A GUI for Viewing Depth Profile XPS Images

Laura Blair

February 26, 2002
Contents

1 Problem Statement 2
2 Explanation of Programs 4
3 Flowchart 7
4 Variables 10
5 User’s Guide 15
6 Testing Procedure 18
7 Program Listing 29
Chapter 1

Problem Statement

X-ray Photoelectron Spectroscopy (XPS) is used by Bausch and Lomb to evaluate the surface chemistry of various products. One XPS method is collection of elemental maps which indicate the location and intensity of a particular element. When a depth profile is conducted, the surface of the sample is sputtered (to remove the top few nanometers) and an elemental map is collected. This process is repeated numerous times resulting in a set of maps. The current graphical user interface only allows interpretation of each image individually. This can become quite cumbersome if the depth profile consists of numerous layers with a variety of elemental maps acquired per layer potentially resulting in 100+ total images. The only way to simultaneously view all of these images is to print them each out and lay them out on a table. The objective of the program written for this project was to display elemental maps collected during a depth profile in a way that eases interpretation.

To create the program a set of sample data had to be acquired. Instead of using a proprietary product, a hard disk was used so the results could be shared with the class. A defective spot was located on the hard disk so the elemental maps would indicate regions with different chemistries. A typical hard disk has five layers of coatings on the substrate. Based on a prior depth profile conducted by PHI a general estimation of the thickness of each layer was calculated. Three different sputter settings were utilized. Each sputtered at a different rate and were used in appropriate layers of the hard disk. The sample was sputtered 13 times. After each sputter a survey was collected to determine what elements were present. Maps were then acquired for the elements found in the survey. The maps were opened in Multipak and then exported as TIFF files onto a CD so they could be read by the IDL program at RIT.

Initially the goal was to put all of the two-dimensional images into one three-dimensional image. Later it was decided that trying to represent 100+ maps in one three-dimensional image would result in too much information to be interpreted. Instead it would be useful to allow the user to view different selections of the 100+ images. For example viewing all of the maps collected at a particular depth would give the user an indication of the chemistry at that
depth. Or allowing simultaneous viewing of all maps of a particular element at the full range of depths would allow the user to observe how the intensity of the element is changing throughout the layers. A third idea was to open the images at a rate it appears the sputter is happening. This allows the user to visualize what the chemistry looks like as the depth profile occurs.

When a programmer dedicates a lot of time to writing code program they want to ensure that the program is used more than once. Since not all samples will have the same elements or the same number of depths it was important to make sure the program didn’t have to be re-written each time a new depth profile was conducted. The code had to allow a variable number of elements and depths. The program also had to be able to recognize changes in the names of the elements or values of the depths. If these goals were not achieved the program would require extensive rewriting before different samples could be analyzed.
Chapter 2

Explanation of Programs

Two programs and two functions were written to accomplish the goals stated in Chapter 1. Both programs and functions were written in IDL using its available widget toolkit. No mathematical computations are performed by either program. Instead they are used to display images appropriately so the user can interpret the results. A description of what each function and program is expected to do will be included in this chapter.

The purpose of the program titled final1image is to open all of the depths of a particular element at a rate that it appears that the sputter is happening. The idea is that this will allow the user to visualize the chemistry change as the depth into the sample increases. If their is a bright region in the center of a map to begin with (indicating high intensity of that element) and it decreases as depth increases it will be easily apparent using this program. Sometimes it is difficult to see differences when two images are placed next to each other but when opened one after the other it is possible to see how the pixels change. This program will give the user a feel for how the chemistry is changing as the depth profile progresses. Since the time each image is viewed is short to give the feeling that the sputter is happening, it is not possible to focus in on the details of any one image. Therefore two text fields were created. One text field accepts the elemental symbol and the other accepts the layer number. The user can type values into each text field and the image of interest will be displayed. For example if the user types c into the text field labeled 'Type in Element Symbol' and then 2 into the text field labeled 'Type in layer number', the carbon image collected after the second sputter will be displayed. It will stay in the image window until another entry is made or a button is pushed by the user. Therefore the user can look at this particular image as long as desired.

This program is also versatile. The element buttons are generated by a text file that the user creates. This is powerful because it means that one person could collect a depth profile for 10 elements and a second person could collect a depth profile with a different number of sputters for 15 different elements. Each person can display their images using this program without any changes to the code. Therefore the user does not have to know anything about IDL to run the
program. The images are automatically resized, cropped, opened and labeled without any need to change the code. They will be accurately labeled since the label is based on the text files created by the user.

If a map is not collected the program will print a generic image indicating to the user that this element was not collected at this depth. The program queries the filename to determine if the file exists and then attempts to open the file only if it does exist. Without this part of the code, the program would crash when attempting to open a file that doesn’t exist. Instead, a generic image is displayed and the program continues to run.

An improvement that could be made to final1image.pro would be adding a slider. After all of the images are opened it would be nice if the user could slide through to look at different images. The text fields do allow the user to look at a specific image but a slider would allow the user to scroll through the images and find which image to study.

The second program is titled finalmanyimage.pro. The most obvious difference between this program and the first one described is that it has 12 windows as opposed to one. The idea is that the user can simultaneously open 12 images and compare them to each other. Their are two options the user has for viewing images- by depth and by element. If the user selects a depth button twelve of the elemental maps collected at that depth will be displayed. This allows the user to see the elemental chemistry at a particular depth. When an element button is chosen the different depths for that element are displayed. Therefore interpretation of how the intensity of that particular element changes with depth is possible.

A compound button widget is generated by the program and the names of the buttons are the element symbols and depth values that the user included in the two text files. The user can select either an element button or a depth button. If the user selects an element button the event.value that is returned by the button press is used to determine the appropriate element for which twelve depths will be opened. The code was written to account for the fact that the user may collect more than twelve depths. Often when a depth profile is collected different elements are detected at different layers. For example nickel may not be observed for depths 0-10 but is observed for depths 11-15. When the nickel button is selected by the user the program will attempt to open nickel for depths 0-11 (the first twelve windows). First it will determine if nickel exists for depth 0 by checking for the filename (ni.tif) in the depth0 folder. When it is determined that ni.tif does not exist in this folder it will check the depth12 folder where ni.tif will be found. The image will be displayed in the first window. This process will repeat for the next three windows and images for depths 13-15 will be displayed. A generic image will be placed in the fifth window because ni.tif does not exist for depth 3 or depth 16. The same will be true for the remaining windows with the exception of the very last window. In the last window the image for the eleventh depth will be shown.

An example can also be given for the case in which there are more elements than 12 when a depth button was chosen. For example it is possible twelve elements are observed at depths 0-10 but at depth 11 carbon, which was detected
before, is no longer observed and nickel, which was not observed in the first layers, is now detected. When the user selects depth 0, the initial twelve elements will be displayed. The same will be true for depths 1-10. Now when the user selects depth 11 the program will attempt to display carbon but when the filename is queried it will determine carbon wasn’t collected for this depth. The program will attempt to display the 13th element, nickel, in the place of carbon. The other 11 elements will display as they had for the previous layers. Therefore not only does the code check to make sure the filename exists before attempting to open the image but if the filename doesn’t exist it attempts to open a different image. If there aren’t enough elements that it is necessary to open a different image then a generic image is printed indicating that a map was not collected.

In certain situations the feature described above may be misleading for the user. Assume nickel was observed at depth 0 and depth 12. The map of nickel at depth 0 will be displayed in window 1 and the map at depth 12 will not be displayed at all (when the nickel button is selected). Therefore the user must be careful to also check the different depth buttons. When depth 12 is selected the image of nickel at that depth will be displayed even though it wasn’t displayed when the nickel button was chosen.

The images are resized, cropped and labeled when displayed. The label is based on the text files created by the user. This program is versatile, like the finalimage program, since the buttons are also based on text files.

The code will not need changing to display elemental maps from depth profiles that contain different elements symbols, depths values, number of elements and number of depths.

An improvement that could be made to this program would be allowing the user to save the image display. Currently, to put the display in another document the user must alt-print scrn and then paste into the appropriate document.

The two functions that were created for this project are used by the programs to read in the text files. One function reads in the text file that contains the number of elements and the elemental symbols whereas the other function is for the text file with the number of depths and the depth values. The functions are titled read_positionelement.pro and read_positiondepth.pro.

The user creates a text file called element.dat and one called depth.dat. The text files are both commented to indicate what information needs to be entered and how it should be entered. The functions strip the comments and only read in the important information. Lines that are commented begin with the # symbol. Therefore the functions strip out the commented lines and save lines that don’t begin with # to the output file. Next the program reads the output file and creates a string array of the depths (or elements). This array is saved as depth (or element) and is used to create the buttons and labels in the two programs.
Chapter 3

Flowchart

The two figures located in this chapter are the flowchart for each program written. The flowcharts are included to give an overall view of the process. They describe the general steps in the program.
Figure 3.1: This is the flowchart for the program final1image.
Figure 3.2: This flowchart is for the finalmanyimage program.
Chapter 4

Variables

This chapter will consist of variable lists for each program and function and what the variables represent.

First the variable list for the `read_positiondepth` function.

original_string = a string variable for the comment stripping routine
input_string = the original string after trimming off extra spaces
comment_position = locations in the input file with comments
a = arbitrary string variable that the file will be read into
depth = string array containing the depth values from the text file

Next the variables for `read_positionelement` are listed.

original_string = a string variable for the comment stripping routine
input_string = the original_string after trimming off extra spaces
comment_position = locations in the input file with comments
a = arbitrary string variable that the input file will be read into
element = string array containing the element symbols from the text file

The variables for the program that has one image window, `final1image` are shown below. Please note that the variables are in the order they would be found
in the program code. Therefore the event handler variables are first followed by the widget definition variables.

```plaintext
resultdepth = the string array containing the depth values from the text file

resultelement = the string array containing the element symbols from the text file

ndepth = number of depths total in the resultdepth array

nelement = number of elements total in the resultelement array

x = temporary variable used in for statement to open images sequentially in the same window.

number_open = the number of depths calculated by ndepth - 1

element = event.value returned by the button selection

directory = the appropriate directory that the image is located in

file = name of the file in the directory that you want to open

slash = /

full_filename = the directory and filename combined

image = the result of reading the tiff of the full filename

y = the tiff image after it was resized

y2 = the tiff image after it was cropped

y3 = the tiff image after it was resized again

num = the generic image which is created by an array

depthfolder=field2.value - this is the value typed into the depth field

field_value = the value typed into the element symbol field

base = widget base
```
text = widget label to give user directions at the top of the program

text2 = widget label to give user more directions at the top of the program

text3-9 = widget labels without any values so they act as spacers

draw = draw widget

button = quit button

label = widget label to give directions for element symbol text field

field = widget text where user types in the element symbol

label2 = widget label to give directions for depth text field

field2 = widget text where user types in layer number

label3 = label to tell user to choose an element button

buttonnames = the names of the button are the array resultelement which is defined above

buttons = creates a compound button widget

Lastly, the variables for finalmanyimage are defined below. Again, they are in the order that they are found in the program, event handler variables followed by widget definition variables.

color_window = a generic image used to show that a file doesn't exist

color_window2 = a generic black image used to show that a file can’t be opened.

resultdepth = the string array containing the depth values from the text file

resultelement = the string array containing the element symbols from the text file

ndepth = number of depths total in the resultdepth array

nelement = number of elements total in the resultelement array
count = arbitrary variable that is initially set to -1 and increased by 1 when searching for an event.value until the case is matched after a depth button has been chosen.

elementcount = arbitrary variable that is initially set to -1 and increased by 1 when searching for an event.value until the case is matched after an element button is selected.

directory = the appropriate directory that the image is located in.

file = an array of files created by adding .tiff to the strings in the resultelement array.

slash = /.

folder = used to indicate which depth folder to look for the image in when a depth button has been selected.

window = used to get the window id for the window that the image will be opened in.

specificfile = determines which file will be opened in the depth folder by selecting one element from the array of elements saved as file.

full_filename = the directory and filename combined.

image = the result of reading the tiff of the full filename.

y = the tiff image after it was resized.

y2 = the tiff image after it was cropped.

y3 = the tiff image after it was resized again.

new_data = used to reinitialize the global data which contains the widget ids.

base = widget base.

label = widget label to tell the user to choose one button.

names = string array containing all of the element symbols and depths.

buttons = compound button widget where each button is labeled using.
the names array

button=quit button

draw = a draw widget

draw1-11 = the other 11 draw widgets

global_data = saves the widget ids that will be used by the event handler
Chapter 5

User’s Guide

The following is the procedure to use both programs.

1) Conduct a depth profile and collect elemental maps after each sputter cycle.

2) Create a folder directory in which each folder is labeled depth(n) where n is a number from 0 to the total number of sputters. For example, depth0 would contain pre-sputter maps and depth6 would contain maps collected after the sixth sputter.

3) Open the maps in Multipak and adjust the endpoints as necessary.

4) When map is finalized, choose the export as tiff file menu option. Do not worry about cropping or labeling the image - the IDL programs will do that.

5) Save the file as it’s elemental symbol in the appropriate folder. For example, a carbon map collected after the third sputter would be saved as C.tif in the depth3 folder.

6) Create a depth text file based on the template shown below or edit the template. Make sure the correct number of depths and correct depth values are entered in the text file. If the wrong number of depths is entered the ‘End of file encountered’ error will be returned when the program is run. Save it as depth.dat

# In the following text document you will need to enter
# the number of depths and the values of each depth in nm
# On the following line enter the total number of depths
14
# On the following lines enter the value of each depth in nm
# only enter one value per line
7) Create an element text file based on the template shown below or edit the
template. Make sure the correct number of elements and the correct element
symbols are entered in the text file. Again, it is important that the correct
number of elements is entered to prevent receiving an error message when the
program is run. Also, be case-sensitive. If you saved your files as lowercase
you must also enter lowercase letters in your text file. Save the text file as
element.dat.

# In this text editor you will enter the number of elements
# and the periodic table symbol for each element
# on the line below enter the total number of elements
10
# on the following lines enter the periodic table symbol for
# each element. It is suggested that you enter the elements in
# order that you observed them. For example if carbon maps were
# collected for the first few depths and nickel for the last few
# depths then carbon should be at the beginning of the list and
# nickel at the end of the list.
# Also be case-sensitive. If your files are lowercase then enter
# lowercase element symbols. Uppercase files require uppercase symbols.
# Only put one symbol per line.
c
o
f
na
k
p
co
ni
pt
8) Compile the program you plan to use - *final1image.pro* or *finalmanyimage.pro*.

9) Run the program.

10) For the *final1image* program select the element button you are interested in and each map for that image will be opened sequentially. It will appear as if you are watching the sputter happen. You can also enter text into the text fields to observe a particular image. For example, to see the carbon map acquired after the 8th sputter enter `c` in the field labeled 'Type in Element Symbol' and `8` in the field labeled 'Type in layer number'. Please note: be careful of lowercase and uppercase. If the images are saved in lowercase (example - c.tif) then your text file should have lowercase element symbols and your entries in the text field should be lowercase. If the images are uppercase then the text file and text field entries should be uppercase.

11) For the *finalmanyimage* program select the element button or depth button you are interested in. If you select an element button then the maps acquired for that element at 12 depths will be displayed. If you select a depth button then the maps of 12 different elements acquired at that depth will be shown.

12) When a generic image (not a black image) is displayed it indicates that a map was not acquired for this particular element at this depth. If a black image is observed then the map was collected but can not be opened for some reason. One possibility is that the map was compressed using LZW compression which is not licensed by IDL. Therefore an image compressed using LZW compression will not be displayed.

13) Enjoy!
Chapter 6

Testing Procedure

After the programs were written four data files were created to test the programs. The four data files varied in number of depths, number of elements, values of depths and names of elements. This extensive testing proved to the author that the program was working as anticipated. Files were opening as expected and the ability of the program to look for a second image if the first was not available for display was operating as anticipated. A shorter testing procedure will be proposed below for the user to verify the code is working properly.

A few key features of the program require verification before it can be concluded that the program is running as anticipated. Below is a list of 6 verification procedures to use on the finalmanyimage program. This will thoroughly prove the features of the program work. If you want a quick and easy test just run verification 1 and verification 6. Skip the other verification procedures.

Following the finalmanyimage verification tests is an example of an issue that the user may encounter with this program. Run this to see the limitation of the program’s wrap around feature.

A description of verification tests for the finalimage program is given at the end of this chapter. Example output images are not given because of the way in which the program runs. Instead the output will be described.

The first tests will be conducted using images saved in the 3ddepth3 folder. Use the replace option in the search menu to change the current folder to 3ddepth3. The element.dat file should have 10 elements and they should be c, o, f, na, k, p, ca, ni, pt, and cl. The depth.dat file should have the following 14 depths: 0.0, 0.78, 6.18, 11.58, 16.98, 22.38, 27.78, 33.18, 38.58, 56.38, 74.18, 91.98, 109.78, and 127.58. Now compile and run the program. Note - If you use finalmanyimage4.pro the folder will already be set at 3ddepth3.

Verification 1 - Determine if images can be displayed in all 12 windows.
Click on oxygen and observe a different image with the appropriate label in each window. Compare this to the figure labeled verification 1.

Verification 2 - Select a depth button and determine what happens when there aren’t enough images to fill all the windows.

Select 56.38 and check the element names and labels. Note: if your element.dat file has the elements in a different order than the one used to create the output file then they will be in different windows on your screen. Just check to make sure all of the same elements are present and they are all labeled 56.38. Notice that a map was not collected for fluorine at 56.38 so that window has a generic image. Also note that there are only 9 other elements so two windows have generic images because their aren’t enough elements to fill all 12.

The next tests will be conducted using images in the 3ddepth4 folder. If you use program finalmanyimage6.pro you will not have to replace the folder. The program finalmanyimage6.pro calls the depth6.dat text file as opposed to the depth.dat text file. You may have to make changes to the element.dat and depth6.dat files. The depth6.dat file should have 22 folders numbered 0-21. Element.dat should have 9 elements and can be any elemental symbols as long as they include oxygen and nickel. Compile and run the program.

Verification 3 - More than 12 depths

Select the oxygen button and compare the results to the output file shown. This test input file was created by having 22 folders but images only located in 5 of the folders (0,1,2,4,21). Again, you will notice a generic image in windows where a map was not collected. This verification also demonstrates the wrap around feature of the program. Since an oxygen map did not exist for depth 9 the program checked depth 21 and displayed that image.

Verification 4 - More than 12 depths

Select the button labeled 21 and observe oxygen and nickel. This proves that even though there are only 12 windows, depths higher than 12 can be displayed.

The final set of tests will be from the 3ddepth5 folder. You can use finalmanyimage7.pro to avoid changing the code to 3ddepth5. The program finalmanyimage7.pro calls text files labeled element6.dat and depth6.dat as opposed to element.dat and depth.dat. You will need to check the element6.dat and
Figure 6.1: Verification 1
Figure 6.3: Verification 3
Figure 6.4: Verification 4
depth6.dat text files to determine if they have the following 23 elements ni, p, o, y, xe, ur, sn, rn, kr, h, cu, ca, ba, b, au, as, ar, ag, li, he, ne, i, and cl, in the order given, and 22 folders labeled 0-21.

Verification 5 - More than 12 elements

Select depth 7 and compare to the output file. This proves the wrap around feature of the program. Although as, ar, and ag are not of the first twelve elements listed their elemental maps are displayed. The program first queried filenames for elements 4-6. When the result was negative it queried for elements 16-18, found elemental maps existed and displayed them.

Verification 6 - More than 12 elements and more than 12 depths
Select element ar and compare to the output file. This shows the wrap around feature of element and depth since ar is element number 17 on the list of elements and depth 20 is displayed.

Now, an example will be given to show the limitations of the wrap around feature of the code. To demonstrate use 3ddepth5 with the same element6.dat and depth6.dat files as used above.

Example-More than 12 elements and maps are collected at a particular depth.

Choose depth 0 and compare to the output file. At first glance everything looks fine but go back and look at the results from verification 6. You see that ar is present at depth 0 but not shown when you selected depth 0. Why? Because all of the first twelve elements had maps at
depth 0 so they were displayed and the program never looked for elements beyond them. It is suggested that the user check both depth and element buttons to avoid misinterpretation of results due to situations such as this one.

Two verification tests are suggested for the finalImage program.

Verification A - Opening multiple images or individual images.

The first test will utilize folder 3ddepth3. Compile and run the program. Now select the C button for carbon. Watch the files open and make sure they are opening sequentially and labeling appropriately. At the end you will have a generic image because a carbon map was not collected for the last depth. Now, type C in the element symbol text field and 6 in the layer number text field. You should see an image
labeled 'Element = c Depth = 27.78'. Compare this to the image shown below.

Verification B - More than 12 elements and depths.

Now use folder 3ddepth5. You can use program final1imagev.pro which will already be set at 3ddepth5. Use the same element6.dat and depth6.dat you used in verification 6 of finalmanyimage.pro. Select the button for ar and watch the windows open. You should see a map at depth 0, 7, and 20. A generic image will be displayed for the other depths. Now, type in o in the element symbol text field and 20 in the layer number text field and you should see an image labeled 'Element = o Depth = 20'.
Figure 6.8: Verification A
Chapter 7

Program Listing

The code to strip comments and read the depths that the user typed in a text file is shown first.

; The following program reads the text file that the user creates with the depths of ; the 3d depth profile.

; The program first opens and input file and an output file and then strips the ; comments from the input file and puts the information without the comments into ; the output file. The input file created by the user will have comments ; indicated by # that must be removed before the depths can be ; read by the IDL program.

; when the user runs the function he/she will have to indicate an input file and an ; output file
function read_positiondepth, input_file_name, output_file_name
; open the input file as read only
openr, input_file, input_file_name, /get_lun
; open the output file as writable
openw, output_file, output_file_name, /get_lun
; create a string variable
original_string=''
; while not at the end of the input file begin the following sequence
while (NOT EOF(input_file)) do begin
; read the input file into the string
readf, input_file, original_string
; cut out extra spaces
input_string=strtrim( original_string, 2)
; strip out the comments
comment_position = strpos(input_string, '#')
if( comment_position eq -1 and input_string ne '') then begin
The above portion of the program was used to strip the comments out and place the
uncommented program into an output file.
The remaining part of the program creates an array with the depths. This is
used to create the buttons in the IDL program to display the maps.

;The following function is used to strip out comments and read the elements
that the user put in a text file.

;The following program reads the text file that the user creates with the periodic table
; symbols for the elements.

;The program first opens an input file and an output file and then strips the
; comments from the input file and puts the information without the comments into
; the output file. The input file created by the user will have comments
; indicated by # that must be removed before the element symbols can be
; read by the IDL program.

;When the user runs this function they must indicate the input and output filenames
function read_authority, input_file_name, output_file_name
;The program opens the input file as read only and the output file as writable
openr, input_file, input_file_name, /get_lun
openw, output_file, output_file_name, /get_lun
original_string=''
;while not at the end of file begin the following sequence
while (NOT EOF(input_file)) do begin
printf, output_file, comment_position
end else if(comment_position gt 0) then begin
printf, output_file, strmid(input_string, 0, comment_position)
endif
endwhile
free_lun, input_file, output_file

return, depth
free_lun, output_file
end
;read the input file
readf, input_file, original_string
;strtrim trims off any excess spaces
input_string=strtrim(original_string, 2)
;remove the comments
comment_position = strpos(input_string, '#')
if( comment_position eq -1 and input_string ne '') then begin
;print the uncommented file to the output
printf, output_file, input_string
end else if( comment_position gt 0) then begin
printf, output_file, strmid(input_string, 0, comment_position)
endif
endwhile
free_lun, input_file, output_file

;The above portion of the program was used to strip the comments out and place the
;uncommented program into an output file.
;The remaining part of the program creates an array with the element symbols. This is
;used to create the buttons in the IDL program to display the maps.

;open the output file
openr, output_file, output_file_name, /get_lun
a=''
;read the output file into the string a
readf, output_file, a
;create an array called element of the string a
element=strarr(a)
readf, output_file, element
return, element
free_lun, output_file
end

Next is code for the program that displays all of the depths for a particular
element. Each image is opened right after the previous image giving the appearance that the user is watching the sputter happen. The user also has the ability to look at a specific image by typing appropriate information into two
text fields.

;The following program displays all of the depths for a certain element. It opens one
;right after the other making it seem as if the user is watching the sputter.
;If the user wants to look at one specific image they can type the element symbol
;and the layer number into text widgets and that image will be displayed.

pro final1image_event, event
;read_positiondepth and read_positionelement programs are used to read the
text files created by the user to determine the specific depths and elements,
respectively and save the depths and elements each into an array.
The n_elements command counts the total number of variables in each array
so the program knows how many elements and how many depths their are.
The user must save the text files as depth.dat and element.dat

resultdepth = read_positiondepth('depth.dat', 'depth2.dat')
ndepth = n_elements(resultdepth)
resultelement = read_positionelement('element.dat', 'element2.dat')
nelement = n_elements(resultelement)

;When a button is chosen the event.value is returned. The following
;lines of code are used to indicate what event will happen when each button
;is chosen
;For each element chosen the depths are opened from 0 to the total number
;of depths (ndepth - calculated above). The folders must be labeled as
;depthx where x is the layer number (0 - ndepth). For example maps collected for the
;sixth sputter must be saved in a folder labeled depth6. The folder labeled depth0
;is for maps collected prior to the first sputter

event.value is returned when a button is pressed - now call it element

x = 0

for x = 0, number_open, 1 do begin
close, /all
;identify the directory
directory = '/cis/grad/lab5756/3ddepth3/depth'
;the file is identified based on which button was pressed (element = event.value)
file = string(element) + '.tif'

;put together the full filename
full_filename = directory + strtrim(string(x), 2) + slash + file
;if the file exists proceed as described below
if(query_tiff(full_filename)) then begin
;read the file
image = read_tiff(full_filename)
;resize the image
y = congrid(image, 3, 512, 512)
;crop the image
y2 = y[*, 30:475, 75:400]
y3=congrid(y2, 3, 512, 512)

display the image
tvscl, y3, /order, true=1

The commands below are used to appropriately label each image.
xyouts, 300, 485, 'Depth =', /device, charsize=2, charthick=2.0, color=128
xyouts, 400, 485, resultdepth[x], /device, charsize=2, charthick=2.0, color=128
xyouts, 10, 485, 'Element =', /device, charsize=2, charthick=2.0, color=128
xyouts, 130, 485, element, /device, charsize=2, charthick=2.0, color=128

if the file doesn’t exist proceed as described below
endif else begin
display a generic image which indicates data wasn’t collected
num=indgen(3, 512, 512) * 10000
tvscl, num, /order, true=1

xyouts, 300, 485, 'Depth =', /device, charsize=2, charthick=2.0, color=128
xyouts, 400, 485, resultdepth[x], /device, charsize=2, charthick=2.0, color=128
xyouts, 10, 485, 'Element =', /device, charsize=2, charthick=2.0, color=128
xyouts, 130, 485, element, /device, charsize=2, charthick=2.0, color=128
endelse endelse endfor end

The following event handler is for the text fields. The user enters values
into the event handler and these are used to determine which image to
display and how to label it
pro field_event, event
close, /all
resultdepth=read_positiondepth('depth.dat', 'depth2.dat')
ndepth=n_elements(resultdepth)
resultelement=read_positionelement('element.dat', 'element2.dat')
nelement=n_elements(resultelement)

Following commands are used to get the uvalues of the text widgets
Widget_Control, event.top, get_uvalue=widgets
field=widgets.field
field2=widgets.field2

Widget_Control, field, get_value=field_value
Widget_Control, field2, get_value=field2_value
directory='/cis/grad/lab5756/3ddepth2/depth'

The result from the element text entered in is incorporated in the file name
file=field_value+'.tif'
slash='/'

The result from the depth layer text is added to the end of the directory
so the appropriate depth folder is opened.
depthfolder=field2_value

33
full_filename=directory+depthfolder+slash+file
if(query_tiff(full_filename)) then begin
image=read_tiff(full_filename)
y=congrid(image, 3, 512, 512)
y2=y[*, 30:475, 75:400]
y3=congrid(y2, 3, 512, 512)
tvscl, y3, /order,true=1
;The above commands first check to see if the file exists. If it does the image is
;sized for the window, cropped and
;then resized. Finally the image is displayed.
;The commands below are used to appropriately label each image.
xyouts, 300, 485, 'Depth =', /device, charsize=2, charthick=2.0, color=128
xyouts, 400, 485, resultdepth[depthfolder], /device, charsize=2, charthick=2.0, color=128
xyouts, 10, 485, 'Element =', /device, charsize=2, charthick=2.0, color=128
xyouts, 130, 485, field_value, /device, charsize=2, charthick=2.0, color=128
;if the file doesn't exist then a generic image is displayed
endif else begin
num=indgen(3,512,512)*10000
tvscl, num, /order, true=1
xyouts, 300, 485, 'Depth =', /device, charsize=2, charthick=2.0, color=128
xyouts, 400, 485, resultdepth[depthfolder], /device, charsize=2, charthick=2.0, color=128
xyouts, 10, 485, 'Element =', /device, charsize=2, charthick=2.0, color=128
xyouts, 130, 485, field_value, /device, charsize=2, charthick=2.0, color=128
endelse
end

;Event handler for the clear window button which plots an array with values of zero that
;is the size of the window
pro clear_button_event, event
close, /all
num=fltarr(3,512,512)
tvscl, num, /order, true=1
end

;Event handler for the quit button
pro quit_button_event, event
Widget_Control, event.top, /destroy
close, /all
end

pro final1image
;Create base widget
base=Widget_Base(column=2, Title='Depth Profile')
;Create a text box with information about the program
text=Widget_label(base, value='Click on an element button to sequentially open all of the depths for that element or type in an element symbol and layer number to view')
Finally the code for the program that displays twelve images at once is shown below. The images can be displayed based on depth (all of the elements for a particular depth) or based on element (all of the depths for a particular element).

;The following program is used to open all of the elements at one particular depth or all of the depths for one particular element. The user must create two text files; one with the element symbols and the other with the depths and save them as element.dat
pro finalmanyimage4_event, event
;Get the global data
widget_control, event.top, get_uvalue=global_data
;Define two windows that will be used as blank windows. The first will be used
;when a file isn’t present (the data was collected) whereas the second will
;be used if the file can’t be opened (error).
   color_window=indgen(3,256,256)*10000
   color_window2=fitarr(3,256,256)
;Read the depth text file created by the user containing the depths
resultdepth=read_positiondepth('depth.dat', 'depth2.dat')
;Calculate the total number of depths
ndepth=n_elements(resultdepth)
;Read the element text file created by the user containing the element symbols
resultelement=read_positionelement('element.dat', 'element2.dat')
;Calculate the number of elements
nelement=n_elements(resultelement)
;set the arbitrary variables count and elementcount at -1
count=-1
elementcount=-1
;Jump is the label for the goto command later in the program.
;When the goto command is interpreted the program will go come to the line labeled jump
;At this point it will increment the elementcount and count by 1
jump: elementcount=elementcount+1
   count=count+1
;The following if commands allow the program to work when the number of
;elements is not the same as the number of depths. For example, if the number
;of elements is greater than the number of depths when elementcount has reached
;the total number of elements elementcount will be set back to 0.
;Had the if commands not been included the program would have failed at
;the time when the number of depths exceeded the number of elements or vice versa.
if (elementcount ge nelement) then begin
   elementcount=0
endif
if (count ge ndepth) then begin
   count=0
endif
;The case statements for selecting a button start here
Case event.value of
   ;The first case is for the different depths (count variable). This will cause
   ;all of the elements to open for a specific depth.
resultdepth[count] :begin
   close, /all
36
The directory and file are defined below. `file[0]` opens the first element in the `resultelement` array. After it is displayed the specific file will be `file[1]` which will open the second element. So on and so forth until all twelve windows have an image. Since there are only twelve windows the maximum number of elements that can be displayed at a time is 12. As you read the code you will notice an `if` statement structure that checks for an image at a higher depth if one at a lower depth is not present. For example if a nickel map is not collected at depth 2 then it will check and see if a nickel map was collected at depth 14. If so, that image will be displayed.

```plaintext
file=string(resultelement)+'.tif'
directory='/cis/grad/lab5756/3ddepth3'/
folder='depth'+strtrim(string(count),2)

; Identify the window to be opened
window=global_data.window_id
wset, window

; first if statement to determine if enough depths exist to need this window
if(nelement gt 0) then begin
  specificfile=file[0]
  full_filename=directory+folder+slash+specificfile
  ; second if statement to see if tiff file exists
  if(query_tiff(full_filename)) then begin
    ; if the file exists then read it
    image=read_tiff(full_filename)
    ; The following three commands resize the image, crop it and resize it again
    y=congrid(image,3,256,256)
    y2=y[*, 10:240, 40:215]
    y3=congrid(y2, 3, 256, 256)
    ; Display the image
    tvscl, y3, /order,true=1
    ; Label the element name on the image
    xyouts, 25, 10, resultelement[0], /device, charsize=1.5, charthick=1.0, color=64
    ; Label the depth on the image
    xyouts, 150, 10, resultdepth[count], /device, charsize=1.5, charthick=1.0, color=64
  endif else begin
    ; third if statement to see if more elements than windows so can loop around
    if(nelement gt 12) then begin
      specificfile=file[12]
      full_filename=directory+folder+slash+specificfile
      ; fourth if statement to see if a tiff file exists for the particular depth
      ; for the 13th element. If it does the commands described above are used
      ; to display and label the image
    end
end
```

37
if(query_tiff(full_filename)) then begin
    image=read_tiff(full_filename)
y=congrid(image,3,256,256)
y2=y[*, 10:240, 40:215]
y3=congrid(y2, 3, 256, 256)
tvscl, y3, /order,true=1
    xyouts, 25, 10, resultelement[12], /device, charsize=1.5, charthick=1.0, color=64
    xyouts, 150, 10, resultdepth[count], /device, charsize=1.5, charthick=1.0, color=64
    ;if the tiff file can’t be read a generic image will be displayed
endif else begin
    tvscl, color_window2, /order, true=1
endelse
;if the 13th element doesn’t exist a generic image will be displayed
endif else begin
    tvscl, color_window, /order, true=1
endelse
;if there isn’t one element then the generic image is displayed
endif else begin
    tvscl, color_window, /order, true=1
endelse

;Below the process repeats to open the second image (element 2) at the depth chosen
;This continues for all 12 images. I will not comment these since they are
;the same as the above image with just a change of file number

window1=global_data.window_id1
wset, window1
;first if statement to determine if enough depths exist to need this window
if(nelement gt 1) then begin
    specificfile=file[1]
    full_filename=directory+folder+slash+specificfile
    ;second if statement to see if tiff file exists
    if(query_tiff(full_filename)) then begin
        image=read_tiff(full_filename)
y=congrid(image,3,256,256)
y2=y[*, 10:240, 40:215]
y3=congrid(y2, 3, 256, 256)
tvscl, y3, /order,true=1
    xyouts, 25, 10, resultelement[1], /device, charsize=1.5, charthick=1.0, color=64
    xyouts, 150, 10, resultdepth[count], /device, charsize=1.5, charthick=1.0, color=64
    ;if the tiff file doesn’t exist then follow next steps
endif else begin
    ;third if statement to see if more elements than windows so can loop around
    if(nelement gt 13) then begin

specificfile=file[13]  
full_filename=directory+folder+slash+specificfile  
if(query_tiff(full_filename)) then begin  
  image=read_tiff(full_filename)  
y=congrid(image,3,256,256)  
y2=y[*, 10:240, 40:215]  
y3=congrid(y2, 3, 256, 256)  
  tvscl, y3, /order,true=1  
  xyouts, 25, 10, resultelement[13], /device, charsize=1.5, charthick=1.0, color=64  
  xyouts, 150, 10, resultdepth[count], /device, charsize=1.5, charthick=1.0, color=64  
endif else begin  
tvscl, color_window2, /order, true=1  
endelse  
dendif else begin  
tvscl, color_window, /order, true=1  
endelse  
edendif else begin  
tvscl, color_window, /order, true=1  
endelse  
edendif else begin  
tvscl, color_window2, /order, true=1  
endelse  
edendif else begin  
tvscl, color_window, /order, true=1  
endelse  
edendif else begin  
tvscl, color_window2, /order, true=1  
endelse  
edendif
d
d 

window2=global_data.window_id2  
wset, window2  
; first if statement to determine if enough depths exist to need this window  
if(nelement gt 2) then begin  
specificfile=file[2]  
full_filename=directory+folder+slash+specificfile  
; second if statement to see if tiff file exists  
if(query_tiff(full_filename)) then begin  
  image=read_tiff(full_filename)  
y=congrid(image,3,256,256)  
y2=y[*, 10:240, 40:215]  
y3=congrid(y2, 3, 256, 256)  
  tvscl, y3, /order,true=1  
  xyouts, 25, 10, resultelement[2], /device, charsize=1.5, charthick=1.0, color=64  
  xyouts, 150, 10, resultdepth[count], /device, charsize=1.5, charthick=1.0, color=64  
; if the tiff file doesn't exist then follow next steps  
endif else begin  
; third if statement to see if more elements than windows so can loop around  
if(nelement gt 14) then begin  
specificfile=file[14]  
full_filename=directory+folder+slash+specificfile  
if(query_tiff(full_filename)) then begin  
  image=read_tiff(full_filename)  
y=congrid(image,3,256,256)  
y2=y[*, 10:240, 40:215]  

y3=congrid(y2, 3, 256, 256)
tvscl, y3, /order, true=1
xyouts, 25, 10, resultelement[14], /device, charsize=1.5, charthick=1.0, color=64
xyouts, 150, 10, resultdepth[count], /device, charsize=1.5, charthick=1.0, color=64
endif else begin
tvscl, color_window2, /order, true=1
delse
endif else begin
tvscl, color_window, /order, true=1
delse
dendif else begin
tvscl, color_window, /order, true=1
delse
window3=global_data.window_id3
wset, window3
; first if statement to determine if enough depths exist to need this window
if(nelement ge 3) then begin
specificfile=file[3]
full_filename=directory+folder+slash+specificfile
; second if statement to see if tiff file exists
if(query_tiff(full_filename)) then begin
image=read_tiff(full_filename)
y=congrid(image, 3, 256, 256)
y2=y[*, 10:240, 40:215]
y3=congrid(y2, 3, 256, 256)
tvscl, y3, /order, true=1
xyouts, 25, 10, resultelement[3], /device, charsize=1.5, charthick=1.0, color=64
xyouts, 150, 10, resultdepth[count], /device, charsize=1.5, charthick=1.0, color=64
endif else begin
; third if statement to see if more elements than windows so can loop around
if(nelement ge 15) then begin
specificfile=file[15]
full_filename=directory+folder+slash+specificfile
if(query_tiff(full_filename)) then begin
image=read_tiff(full_filename)
y=congrid(image, 3, 256, 256)
y2=y[*, 10:240, 40:215]
y3=congrid(y2, 3, 256, 256)
tvscl, y3, /order, true=1
xyouts, 25, 10, resultelement[15], /device, charsize=1.5, charthick=1.0, color=64
xyouts, 150, 10, resultdepth[count], /device, charsize=1.5, charthick=1.0, color=64
endif else begin
tvscl, color_window2, /order, true=1
endelse
endif else begin
tvscl, color_window, /order, true=1
endelse
endif else begin
tvscl, color_window, /order, true=1
endelse

window4=global_data.window_id4
wset, window4
; first if statement to determine if enough depths exist to need this window
if(nelement gt 4) then begin
specificfile=file[4]
full_filename=directory+folder+slash+specificfile
; second if statement to see if tiff file exists
if(query_tiff(full_filename)) then begin
image=read_tiff(full_filename)
y=congrid(image,3,256,256)
y2=y[*, 10:240, 40:215]
y3=congrid(y2, 3, 256, 256)
tvscl, y3, /order, true=1
xyouts, 25, 10, resultelement[4], /device, charsize=1.5, charthick=1.0, color=64
xyouts, 150, 10, resultdepth[count], /device, charsize=1.5, charthick=1.0, color=64
; if the tiff file doesn't exist then follow next steps
endif else begin
; third if statement to see if more elements than windows so can loop around
if(nelement gt 16) then begin
specificfile=file[16]
full_filename=directory+folder+slash+specificfile
if(query_tiff(full_filename)) then begin
image=read_tiff(full_filename)
y=congrid(image,3,256,256)
y2=y[*, 10:240, 40:215]
y3=congrid(y2, 3, 256, 256)
tvscl, y3, /order, true=1
xyouts, 25, 10, resultelement[16], /device, charsize=1.5, charthick=1.0, color=64
xyouts, 150, 10, resultdepth[count], /device, charsize=1.5, charthick=1.0, color=64
endif else begin
tvscl, color_window2, /order, true=1
endelse
endif else begin
tvscl, color_window, /order, true=1
endelse
endelse
endif else begin
tvscl, color_window, /order, true=1
endelse

window5=global_data.window_id5
wset, window5
; first if statement to determine if enough depths exist to need this window
if(nelement gt 5) then begin
  specificfile=file[5]
  full_filename=directory+folder+slash+specificfile
; second if statement to see if tiff file exists
  if(query_tiff(full_filename)) then begin
    image=read_tiff(full_filename)
    y=congrid(image,3,256,256)
    y2=y[*, 10:240, 40:215]
    y3=congrid(y2, 3, 256, 256)
    tvscl, y3, /order, true=1
    xyouts, 25, 10, resultelement[5], /device, charsize=1.5, charthick=1.0, color=64
    xyouts, 150, 10, resultdepth[count], /device, charsize=1.5, charthick=1.0, color=64
  endelse else begin
    tvscl, color_window2, /order, true=1
  endelse
endif else begin
; third if statement to see if more elements than windows so can loop around
  if(nelement gt 17) then begin
    specificfile=file[17]
    full_filename=directory+folder+slash+specificfile
    if(query_tiff(full_filename)) then begin
      image=read_tiff(full_filename)
      y=congrid(image,3,256,256)
      y2=y[*, 10:240, 40:215]
      y3=congrid(y2, 3, 256, 256)
      tvscl, y3, /order, true=1
      xyouts, 25, 10, resultelement[17], /device, charsize=1.5, charthick=1.0, color=64
      xyouts, 150, 10, resultdepth[count], /device, charsize=1.5, charthick=1.0, color=64
    endif else begin
      tvscl, color_window2, /order, true=1
    endelse
  endelse else begin
    tvscl, color_window, /order, true=1
  endelse
endelse
endelse

tvscl, color_window, /order, true=1
endelse

window6=global_data.window_id6
wset, window6
;first if statement to determine if enough depths exist to need this window
if(nelement gt 6) then begin
  specificfile=file[6]
  full_filename=directory+folder+slash+specificfile
;second if statement to see if tiff file exists
  if(query_tiff(full_filename)) then begin
    image=read_tiff(full_filename)
    y=congrid(image,3,256,256)
    y2=y[*, 10:240, 40:215]
    y3=congrid(y2, 3, 256, 256)
    tvscl, y3, /order,true=1
    xyouts, 25, 10, resultelement[6], /device, charsize=1.5, charthick=1.0, color=64
    xyouts, 150, 10, resultdepth[count], /device, charsize=1.5, charthick=1.0, color=64
  endif else begin
;third if statement to see if more elements than windows so can loop around
    if(nelement gt 18) then begin
      specificfile=file[18]
      full_filename=directory+folder+slash+specificfile
      if(query_tiff(full_filename)) then begin
        image=read_tiff(full_filename)
        y=congrid(image,3,256,256)
        y2=y[*, 10:240, 40:215]
        y3=congrid(y2, 3, 256, 256)
        tvscl, y3, /order,true=1
        xyouts, 25, 10, resultelement[18], /device, charsize=1.5, charthick=1.0, color=64
        xyouts, 150, 10, resultdepth[count], /device, charsize=1.5, charthick=1.0, color=64
      endif else begin
        tvscl, color_window2, /order, true=1
      endelse
    endelse
    endelse
    endelse
    tvscl, color_window, /order, true=1
  endelse
endelse
endelse
endelse
endelse
endelse
; second if statement to see if tiff file exists
if(query_tiff(full_filename)) then begin
  image=read_tiff(full_filename)
y=congrid(image,3,256,256)
y2=y[*, 10:240, 40:215]
y3=congrid(y2, 3, 256, 256)
tvscl, y3, /order, true=1
xyouts, 25, 10, resultelement[7], /device, charsize=1.5, charthick=1.0, color=64
xyouts, 150, 10, resultdepth[count], /device, charsize=1.5, charthick=1.0, color=64
; if the tiff file doesn't exist then follow next steps
endif else begin
; third if statement to see if more elements than windows so can loop around
if(nelement gt 19) then begin
  specificfile=file[19]
  full_filename=directory+folder+slash+specificfile
  if(query_tiff(full_filename)) then begin
    image=read_tiff(full_filename)
y=congrid(image,3,256,256)
y2=y[*, 10:240, 40:215]
y3=congrid(y2, 3, 256, 256)
tvscl, y3, /order, true=1
    xyouts, 25, 10, resultelement[19], /device, charsize=1.5, charthick=1.0, color=64
    xyouts, 150, 10, resultdepth[count], /device, charsize=1.5, charthick=1.0, color=64
  endif else begin
    tvscl, color_window2, /order, true=1
  endelse
endif else begin
  tvscl, color_window, /order, true=1
endelse
endif else begin
  tvscl, color_window, /order, true=1
endelse
endif else begin
  tvscl, color_window, /order, true=1
endelse
endif else begin
  tvscl, color_window, /order, true=1
endelse
endif else begin
  tvscl, color_window, /order, true=1
endelse
endif else begin
  tvscl, color_window, /order, true=1
endelse
endif else begin
  tvscl, color_window, /order, true=1
endelse
endif else begin
  tvscl, color_window, /order, true=1
endelse
endif else begin
  tvscl, color_window, /order, true=1
endelse
endif else begin
  tvscl, color_window, /order, true=1
endelse
tvscl, y3, /order,true=1
xyouts, 25, 10, resultelement[8], /device, charsize=1.5, charthick=1.0, color=64
xyouts, 150, 10, resultdepth[count], /device, charsize=1.5, charthick=1.0, color=64
; if the tiff file doesn't exist then follow next steps
endif else begin
; third if statement to see if more elements than windows so can loop around
if(nelement gt 20) then begin
    specificfile=file[20]
    full_filename=directory+folder+slash+specificfile
    if(query_tiff(full_filename)) then begin
        image=read_tiff(full_filename)
        y=congrid(image,3,256,256)
        y2=y[, 10:240, 40:215]
        y3=congrid(y2, 3, 256, 256)
        tvscl, y3, /order,true=1
        xyouts, 25, 10, resultelement[20], /device, charsize=1.5, charthick=1.0, color=64
        xyouts, 150, 10, resultdepth[count], /device, charsize=1.5, charthick=1.0, color=64
    endif else begin
        tvscl, color_window2, /order, true=1
        endelse
    endif else begin
        tvscl, color_window, /order, true=1
        endelse
    endif else begin
        tvscl, color_window, /order, true=1
        endelse
    endif else begin
        tvscl, color_window, /order, true=1
        endelse
endif else begin
window9=global_data.window_id9
wset, window9
; first if statement to determine if enough depths exist to need this window
if(nelement gt 9) then begin
    specificfile=file[9]
    full_filename=directory+folder+slash+specificfile
    ; second if statement to see if tiff file exists
    if(query_tiff(full_filename)) then begin
        image=read_tiff(full_filename)
        y=congrid(image,3,256,256)
        y2=y[, 10:240, 40:215]
        y3=congrid(y2, 3, 256, 256)
        tvscl, y3, /order,true=1
        xyouts, 25, 10, resultelement[9], /device, charsize=1.5, charthick=1.0, color=64
        xyouts, 150, 10, resultdepth[count], /device, charsize=1.5, charthick=1.0, color=64
        ; if the tiff file doesn't exist then follow next steps
        endif else begin
            ; third if statement to see if more elements than windows so can loop around

if(nelement gt 21) then begin
  specificfile=file[21]
  full_filename=directory+folder+slash+specificfile
if(query_tiff(full_filename)) then begin
  image=read_tiff(full_filename)
  y=congrid(image,3,256,256)
  y2=y[*, 10:240, 40:215]
  y3=congrid(y2, 3, 256, 256)
  tvscl, y3, /order, true=1
  xyouts, 25, 10, resultelement[21], /device, charsize=1.5, charthick=1.0, color=64
  xyouts, 150, 10, resultdepth[count], /device, charsize=1.5, charthick=1.0, color=64
endelse
endif else begin
  tvscl, color_window2, /order, true=1
endelse
endif else begin
  tvscl, color_window, /order, true=1
endelse
endif else begin
  tvscl, color_window, /order, true=1
endelse
endif else begin
  tvscl, color_window, /order, true=1
endelse
endif else begin
  tvscl, color_window, /order, true=1
endelse
endelse

window10=global_data.window_id10
wset, window10
;first if statement to determine if enough depths exist to need this window
if(nelement gt 10) then begin
  specificfile=file[10]
  full_filename=directory+folder+slash+specificfile
;second if statement to see if tiff file exists
if(query_tiff(full_filename)) then begin
  image=read_tiff(full_filename)
  y=congrid(image,3,256,256)
  y2=y[*, 10:240, 40:215]
  y3=congrid(y2, 3, 256, 256)
  tvscl, y3, /order, true=1
  xyouts, 25, 10, resultelement[10], /device, charsize=1.5, charthick=1.0, color=64
  xyouts, 150, 10, resultdepth[count], /device, charsize=1.5, charthick=1.0, color=64
;if the tiff file doesn’t exist then follow next steps
endelse
endif else begin
;third if statement to see if more elements than windows so can loop around
if(nelement gt 22) then begin
  specificfile=file[22]
  full_filename=directory+folder+slash+specificfile
if(query_tiff(full_filename)) then begin
  image=read_tiff(full_filename)
  y=congrid(image,3,256,256)

46
y2 = y[* , 10:240, 40:215]
y3 = congrid(y2, 3, 256, 256)
tvscl, y3, /order, true=1
xyouts, 25, 10, resultelement[22], /device, charsize=1.5, charthick=1.0, color=64
xyouts, 150, 10, resultdepth[count], /device, charsize=1.5, charthick=1.0, color=64
endif else begin
tvscl, color_window2, /order, true=1
endelse
endif else begin
tvscl, color_window, /order, true=1
endelse
endif else begin
tvscl, color_window, /order, true=1
endelse
endif else begin

window11 = global_data.window_id11
wset, window11

; first if statement to determine if enough depths exist to need this window
if(nelement gt 11) then begin
    specificfile = file[11]
    full_filename = directory + folder + slash + specificfile
    ; second if statement to see if tiff file exists
    if(query_tiff(full_filename)) then begin
        image = read_tiff(full_filename)
y = congrid(image, 3, 256, 256)
y2 = y[* , 10:240, 40:215]
y3 = congrid(y2, 3, 256, 256)
tvscl, y3, /order, true=1
xyouts, 25, 10, resultelement[11], /device, charsize=1.5, charthick=1.0, color=64
xyouts, 150, 10, resultdepth[count], /device, charsize=1.5, charthick=1.0, color=64
    endelse
    ; if the tiff file doesn’t exist then follow next steps
    endif else begin
        ; third if statement to see if more elements than windows so can loop around
        if(nelement gt 23) then begin
            specificfile = file[23]
            full_filename = directory + folder + slash + specificfile
            if(query_tiff(full_filename)) then begin
                image = read_tiff(full_filename)
y = congrid(image, 3, 256, 256)
y2 = y[* , 10:240, 40:215]
y3 = congrid(y2, 3, 256, 256)
tvscl, y3, /order, true=1
xyouts, 25, 10, resultelement[23], /device, charsize=1.5, charthick=1.0, color=64
xyouts, 150, 10, resultdepth[count], /device, charsize=1.5, charthick=1.0, color=64
        endelse
        endwhile
tvscl, color_window2, /order, true=1
endif else begin
tvscl, color_window, /order, true=1
endelse
endif else begin
tvscl, color_window, /order, true=1
endelse
endelse
end else begin
tvscl, color_window, /order, true=1
endelse
endelse
end

;That ends the first case. Now begins the second case which is opening all of the
;depths of one element
resultelement[elementcount] :begin
    color_window2=indgen(3,256,256)*10000
    color_window=fltarr(3,256,256)
close, /all
;identify the window
window=global_data.window_id
;The directory is specified below for depth0.
directory='/cis/grad/lab5756/3ddepth3/depth0/
file=string(resultelement[elementcount])+'.tif'
full_filename=directory+file
;Determine if the tiff file exists for this element at depth 0
if(query_tiff(full_filename)) then begin
    ;if the file does exist, read the image
    image=read_tiff(full_filename)
    ;resize the image
    y=congrid(image,3,256,256)
    ;crop the image
    y2=y[*, 10:240, 40:215]
    ;select the window
    wset, window
    ;resize after cropping
    y3=congrid(y2, 3, 256, 256)
    ;display the image
tvscl, y3, /order,true=1
    ;label the element name
    xyouts, 25, 10, resultelement[elementcount], /device, charsize=1.5, charthick=1.0, color=64
    ;label the depth
    xyouts, 150, 10, resultdepth[0], /device, charsize=1.5, charthick=1.0, color=64
    ;Define what to do if the tiff file does not exist - try the 13th depth
endif else begin
directory='/cis/grad/lab5756/3ddepth3/depth12/
file=string(resultelement[elementcount])+'.tif'
end
full_filename=directory+file
;check to see if the 13th depth exists
if(query_tiff(full_filename)) then begin
; if so, read the image
image=read_tiff(full_filename)
; resize the image
y=congrid(image,3,256,256)
; crop the image
y2=y[*, 10:240, 40:215]
; select the window
wset, window
; resize after cropping
y3=congrid(y2, 3, 256, 256)
; display the image
tvscl, y3, /order, true=1
; label the element name
xyouts, 25, 10, resultelement[elementcount], /device, charsize=1.5, charthick=1.0, color=64
; label the depth
xyouts, 150, 10, resultdepth[12], /device, charsize=1.5, charthick=1.0, color=64
; Define what to do if the second tiff file doesn't exist - print a generic image
endif else begin
wset, window & tvscl, color_window2, /order, true=1
endelse
endelse

; The above process will repeat for the second depth which is directory depth1
; I will not comment the lines below that are the same as above with just
; a directory change. There is one change though. I made the assumption that their
; was at least one element. To avoid an error causing the program to crash
; I will add an if statement to the following sections that checks to make sure the
; number of elements that exists is greater than the current window number.
window1=global_data.window_id1
; first if statement to determine if enough depths exist to need this window
if(ndepth gt 1) then begin
directory='/cis/grad/lab5756/3ddepth3/depth1/
file=string(resultelement[elementcount])+'.tif'
full_filename=directory+file
; second if statement to see if tiff file exists
if(query_tiff(full_filename)) then begin
image=read_tiff(full_filename)
y=congrid(image,3,256,256)
y2=y[*, 10:240, 40:215]
wset, window1
y3=congrid(y2, 3, 256, 256)
tvscl, y3, /order, true=1
xyouts, 25, 10, resultelement[elementcount], /device, charsize=1.5, charthick=1.0, color=64
xyouts, 150, 10, resultdepth[1], /device, charsize=1.5, charthick=1.0, color=64
;if the tiff file doesn’t exist then follow next steps
endif else begin
;third if statement to see if more depths than windows so can loop around
if(ndepth gt 13) then begin
    directory='/cis/grad/lab5756/3ddepth3/depth13/'
    file=string(resultelement[elementcount])+'.tif'
    full_filename=directory+file
    ;fourth if statement to see if a tiff file exists for the particular element
    ;for the 14th depth
    if(query_tiff(full_filename)) then begin
        image=read_tiff(full_filename)
        y=congrid(image,3,256,256)
        y2=y[*, 10:240, 40:215]
        wset, window1
        y3=congrid(y2, 3, 256, 256)
        tvscl, y3, /order,true=1
        xyouts, 25, 10, resultelement[elementcount], /device, charsize=1.5, charthick=1.0, color=64
        ;if there isn’t an tiff file for that element then print a generic image
        endif else begin
            wset, window1 & tvscl, color_window2, /order, true=1
        endelse
        ;if there aren’t enough images that you need to loop around then
        ;print a generic image
        endif else begin
            wset, window1 & tvscl, color_window2, /order, true=1
        endelse
        endelse
    endwhile
endif else begin
    wset, window1 & tvscl, color_window, /order, true=1
endelse
if(ndepth gt 2) then begin
    window2=global_data.window_id2
    wset, window2
directory='/cis/grad/lab5756/3ddepth3/depth2/'
    file=string(resultelement[elementcount])+'.tif'
    full_filename=directory+file
    ;first if statement to see if tiff file exists
    if(query_tiff(full_filename)) then begin
        image=read_tiff(full_filename)
        y=congrid(image,3,256,256)
        y2=y[*, 10:240, 40:215]
        y3=congrid(y2, 3, 256, 256)
        tvscl, y3, /order,true=1
    endif else begin

xyouts, 25, 10, resultelement[elementcount], /device, charsize=1.5, charthick=1.0, color=64
xyouts, 150, 10, resultdepth[2], /device, charsize=1.5, charthick=1.0, color=64
endif else begin
if(ndepth gt 14) then begin
directory='\cis/grad/lab5756/3ddepth3/depth14/'
file=string(resultelement[elementcount])+'.tif'
full_filename=directory+file
if(query_tiff(full_filename)) then begin
image=read_tiff(full_filename)
y=congrid(image,3,256,256)
y2=y[*, 10:240, 40:215]
y3=congrid(y2, 3, 256, 256)
tvscl, y3, /order, true=1
xyouts, 25, 10, resultelement[elementcount], /device, charsize=1.5, charthick=1.0, color=64
xyouts, 150, 10, resultdepth[14], /device, charsize=1.5, charthick=1.0, color=64
endif else begin
tvscl, color_window2, /order, true=1
endelse
endif else begin
xyouts, 25, 10, resultelement[elementcount], /device, charsize=1.5, charthick=1.0, color=64
xyouts, 150, 10, resultdepth[14], /device, charsize=1.5, charthick=1.0, color=64
endif else begin
tvscl, color_window2, /order, true=1
endelse
endif else begin
if(ndepth gt 3) then begin
window3=global_data.window_id3
wset, window3
directory='\cis/grad/lab5756/3ddepth3/depth3/'
file=string(resultelement[elementcount])+'.tif'
full_filename=directory+file
if(query_tiff(full_filename)) then begin
image=read_tiff(full_filename)
y=congrid(image,3,256,256)
y2=y[*, 10:240, 40:215]
y3=congrid(y2, 3, 256, 256)
tvscl, y3, /order, true=1
xyouts, 25, 10, resultelement[elementcount], /device, charsize=1.5, charthick=1.0, color=64
xyouts, 150, 10, resultdepth[3], /device, charsize=1.5, charthick=1.0, color=64
endif else begin
if(ndepth gt 15) then begin
directory='\cis/grad/lab5756/3ddepth3/depth15/'
file=string(resultelement[elementcount])+'.tif'
full_filename=directory+file
if(query_tiff(full_filename)) then begin
  image = read_tiff(full_filename)
y = congrid(image, 3, 256, 256)
y2 = y[*, 10:240, 40:215]
y3 = congrid(y2, 3, 256, 256)
tvscl, y3, /order, true=1
xyouts, 25, 10, resultelement[elementcount], /device, charsize=1.5, charthick=1.0, color=64
xyouts, 150, 10, resultdepth[15], /device, charsize=1.5, charthick=1.0, color=64
end else begin
tvscl, color_window2, /order, true=1
endelse
endif else begin
  vscl, color_window2, /order, true=1
endelse
endif else begin
tvscl, color_window, /order, true=1
endelse
endif else begin
  if(ndepth gt 4) then begin
    window4 = global_data.window_id4
    wset, window4
directory = '/cis/grad/lab5756/3ddepth3/depth4'/
    file = string(resultelement[elementcount]) + '.tif'
    full_filename = directory + file
    ; first if statement to see if tiff file exists
    if(query_tiff(full_filename)) then begin
      image = read_tiff(full_filename)
y = congrid(image, 3, 256, 256)
y2 = y[*, 10:240, 40:215]
y3 = congrid(y2, 3, 256, 256)
tvscl, y3, /order, true=1
xyouts, 25, 10, resultelement[elementcount], /device, charsize=1.5, charthick=1.0, color=64
xyouts, 150, 10, resultdepth[4], /device, charsize=1.5, charthick=1.0, color=64
end else begin
  if(ndepth gt 16) then begin
    directory = '/cis/grad/lab5756/3ddepth3/depth16'/
    file = string(resultelement[elementcount]) + '.tif'
    full_filename = directory + file
    if(query_tiff(full_filename)) then begin
      image = read_tiff(full_filename)
y = congrid(image, 3, 256, 256)
y2 = y[*, 10:240, 40:215]
y3 = congrid(y2, 3, 256, 256)
tvscl, y3, /order, true=1
xyouts, 25, 10, resultelement[elementcount], /device, charsize=1.5, charthick=1.0, color=64
xyouts, 150, 10, resultdepth[16], /device, charsize=1.5, charthick=1.0, color=64
end else begin
  endif
end
endif else begin
tvscl, color_window2, /order, true=1
endelse
endif else begin
tvscl, color_window2, /order, true=1
endelse
endif else begin
tvscl, color_window, /order, true=1
endelse
endif else begin
if(ndepth gt 5) then begin
window5=global_data.window_id5
wset, window5
directory='/cis/grad/lab5756/3ddepth3/depth5/'
file=string(resultelement[elementcount])+'.tif'
full_filename=directory+file
;first if statement to see if tiff file exists
if(query_tiff(full_filename)) then begin
image=read_tiff(full_filename)
y=congrid(image,3,256,256)
y2=y[*, 10:240, 40:215]
y3=congrid(y2, 3, 256, 256)
tvscl, y3, /order,true=1
xyouts, 25, 10, resultelement[elementcount], /device, charsize=1.5, charthick=1.0, color=64
xyouts, 150, 10, resultdepth[5], /device, charsize=1.5, charthick=1.0, color=64
endif else begin
if(ndepth gt 17) then begin
directory='/cis/grad/lab5756/3ddepth3/depth17/'
file=string(resultelement[elementcount])+'.tif'
full_filename=directory+file
if(query_tiff(full_filename)) then begin
image=read_tiff(full_filename)
y=congrid(image,3,256,256)
y2=y[*, 10:240, 40:215]
y3=congrid(y2, 3, 256, 256)
tvscl, y3, /order,true=1
xyouts, 25, 10, resultelement[elementcount], /device, charsize=1.5, charthick=1.0, color=64
xyouts, 150, 10, resultdepth[17], /device, charsize=1.5, charthick=1.0, color=64
endif else begin
tvscl, color_window2, /order, true=1
endelse
endif else begin
tvscl, color_window2, /order, true=1
endelse
endif else begin
53
tvsc1, color_window, /order, true=1
endelse
if(ndepth gt 6) then begin
window6=global_data.window_id6
wset, window6
directory='/cis/grad/lab5756/3ddepth3/depth6/'
file=string(resultelement[elementcount])+'.tif'
full_filename=directory+file
;first if statement to see if tiff file exists
if(query_tiff(full_filename)) then begin
image=read_tiff(full_filename)
y=congrid(image,3,256,256)
y2=y[*, 10:240, 40:215]
y3=congrid(y2, 3, 256, 256)
tvsc1, y3, /order,true=1
xyouts, 25, 10, resultelement[elementcount], /device, charsize=1.5, charthick=1.0, color=64
xyouts, 150, 10, resultdepth[6], /device, charsize=1.5, charthick=1.0, color=64
endif else begin
tvsc1, color_window2, /order, true=1
endelse
endif else begin
if(ndepth gt 18) then begin
directory='/cis/grad/lab5756/3ddepth3/depth18/'
file=string(resultelement[elementcount])+'.tif'
full_filename=directory+file
if(query_tiff(full_filename)) then begin
image=read_tiff(full_filename)
y=congrid(image,3,256,256)
y2=y[*, 10:240, 40:215]
y3=congrid(y2, 3, 256, 256)
tvsc1, y3, /order,true=1
xyouts, 25, 10, resultelement[elementcount], /device, charsize=1.5, charthick=1.0, color=64
xyouts, 150, 10, resultdepth[18], /device, charsize=1.5, charthick=1.0, color=64
endif else begin
tvsc1, color_window2, /order, true=1
endelse
endif else begin
tvsc1, color_window2, /order, true=1
endelse
endif else begin
tvsc1, color_window, /order, true=1
endelse
endif else begin
tvsc1, color_window2, /order, true=1
endelse
endif else begin
tvsc1, color_window, /order, true=1
endelse
endif else begin
if(ndepth gt 7) then begin
window7=global_data.window_id7
wset, window7
directory='/cis/grad/lab5756/3ddepth3/depth7/'
file=string(resultelement[elementcount])+'.tif'
full_filename=directory+file

if(query_tiff(full_filename)) then begin
    image=read_tiff(full_filename)
    y=congrid(image,3,256,256)
    y2=y[*, 10:240, 40:215]
    y3=congrid(y2, 3, 256, 256)
    tvscl, y3, /order,true=1
    xyouts, 25, 10, resultelement[elementcount], /device, charsize=1.5, charthick=1.0, color=64
    xyouts, 150, 10, resultdepth[7], /device, charsize=1.5, charthick=1.0, color=64
endif else begin
if(ndepth gt 19) then begin
    directory='/cis/grad/lab5756/3ddepth3/depth19/'
    file=string(resultelement[elementcount])+'.tif'
    full_filename=directory+file
    if(query_tiff(full_filename)) then begin
        image=read_tiff(full_filename)
        y=congrid(image,3,256,256)
        y2=y[*, 10:240, 40:215]
        y3=congrid(y2, 3, 256, 256)
        tvscl, y3, /order,true=1
        xyouts, 25, 10, resultelement[elementcount], /device, charsize=1.5, charthick=1.0, color=64
        xyouts, 150, 10, resultdepth[19], /device, charsize=1.5, charthick=1.0, color=64
    endif else begin
        tvscl, color_window2, /order, true=1
    endelse
endif else begin
    tvscl, color_window2, /order, true=1
endelse
endif else begin
if(ndepth gt 8) then begin
    window8=global_data.window_id8
    wset, window8
    directory='/cis/grad/lab5756/3ddepth3/depth8/
    file=string(resultelement[elementcount])+'.tif'
    full_filename=directory+file
    if(query_tiff(full_filename)) then begin
        image=read_tiff(full_filename)
        y=congrid(image,3,256,256)
        y2=y[*, 10:240, 40:215]
        y3=congrid(y2, 3, 256, 256)
        tvscl, y3, /order,true=1
        xyouts, 25, 10, resultelement[elementcount], /device, charsize=1.5, charthick=1.0, color=64
    endif else begin
        tvscl, color_window, /order, true=1
    endelse
endif else begin
    tvscl, color_window, /order, true=1
endelse
endif else begin
if(query_tiff(full_filename)) then begin
    image=read_tiff(full_filename)
    y=congrid(image,3,256,256)
    y2=y[*, 10:240, 40:215]
    y3=congrid(y2, 3, 256, 256)
    tvscl, y3, /order,true=1
    xyouts, 25, 10, resultelement[elementcount], /device, charsize=1.5, charthick=1.0, color=64
endelse
endif else begin
    tvscl, color_window, /order, true=1
endelse
xyouts, 150, 10, resultdepth[8], /device, charsize=1.5, charthick=1.0, color=64
endif else begin
if(ndepth gt 20) then begin
directory='"/cis/grad/lab5756/3ddepth3/depth20/"
file=string(resultelement[elementcount])+'.tif'
full_filename=directory+file
if(query_tiff(full_filename)) then begin
image=read_tiff(full_filename)
y=congrid(image, 3, 256, 256)
y2=y[*], 10:240, 40:215]
y3=congrid(y2, 3, 256, 256)
tvscl, y3, /order, true=1
xyouts, 25, 10, resultelement[elementcount], /device, charsize=1.5, charthick=1.0, color=64
xyouts, 150, 10, resultdepth[20], /device, charsize=1.5, charthick=1.0, color=64
endif else begin
tvscl, color_window2, /order, true=1
endelse
endif else begin
tvscl, color_window2, /order, true=1
endelse
endif else begin
tvscl, color_window, /order, true=1
endelse
endif else begin
if(ndepth gt 9) then begin
window9=global_data.window_id9
wset, window9
directory='"/cis/grad/lab5756/3ddepth3/depth9/"
file=string(resultelement[elementcount])+'.tif'
full_filename=directory+file
if(query_tiff(full_filename)) then begin
image=read_tiff(full_filename)
y=congrid(image, 3, 256, 256)
y2=y[*], 10:240, 40:215]
y3=congrid(y2, 3, 256, 256)
tvscl, y3, /order, true=1
xyouts, 25, 10, resultelement[elementcount], /device, charsize=1.5, charthick=1.0, color=64
xyouts, 150, 10, resultdepth[9], /device, charsize=1.5, charthick=1.0, color=64
endif else begin
if(ndepth gt 21) then begin
directory='"/cis/grad/lab5756/3ddepth3/depth21/"
file=string(resultelement[elementcount])+'.tif'
full_filename=directory+file
if(query_tiff(full_filename)) then begin
image=read_tiff(full_filename)
endelse
endif else begin
tvscl, color_window, /order, true=1
endelse
endif else begin
tvscl, color_window2, /order, true=1
endelse
endif else begin
tvscl, color_window, /order, true=1
endelse
y = congrid(image, 3, 256, 256)
y2 = y[*, 10:240, 40:215]
y3 = congrid(y2, 3, 256, 256)
tvscl, y3, /order, true=1
xyouts, 25, 10, resultelement[elementcount], /device, charsize=1.5, charthick=1.0, color=64
xyouts, 150, 10, resultdepth[21], /device, charsize=1.5, charthick=1.0, color=64
endif else begin
tvscl, color_window2, /order, true=1
endelse
endif else begin
tvscl, color_window2, /order, true=1
endelse
endelse
endif else begin
tvscl, color_window, /order, true=1
endelse
endif else begin
if(ndepth gt 10) then begin
window10 = global_data.window_id10
wset, window10
directory = '/cis/grad/lab5756/3ddepth3/depth10/'
file = string(resultelement[elementcount]) + '.tif'
full_filename = directory + file
; first if statement to see if tiff file exists
if(query_tiff(full_filename)) then begin
image = read_tiff(full_filename)
y = congrid(image, 3, 256, 256)
y2 = y[*, 10:240, 40:215]
y3 = congrid(y2, 3, 256, 256)
tvscl, y3, /order, true=1
xyouts, 25, 10, resultelement[elementcount], /device, charsize=1.5, charthick=1.0, color=64
xyouts, 150, 10, resultdepth[10], /device, charsize=1.5, charthick=1.0, color=64
endif else begin
if(ndepth gt 22) then begin
directory = '/cis/grad/lab5756/3ddepth3/depth22/'
file = string(resultelement[elementcount]) + '.tif'
full_filename = directory + file
if(query_tiff(full_filename)) then begin
image = read_tiff(full_filename)
y = congrid(image, 3, 256, 256)
y2 = y[*, 10:240, 40:215]
y3 = congrid(y2, 3, 256, 256)
tvscl, y3, /order, true=1
xyouts, 25, 10, resultelement[elementcount], /device, charsize=1.5, charthick=1.0, color=64
xyouts, 150, 10, resultdepth[22], /device, charsize=1.5, charthick=1.0, color=64
endif else begin
tvscl, color_window2, /order, true=1
endelse
endif else begin
tvscl, color_window2, /order, true=1
endelse
endif else begin
tvscl, color_window, /order, true=1
endelse
endif else begin
tvscl, color_window2, /order, true=1
endelse
endif else begin
tvscl, color_window, /order, true=1
endelse
endif else begin
tvscl, color_window2, /order, true=1
endelse
endif else begin
tvscl, color_window, /order, true=1
endelse
endif else begin
if(ndepth gt 11) then begin
window11=global_data.window_id11
wset, window11
directory='/cis/grad/lab5756/3ddepth3/depth11/'
file=string(resultelement[elementcount])+'.tif'
full_filename=directory+file
;if statement to see if tiff file exists
if(query_tiff(full_filename)) then begin
image=read_tiff(full_filename)
y=congrid(image,3,256,256)
y2=y[*, 10:240, 40:215]
y3=congrid(y2, 3, 256, 256)
tvscl, y3, /order,true=1
xyouts, 25, 10, resultelement[elementcount], /device, charsize=1.5, charthick=1.0, color=64
xyouts, 150, 10, resultdepth[11], /device, charsize=1.5, charthick=1.0, color=64
endif else begin
if(ndepth gt 23) then begin
directory='/cis/grad/lab5756/3ddepth3/depth23/'
file=string(resultelement[elementcount])+'.tif'
full_filename=directory+file
;if statement to see if tiff file exists
if(query_tiff(full_filename)) then begin
image=read_tiff(full_filename)
y=congrid(image,3,256,256)
y2=y[*, 10:240, 40:215]
y3=congrid(y2, 3, 256, 256)
tvscl, y3, /order,true=1
xyouts, 25, 10, resultelement[elementcount], /device, charsize=1.5, charthick=1.0, color=64
xyouts, 150, 10, resultdepth[23], /device, charsize=1.5, charthick=1.0, color=64
endif else begin
tvscl, color_window2, /order, true=1
endelse
endif else begin
tvscl, color_window2, /order, true=1
endelse
endif else begin
tvscl, color_window, /order, true=1
endelse
That ends the case statement. Below, the else defines what to do if the event.value did not match either of the case statements. The program will goto the line labeled jump; which is at the start of the program. At that point it will increment the count; and elementcount by 1 and see if that matches either of the cases. This cycle will; continue until the case statement has been matched.

else: goto, jump
endcode

; reinitialize the global data
new_data={draw_id:global_data.draw_id, window_id:global_data.window_id, $
  draw_id1:global_data.draw_id1, window_id1:global_data.window_id1, $
  draw_id2:global_data.draw_id2, window_id2:global_data.window_id2, $
  draw_id3:global_data.draw_id3, window_id3:global_data.window_id3, $
  draw_id4:global_data.draw_id4, window_id4:global_data.window_id4, $
  draw_id5:global_data.draw_id5, window_id5:global_data.window_id5, $
  draw_id6:global_data.draw_id6, window_id6:global_data.window_id6, $
  draw_id7:global_data.draw_id7, window_id7:global_data.window_id7, $
  draw_id8:global_data.draw_id8, window_id8:global_data.window_id8, $
  draw_id9:global_data.draw_id9, window_id9:global_data.window_id9, $
  draw_id10:global_data.draw_id10, window_id10:global_data.window_id10, $
  draw_id11:global_data.draw_id11, window_id11:global_data.window_id11}
end

; The event handler for the quit button which results in the program being closed
pro quit_button_event, event
  Widget_Control, event.top, /destroy
  close, /all
end

pro finalmanyimage4
; Create the base
  base=Widget_Base(column=5, Title='Depth Profile')
; create a label widget
  label=Widget_label(base, Value='Choose One')
; read in the element symbols from the text file created by the user
  resultelement=read_positionelement('element.dat', 'element2.dat')
; read in the depths from a text file created by the user
  resultdepth=read_positiondepth('depth.dat', 'depth2.dat')
; all of the element symbols and depths are placed in a string array called names
  names=[resultelement, resultdepth]
; Create a compound button widget with the label for each button coming from the
; names array. Note the buttons are exclusive, do not cause an event when released
; and return the button name when pressed.
  buttons=CW_bgroup(base, names, /exclusive, /return_name, /no_release)
; Create the quit button
button=Widget_Button(base, value='Quit', xsize=40, ysize=30, /align_center, event_pro='quit_button_event')
;Create 12 Draw Widgets
draw=Widget_Draw(base, xsize=256, ysize=256)
draw1=Widget_Draw(base, xsize=256, ysize=256)
draw2=Widget_Draw(base, xsize=256, ysize=256)
draw3=Widget_Draw(base, xsize=256, ysize=256)
draw4=Widget_Draw(base, xsize=256, ysize=256)
draw5=Widget_Draw(base, xsize=256, ysize=256)
draw6=Widget_Draw(base, xsize=256, ysize=256)
draw7=Widget_Draw(base, xsize=256, ysize=256)
draw8=Widget_Draw(base, xsize=256, ysize=256)
draw9=Widget_Draw(base, xsize=256, ysize=256)
draw10=Widget_Draw(base, xsize=256, ysize=256)
draw11=Widget_Draw(base, xsize=256, ysize=256)
;Realize the 12 draw widgets
Widget_control, draw, /realize
Widget_control, draw1, /realize
Widget_control, draw2, /realize
Widget_control, draw3, /realize
Widget_control, draw4, /realize
Widget_control, draw5, /realize
Widget_control, draw6, /realize
Widget_control, draw7, /realize
Widget_control, draw8, /realize
Widget_control, draw9, /realize
Widget_control, draw10, /realize
Widget_control, draw11, /realize
;Realize the base
Widget_Control, base, /realize
;Get the values for the draw widgets
Widget_Control, draw, get_value=window
Widget_Control, draw1, get_value=window1
Widget_Control, draw2, get_value=window2
Widget_Control, draw3, get_value=window3
Widget_Control, draw4, get_value=window4
Widget_Control, draw5, get_value=window5
Widget_Control, draw6, get_value=window6
Widget_Control, draw7, get_value=window7
Widget_Control, draw8, get_value=window8
Widget_Control, draw9, get_value=window9
Widget_Control, draw10, get_value=window10
Widget_Control, draw11, get_value=window11
;Save all of the draw_ids and window_ids into global data
global_data={draw_id:draw, window_id:window, draw_id1: draw1, window_id1:window1,$
draw_id2:draw2, window_id2:window2, draw_id3:draw3, window_id3:window3,$
draw_id4:draw4, window_id4:window4,$
draw_id5:draw5, window_id5:window5, draw_id6:draw6, window_id6:window6, 
        draw_id7:draw7, window_id7:window7, draw_id8:draw8, window_id8:window8, 
        draw_id9:draw9, window_id9:window9, draw_id10:draw10, window_id10:window10, 
        draw_id11:draw11, window_id11:window11}

widget_control, base, set_uvalue=global_data

;Specify the event handler for the compound button widgets
Xmanager,'finalmanyimage4',base
end