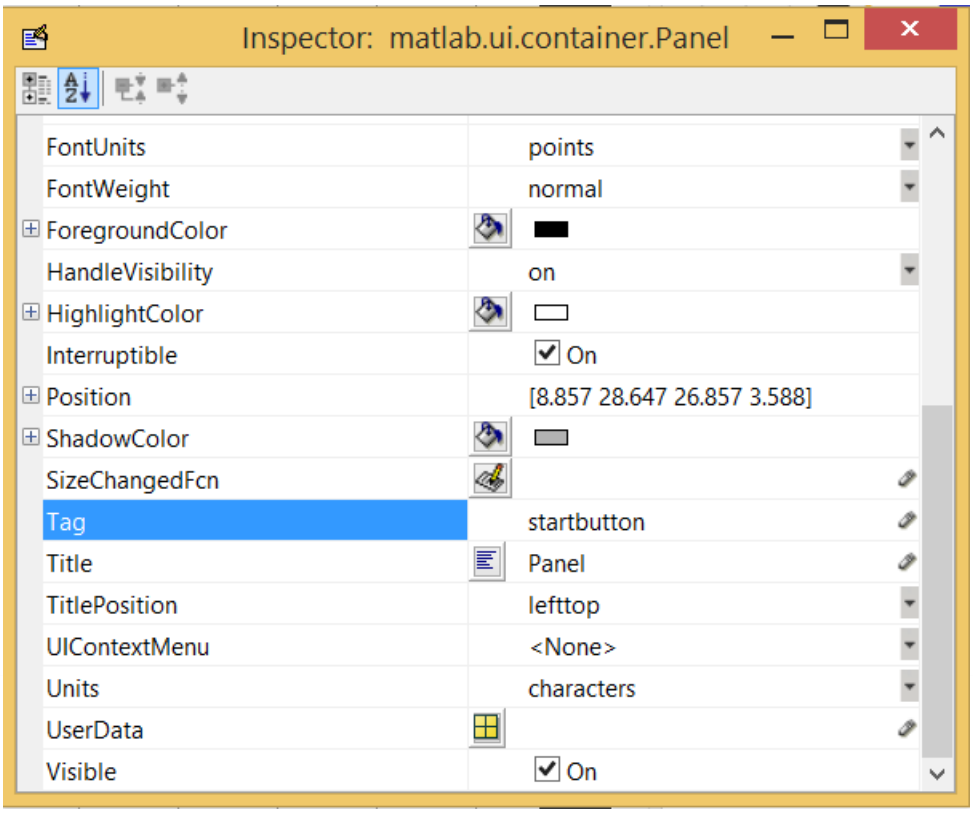
support@dataray.com
1675 Market Street
Redding, CA 96001
+1 530 395 2500

Right-click on panel and then click and drag to establish the panel's size (and thereby establish the ActiveX Control's size). Once the panels which will hold the Active X Controls have been placed, you may left-click them and open up the Property Inspector to select convenient names for them such as "startbutton" which will make it more straight-forward to programmatically set their ActiveX controls for them.



The Property Inspector allows you to change many aspects of the selected GUI component including its "Tag" which can be used by MATLAB to interact with it programmatically. In order for changes in the Property Inspector to be saved, you must minimize it and save the GUI; do not simply exit the Property Inspector. If the new tag value is not saved, it will cause the GUI to fail.

Once you save the figure, it will create a .fig file and a .m file which will work together to create the GUI. We will edit the .m file to make it create ActiveX controls.



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

FIRST BUTTON & IMAGE

To add ActiveX components, we use the handles from the GUI to instantiate them. The syntax required to instantiate ActiveX control objects is as follows; you must use the **OCX control name**, **a series of raw coordinates OR those derived from the handle specified by a tag name**, and **a handle for the figure itself**:

```
getDataCtrl = actxcontrol('DATARAYOCX.GetDataCtrl.1',[0,0,1,1],handles.figure1);

CCDctrl
=actxcontrol('DATARAYOCX.CCDimageCtrl.1',getpixelposition(handles.twod),handles.fig
ure1);
```

The ActiveX controls are vital for communication to and from the instance of the DataRay program created by the interface. For example, the GetDataCtrl control must be present for the OCX to start and its "StartDriver" method must be called for devices to be recognized.

```
getDataCtrl.StartDriver();
```

To make the controls accessible by other methods of our GUI class, we will add the Dataray class from Dataray.m to our GUI object which will hold all of the ActiveX controls. The reason for doing this will become clearer in the Advanced Tutorial. When the Dataray class is created, the getDataCtrl object is created and the "StartDriver" method is called from its member variable referring to the GetData ActiveX controls like so:

```
dObj.getDataCtrl.StartDriver();
```

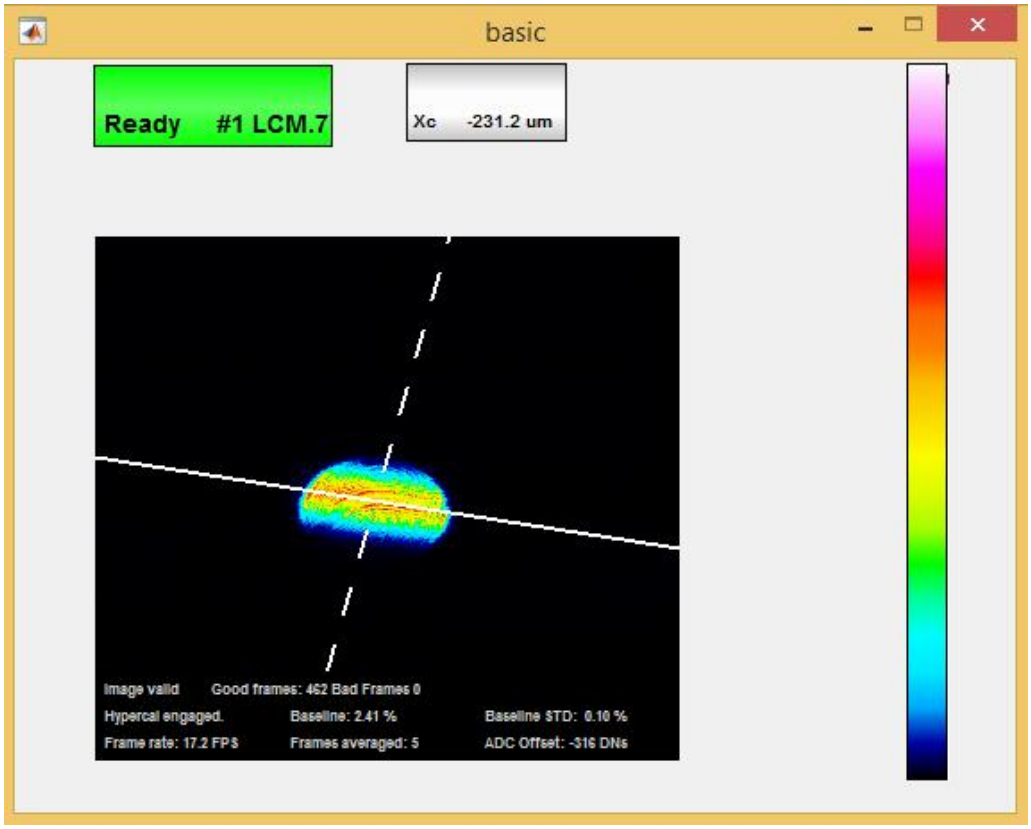
The Dataray class has a useful function for setting buttons to the GUI and storing their respective ActiveX controls into an array:

```
% set button on top of existing uipanel
function setButtonPanel(dObj,panel_handle,buttonID)
    % create button actxcontrol and place on top of the panel
    dObj.btnCtrls{end+1} =
actxcontrol('DATARAYOCX.ButtonCtrl.1',getpixelposition(panel_handle),dObj.currentFi
gure);
    dObj.btnCtrls_handles{end+1} = panel_handle; % save handle to panel in an array
    set(dObj.btnCtrls{end},'ButtonID',buttonID); % set the ButtonID member variable
end
```

The above method on the Dataray class does all of the work for adding buttons to the interface. Adding the following to your GUI's opening function will create the ActiveX components for the interface; make sure to have the handles added after they have been created:

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

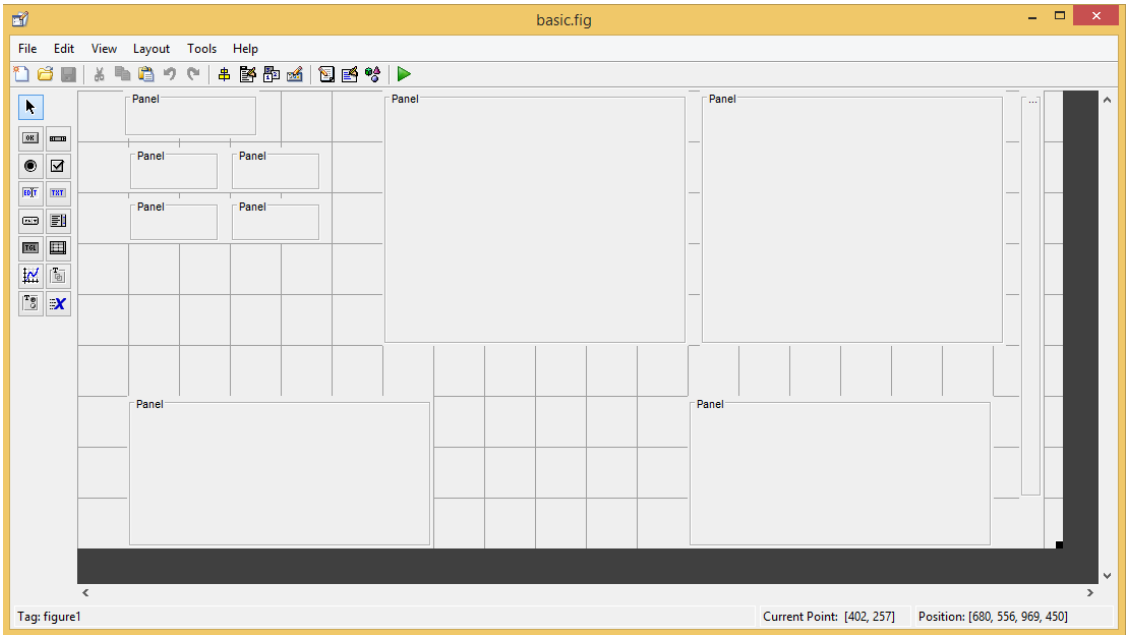
```
function basic_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to basic (see VARARGIN)

% Choose default command line output for basic
handles.output = hObject;

handles.DATARAY = Dataray(handles.figure1); % create DATARAY class
handles.DATARAY.setCCDpanel(handles.twod)
handles.DATARAY.setButtonPanel(handles.startbutton,297); % set start button
handles.DATARAY.setButtonPanel(handles.button1,171);
handles.DATARAY.setPalettePanel(handles.palette);


% Update handles structure
guidata(hObject, handles);
```



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

MORE ACTIVEX CONTROLS

We will now add more buttons and profiles to the interface. Adding profiles not only helps display information, but also it provides the functionality to pull data from the camera. We need to go back to our GUI and add more panels.



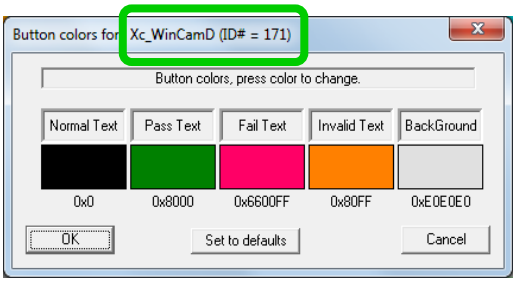
The initialization code to initialize this GUI is as follows:

```
handles.DATARAY = Dataray(handles.figure1); % create DATARAY class
handles.DATARAY.setCCDpanel(handles.twod)
handles.DATARAY.set3DPanel(handles.threed)
handles.DATARAY.setButtonPanel(handles.startbutton,297); % set start button
handles.DATARAY.setButtonPanel(handles.button1,171); % set start button
handles.DATARAY.setButtonPanel(handles.button2,172); % set start button
handles.DATARAY.setButtonPanel(handles.button3,177); % set start button
handles.DATARAY.setButtonPanel(handles.button4,179); % set start button
handles.DATARAY.setProfilesPanel(handles.profile1,22); % set start button
handles.DATARAY.setProfilesPanel(handles.profile2,23); % set start button
handles.DATARAY.setPalettePanel(handles.palette);
```

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

EXTENDING THE GUI

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:

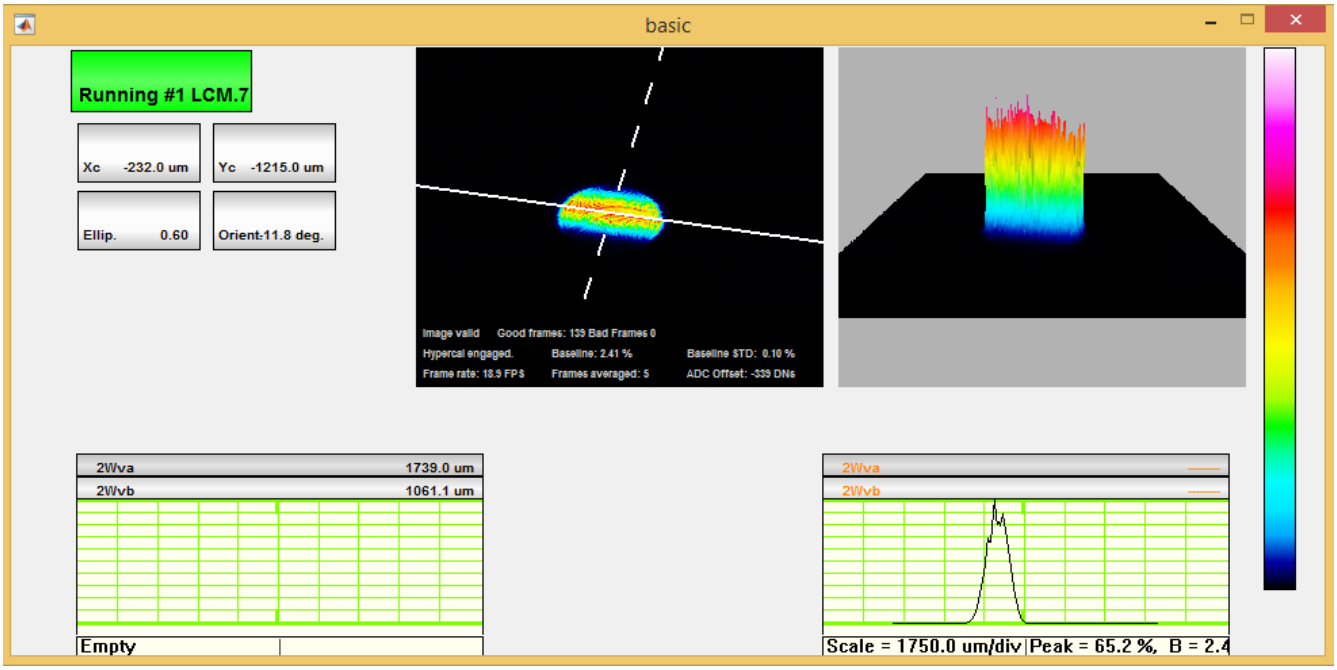
Profiles: http://www.dataray.com/UserFiles/file/ProfilesEnum.pdf

Buttons: http://www.dataray.com/UserFiles/file/IndexToTestParametersEnum.pdf

Finally, there is documentation describing the methods and properties of all the ActiveX controls:

http://www.dataray.com/assets/pdf/OCXDocumentation.pdf

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



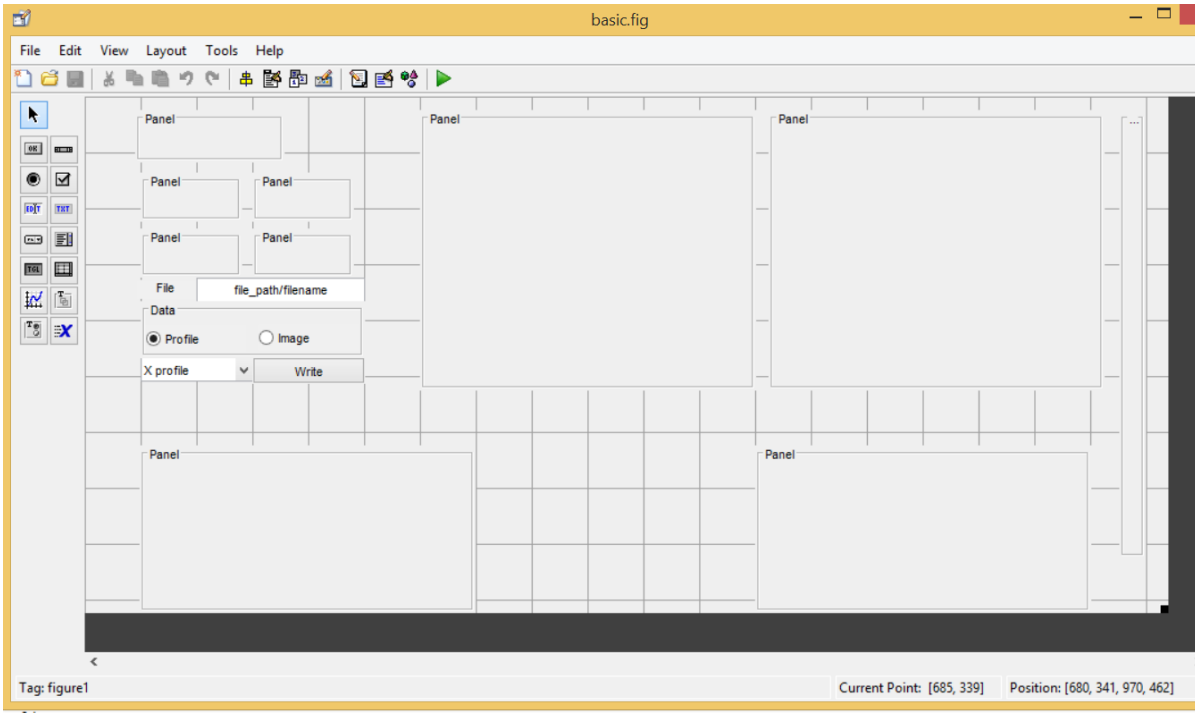
support@dataray.com
1675 Market Street
Redding, CA 96001
+1 530 395 2500

ADVANCED TUTORIAL:

MATLAB CONTROLS

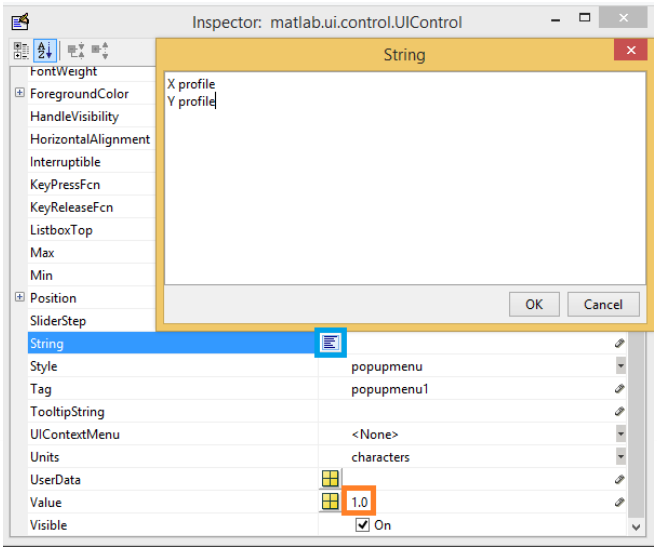
MATLAB provides its own controls and input methods. These can be used to provide custom functionality to your GUI. In this case, we will be using them to select data to write to a file.

First, we will add one static text label, one edit text control (text input), two radio buttons in a button group, one pop-up meu (a dropdown list) and one button to our GUI with GUIDE:



The button group component is like a panel, but it makes radio buttons operate as a group; only one radio button in a button group can be selected. GUI components can have their properties set programmatically, but in this tutorial, we will use the Property Inspector to set them.

String:	• For most components, the String property will set the text it displays • For the pop-up menu, we will want to click the list display of the String property and enter one choice per line
Value	• For radio buttons, 1 is true and 0 is false • For pop-up menus, 1 is the first item, 2 is the second item, etc…



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

Entire tutorials have been written about events and binding in MATLAB. We will stick to the basics. The GUIDE will create functions which will be called on interactions with the various GUI components. You can specify more actions to take when, for instance, the component is created or destroyed by right-clicking on the element and selecting view callbacks. In this case, we will be working with one which is created automatically:

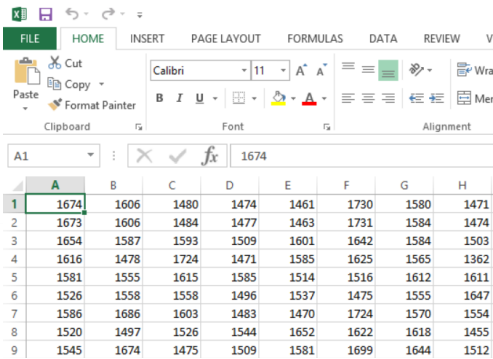
```
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%contents = cellstr(get(handles.popupmenu1,'String'));
fprintf( 'Option %d selected \n', get(handles.popupmenu1,'Value'));
fprintf('%s \n', handles.DATARAY.btnCtrls{2}.GetParameter());
```

The above prints out two items when the “Write” button is hit. First, it prints out the item selected in the pop-up menu by its position. The second item we are printing is the value of the second ActiveX button control we added to the interface; the first is the start button. In general, the syntax to get the value of a GUI component is get(handles.[component’s tag], “Property name”). GUIDE also prints out useful tips near the callbacks of items.

GETTING DATA

Now that we have an understanding of how MATLAB GUI components can be set up, we will rewrite the callback method to output data to a comma separated value file with the csvwrite function. Whenever you are writing a file it is crucial to make sure you have write privileges for the directory by either selecting a public directory or running the program as administrator. With the following code, we can write .csv files to be analyzed in MATLAB or Excel.

```
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
%contents = cellstr(get(handles.popupmenu1,'String'));
fname = get(handles.edit1,'String')
if get(handles.radiobutton1,'Value') %this is for profiles
    axis = get(handles.popupmenu1,'Value');
    data = handles.DATARAY.profileCtrls{axis}.GetProfileDataAsVariant();
    csvwrite(fname, data);
else % this is for image data
    data = handles.DATARAY.getDataCtrl.GetWinCamDataAsVariant();
    denom = 2;
    if
handles.DATARAY.getDataCtrl.CaptureIsFullResolution();
        denom = 1;
    end
    display(denom)
    m = handles.DATARAY.getDataCtrl.GetHorizontalPixels()
/denom;
    n = handles.DATARAY.getDataCtrl.GetVerticalPixels()
/denom;
    image_data = reshape(data,m,n)
    csvwrite(fname, image_data);
end
```



	A	B	C	D	E	F	G	H
1	1674	1606	1480	1474	1461	1730	1580	1471
2	1673	1606	1484	1477	1463	1731	1584	1474
3	1654	1587	1593	1509	1601	1642	1584	1503
4	1616	1478	1724	1471	1585	1625	1565	1362
5	1581	1555	1615	1585	1514	1516	1612	1611
6	1526	1558	1558	1496	1537	1475	1555	1647
7	1586	1686	1603	1483	1470	1724	1570	1554
8	1520	1497	1526	1544	1652	1622	1618	1455
9	1545	1674	1475	1509	1581	1699	1644	1512

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

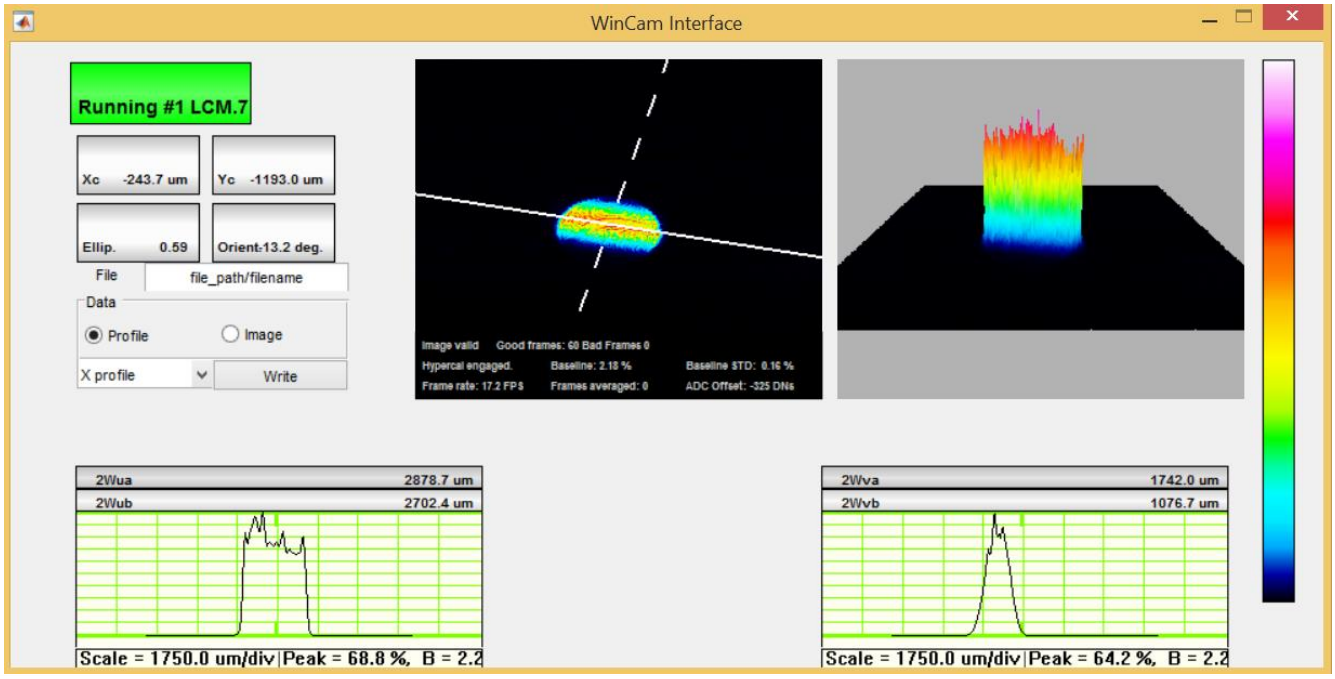
Both methods return a 1-D matrix of data. Therefore, if you wish to have all the values of the raw pixels in their proper locations from the GetWinCamDataAsVariant method, it is necessary to reshape the matrix.

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 SendMessage event which is used for internal communication in the DataRay program; most users are interested in the message with ID # 21 as its 3rd argument which is used whenever a new frame is available.

The object which listens for events is called a sink. We make a function and pass it to the GetData control as a callback for the SendMessage event in the Dataray object:

```
    handlerMessage = @dObj.eventCallback;
    dObj.getDataCtrl.registerevent({'SendMessage',handlerMessage});
end

function eventCallback(dObj,varargin)
    if varargin{3} == 21
        dObj.eventcounter = dObj.eventcounter +1;
        display(sprintf('%d frames seen', dObj.eventcounter));
    end
end
```

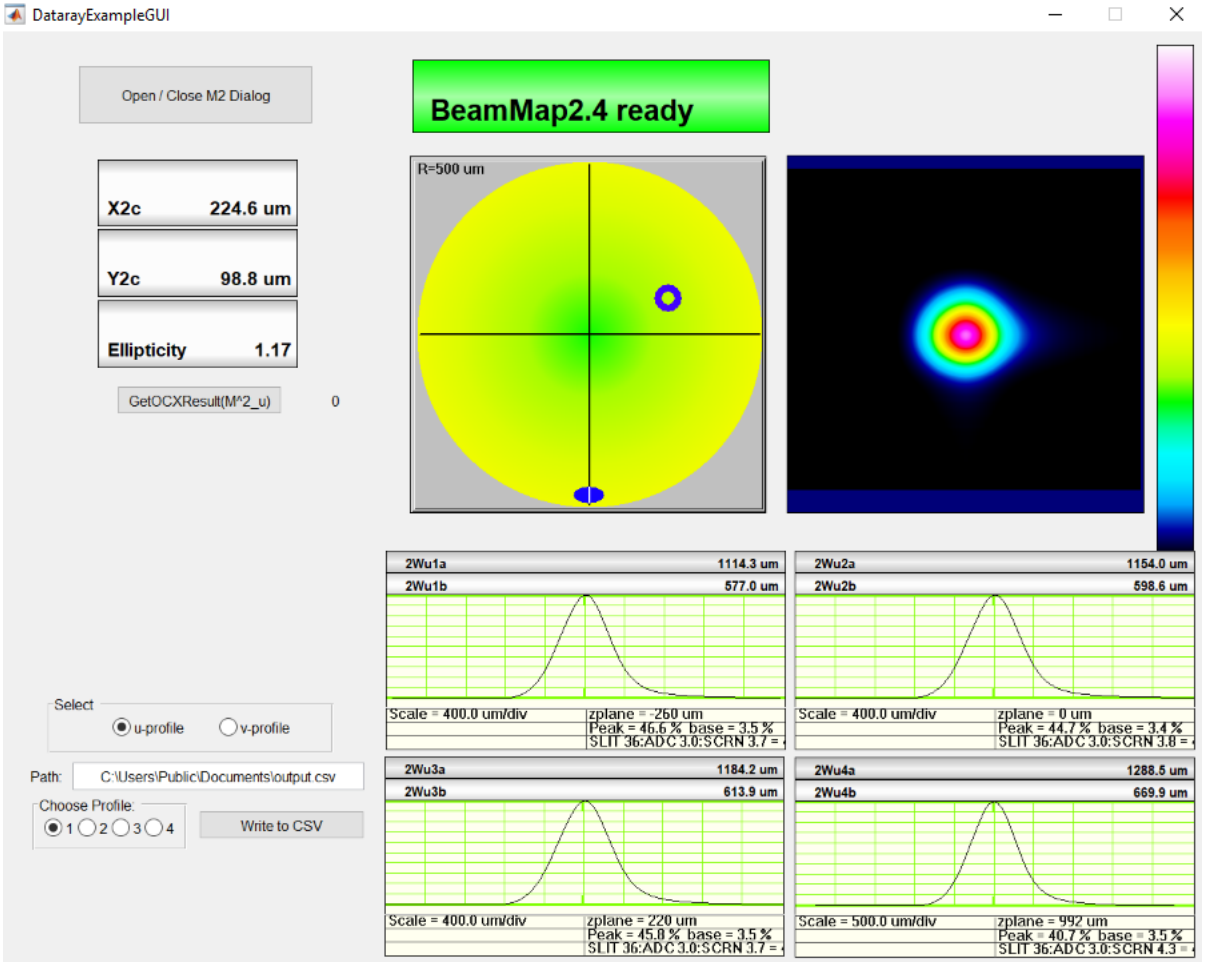


This will print out the total number of frames seen by incrementing a counter. Many customers have found this useful for running data analysis routines on every x number of frames.

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

BeamMap2 Tutorial:

You can download the interface developed in this tutorial as a collection of 3 MATLAB files from the link that is available on the first page of this document. Please build and run this example before you continue. The example should be fully functional and appear like the image displayed below.



This tutorial is brief because it references many of the techniques described in the Basic and Advanced WinCam tutorials; however, we will be using Button ID#s, Profile ID#s and functions unique to the BeamMap2. Please make yourself familiar with the other tutorials before attempting to recreate this example.

Creating DataRay Button Controls

1) Create a **GetData Control** and a **Status Button (ButtonID# = 104)** Button Control by using the instructions on pages 1-4 of the Basic WinCam Tutorial.
2) Unlike the WinCam tutorials, this example does not use a **ccdImage Control**. Instead, this example uses a **TwoD Control.** Follow the instructions on pages 3-4 of the Basic WinCam tutorial, but create and use the **setTwoDPanel** function (shown below) instead of the setCCDPanel function:

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

```
function setTwoDpanel(dObj,panel_handle)
            % create CCD actxcontrol and place on top of the panel
            dObj.TwoDctrl =
actxcontrol('DATARAYOCX.TwoDCtrl.1',getpixelposition(panel_handle),dObj.currentFigure);
end
```

3) Create the **X2c (ButtonID# = 104)**, **Y2c (ButtonID# = 105)**, and **Ellipticity (ButtonID# = 126)** Button Controls by using the same steps you used to create the Status Button, but by substituting in the appropriate ButtonID#s.

4) Create the **Wander Display** for tracking the centroid position by creating a Button Control and assigning it **Button ID# 197.**

CREATING DATARAY PROFILE CONTROLS

5) Create four unique Profile Controls by following the instructions on page 5 of the Basic WinCam Tutorial. Instead of using ProfileID#s 22 and 23, which correspond to the profiles along the WinCamD crosshairs, you will need to use ProfileID#s that apply to the BeamMap2. The example program uses the following commands to set the profiles upon initialization:

```
handles.DATARAY.setProfilesPanel(handles.uipanelProfile,14); % set U1 prof.
handles.DATARAY.setProfilesPanel(handles.uipanelProfile2,16); % set U2 prof.
handles.DATARAY.setProfilesPanel(handles.uipanelProfile3,18); % set U3 prof.
handles.DATARAY.setProfilesPanel(handles.uipanelProfile4,20); % set U4 prof.
```

Please note that these ProfileID#s (14, 16, 18, 20) correspond to the u-profiles at each of the four z-positions on the BeamMap2 puck. We will create radio buttons that allow the user to toggle between the u and v-profiles. The v-profiles have their own ProfileID#s (15, 17, 19, 21).

6) Create the **Radio Buttons** that are used to select the u or v-profiles by using the methods discussed on the first page of the Advanced WinCam tutorial. Make sure the default radio button selected corresponds to the default profile setting, which is the u-profile in this example. We assigned the tag **radiobuttonProfileU** to the u-profiles radio button to check its state in the following function. This function executes when the user-selected radio is changed in the button group:

```
function uibuttongroup1_SelectionChangedFcn(hObject, eventdata, handles)
% hObject    handle to the selected object in uibuttongroup1
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

profile_actxControl = handles.DATARAY.getProfileCtrl(handles.uipanelProfile); % get activeX
control variable to directly access its Parameters (or Functions)
if get(handles.radiobuttonProfileU,'value') == 1
    set(profile_actxControl,'MyID',14); % set profile U
else
    set(profile_actxControl,'MyID',15); % set Profile V
end
profile_actxControl = handles.DATARAY.getProfileCtrl(handles.uipanelProfile2);
if get(handles.radiobuttonProfileU,'value') == 1
    set(profile_actxControl,'MyID',16); % set profile U
else
    set(profile_actxControl,'MyID',17); % set Profile V
end
profile_actxControl = handles.DATARAY.getProfileCtrl(handles.uipanelProfile3);
if get(handles.radiobuttonProfileU,'value') == 1
    set(profile_actxControl,'MyID',18); % set profile U
else
    set(profile_actxControl,'MyID',19); % set Profile V
end
profile_actxControl = handles.DATARAY.getProfileCtrl(handles.uipanelProfile4);
if get(handles.radiobuttonProfileU,'value') == 1
    set(profile_actxControl,'MyID',20); % set profile U
else
    set(profile_actxControl,'MyID',21); % set Profile V
end
```

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

USING DATARAY'S GETDATA CONTROL

7) Next, we will use the **GetData Control** to open the M^2 Dialog and programmatically extract data. First, we need to create a way to access the **GetData Control's** methods from the MATLAB file that contains all the GUI methods. Since the **GetData Control** object belongs to the DataRay class, we will create a function in the DataRay class that returns access to the **GetData Control** object**:**

```
% get the activeX control of a specified panel or panel number to get access to further functions
of this control
function actxCtrl = getGetDataCtrl(dObj)
    % get activeX control to get access to get data ctrl functions
    actxCtrl = dObj.getDataCtrl;
end
```

This function is used in the callback functions that belong to the **Open/Close M2 Dialog Button** and **GetOCXResult(M^2_u)** buttons.

8) Create a **Push Button** using MATLAB's GUIDE. In the example, we assigned this button the tag **m2Button** and the string **Open/Close M2 Dialog**. We use our function from step 7 to gain access to the **GetData Control.** We use the **GetData Control** to set change the state of its member **IsMSquaredOpen**, which effectively opens and closes the M2 Dialog. To use a single button to close and open the M2 Dialog, the button's callback function should be as follows:

```
% --- Executes on button press in m2Button.
function m2Button_Callback(hObject, eventdata, handles)
% hObject    handle to m2Button (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
getData_actxControl = handles.DATARAY.getGetDataCtrl(); % get activeX control variable to
directly access its Parameters (or Functions)
if (getData_actxControl.IsMSquaredOpen == 0)
    getData_actxControl.IsMSquaredOpen = 1; %opens the M2 dialog
else
    getData_actxControl.IsMSquaredOpen = 0; %closes the M2 dialog
end
```

9) Similarly, we will create another **Push Button** using MATLAB's GUIDE and use the **GetData Control** in its callback function to get the result of the M^2 calculation at the instant the button is clicked. In the example we used **getOCXResultButton** as the tag and **GetOCXResult(M^2_u)** for the string of the button. Use MATLAB's GUIDE to create a **Static Text** element to display the result. We used **textOCXResult** as the tag for the Static Text element. The **GetData Control's GetOcxResult()** function accepts any **ButtonID#** as its parameter and returns the value that would be calculated and displayed on the corresponding Button Control. This callback function is used to display the result of the M^2 calculation in the Static Text:

```
% --- Executes on button press in getOCXResultButton.
function getOCXResultButton_Callback(hObject, eventdata, handles)
% hObject    handle to getOCXResultButton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
getData_actxControl =  handles.DATARAY.getGetDataCtrl(); % get activeX control variable to
directly access its Parameters (or Functions)
result = getData_actxControl.GetOcxResult(156); %156 is the index/ID that applies to m^2_u
set(handles.textOCXResult,'String', result);
```

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

EXPORTING PROFILE DATA TO CSV

10) Next, we will use GUIDE to create a group of four **Radio Buttons**. These elements will allow the user to choose any of the four profiles that are currently displayed and export the profile's data. In the example program, we used the radio button tags **radio1**, **radio2**, **radio3**, and **radio4**.
11) In addition, we will use the GUIDE to create an **Edit Control** with the tag **editCSVPath** and string **C:\Users\Public\Documents\output.csv**. This string of this edit control contains the path where the file will be generated and saved when we write to CSV.
12) Finally, we create a **Push Button** with tag **writeToCSVButton** and string **Write to CSV**. This button's callback function will write the selected profile (identified by the state of the four radio buttons and the u-profile / v-profile radio buttons) to a CSV. Here is the callback function:

```
% --- Executes on button press in writeToCSVButton.
function writeToCSVButton_Callback(hObject, eventdata, handles)
% hObject    handle to writeToCSVButton (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
path = get(handles.editCSVPath,'String');
if get(handles.radio1,'value') == 1 %1 means it is selected
    profile_actxControl = handles.DATARAY.getProfileCtrl(handles.uipanelProfile); % get activeX
control variable to directly access its Parameters (or Functions)
end
if get(handles.radio2,'value') == 1 %1 means it is selected
    profile_actxControl = handles.DATARAY.getProfileCtrl(handles.uipanelProfile2); % get
activeX control variable to directly access its Parameters (or Functions)
end
if get(handles.radio3,'value') == 1 %1 means it is selected
    profile_actxControl = handles.DATARAY.getProfileCtrl(handles.uipanelProfile3); % get
activeX control variable to directly access its Parameters (or Functions)
end
if get(handles.radio4,'value') == 1 %1 means it is selected
    profile_actxControl = handles.DATARAY.getProfileCtrl(handles.uipanelProfile4); % get
activeX control variable to directly access its Parameters (or Functions)
end
data = profile_actxControl.GetProfileDataAsVariant();
csvwrite(path,data);
```

The **Profile Control's GetProfileDataAsVariant** returns the current profile data as a variant array.

This completes the BeamMap2 tutorial! **Problems/Questions?** Please contact us with the information listed above.