GRAPHICAL USER INTERFACE (GUI) LAB

This lab will guide you through the complex process of graphical user interface (GUI) creation. GUI's are interfaces computer users invoke to make computer programs easier to use. They provide a graphical means to perform simple and complex operations or procedures. Computer programmers make most of their applications as GUIs so users are not required to learn computer programming languages. We each use GUIs on a daily basis. Any computer program that implements buttons or menus to perform tasks is GUI based. Some examples include; Microsoft Word, ArcMap, ENVI, S-Plus, etc.

GUIs in IDL
In IDL there are two ways to create GUIs; manual script generation (writing code line by line as we have done in the previous labs) or semi-automatic script generation (this process uses a GUI already built into IDL to generate GUIs (this will make more sense later in the lab)).

Before we create a functional GUI we need to understand basic GUI architecture. GUIs are comprised of numerous widgets that interact to accomplish a task. Common widgets used in IDL include the base widget (widget_base), button widgets (widget_button), text widgets (widget_text), and label widgets (widget_label).

MANUAL GUI CREATION
Let's create a simple GUI (manually) to display a few basic concepts.

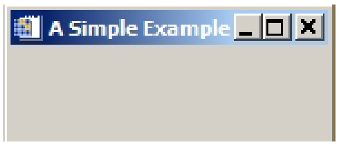
First we must create the base widget (the matrix within which all other widgets in the GUI are contained).

1. Use the widget_base function to create a base widget by typing the following code in the IDL editor window.

```
; creates a widget_base called base
Pro simp_widg
base = widget_base(XSIZE = 175, YSIZE =50, TITLE='A Simple Example')

;realize the widget
widget_control, base, /REALIZE
end
```

The XSIZE and YSIZE keywords specify the horizontal and vertical size (in pixels) of the base widget, while the TITLE keyword creates a title for the widget.

The second line of code invokes the GUI. Running the program should produce a GUI like this.



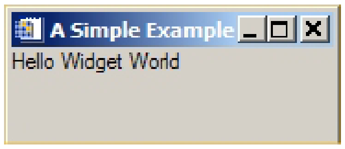
2. Add a label to the widget via the widget_label command. Type the following code between the previous line creating the base widget and the previous line realizing the widget.

label = widget_label (base, value = 'Hello Widget World')

This line of code adds a text label (named label) to the base widget.
Your program should now look like this:

```
Pro simp_widg
base = widget_base(XSIZE = 175, YSIZE =50, TITLE='A Simple Example')
label = widget_label (base, value = 'Hello Widget World')
widget_control, base, /REALIZE
end
```

Running the program should produce a GUI like this.



Task 1. Add a button to the GUI. Make the button text read Goodbye.

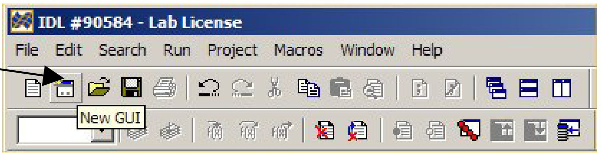
HINT: Look at the widget_button function (to create the button), the value keyword (for the button text), and the x and y offset keywords (for button placement).

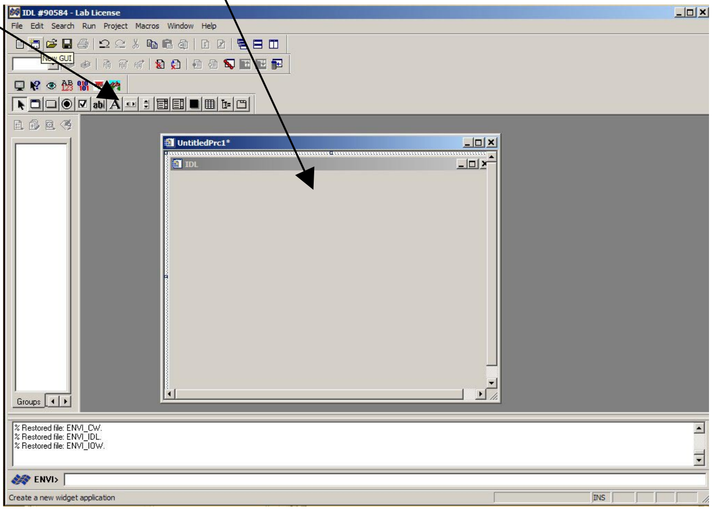
SEMI-AUTOMATIC GUI CREATION

As you can see creating a GUI manually could be very time consuming. One could spend hours experimenting with different widget positions and sizes to make a well designed GUI. Fortunately IDL (and many other programming platforms) have applications that allow a user to create GUIs graphically and generate code semi-automatically.

Let's create the same simple GUI (semi-automatically) to display a few basic concepts.

1. Click NEW GUI button on the IDL menu bar: This creates a base widget and activates the GUI Menu bar.

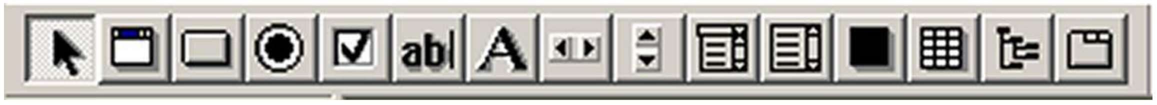



IDL #90584 - Lab License
File Edit Search Run Project Macros Window Help
UntitledPrc1*
IDL
Groups
ENVI>

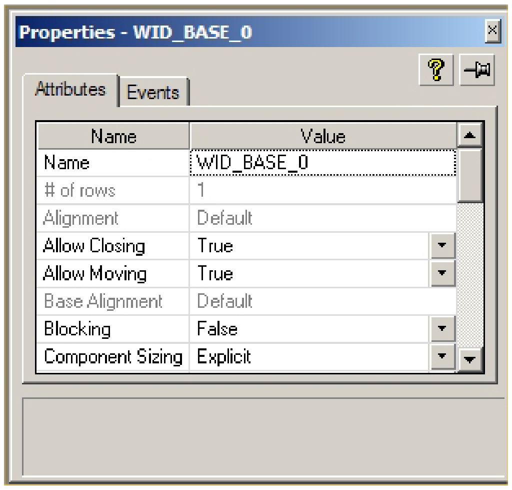
```
% Restored file: ENVI_CW.
% Restored file: ENVI_IDL.
% Restored file: ENVI_IOW.
```

Create a new widget application
INS

There are numerous widget buttons on the menu bar, including a button widget, a text widget, and a label widget. Hold your mouse pointer over the different button to see what they are.



2. Currently the base widget is titled IDL. To change this, we need to edit the base widget's properties. In the main IDL window right click on the base widget and select properties to display the widget property editor.

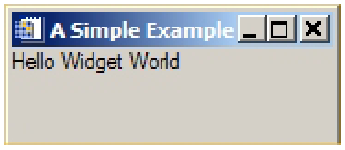


Name	Value
Name	WID_BASE_0
# of rows	1
Alignment	Default
Allow Closing	True
Allow Moving	True
Base Alignment	Default
Blocking	False
Component Sizing	Explicit

There are two tabs in the properties window (Attributes and Events). For now we are only concerned with the Attributes tab. Change the widget Name value from WID_BASE_0 to base. Next find the Title attribute and change it from IDL to A simple example. Close the properties dialog. The base widget should now be titled "A Simple Example".

3. Add a label to the base widget. This is accomplished by selecting the label widget function from the widget menu bar (represented by the capital letter A). Click on the base widget. A text box containing the word "label" should appear on the base widget. Drag it to an appropriate position. Open the labels properties menu (right click and select properties). Change the name attribute to label and the text value to Hello Widget World. Close the properties menu. The widget should now be titled A Simple Example and have a label that reads Hello Widget World. Resize the base widget by dragging the edges.

4. Generate IDL code. In order to run this widget IDL code must be generated. Select the GUI (UntiltePrc1*) and save it to a new directory (File – Save As). Name it sim_widg2.prc. To generate IDL code click generate.pro under the IDL file menu. Save the code to the same directory as the above .prc file. Name it sim_widg2.pro. IDL will create two new files (sim_widg2.pro and sim_widg2_eventcb.pro). Open both of the new files. Examine the code in sim_widg2.pro.Portions of the code should look similar to the code we generated in 1 and 2 above. Compile both files and run sim_widg2.pro. The GUI should appear. →

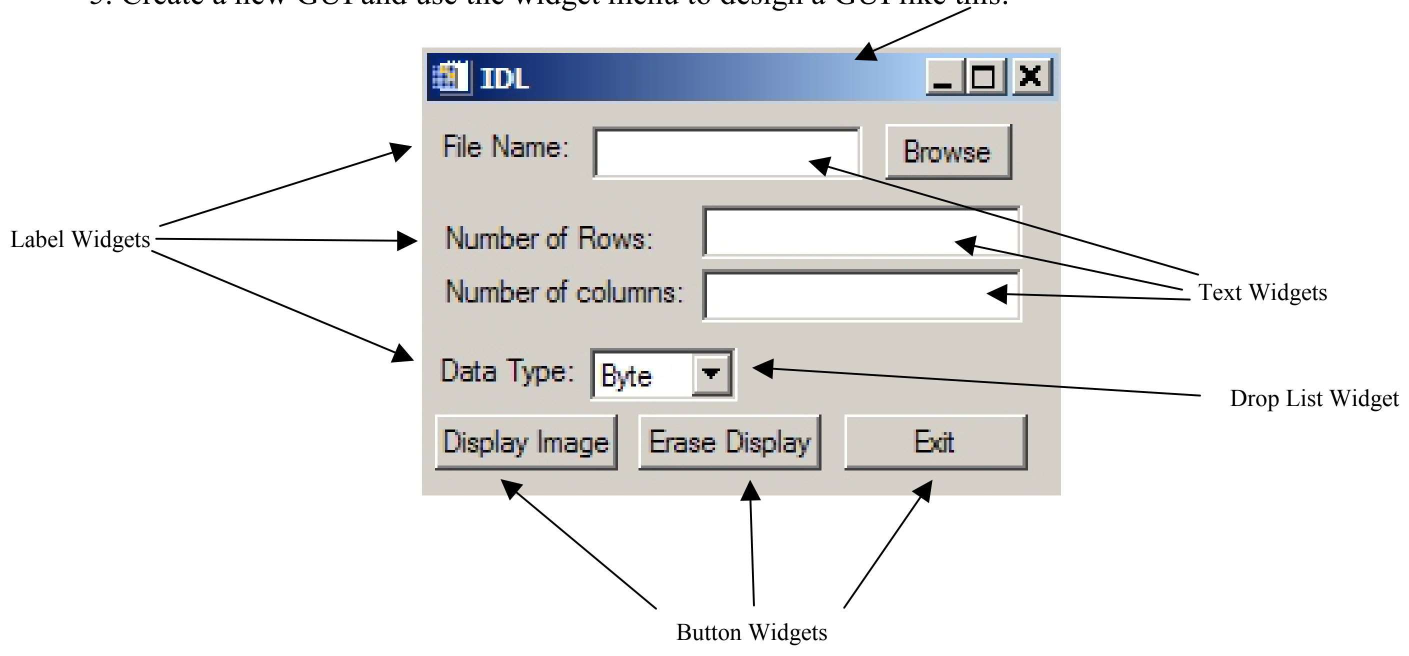


Task 2. Use the widget menu to add a button to the GUI. Make the button text read Goodbye.
HINT – use the button widget function and edit its label attribute.

CREATING A FUNCTIONAL GUI.

So far the GUIs you have created in this lab have limited functionality. The remainder of this lab will teach you how to create a functional GUI. Specifically, you will create a GUI that reads in an image file and displays it after the user specifies its dimensions and data type.

5. Create a new GUI and use the widget menu to design a GUI like this:

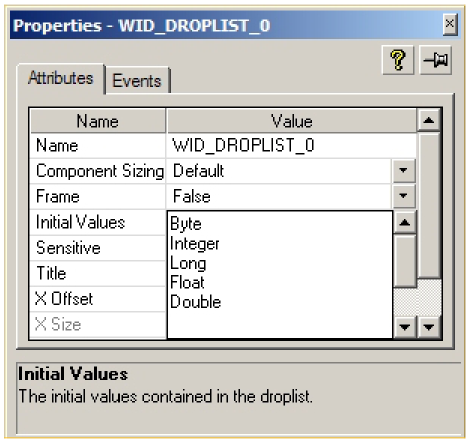


Helpful hints: Give each widget a unique name in the properties menu. For example name the Browse button wid_butt_browse, the Display Image button wid_butt_display, etc. Being consistent with the naming convention you use will make future tasks in this lab much easier.

In the text widget properties the Editable attribute must be changed from false (the default) to true.

Under the drop list widget properties add type the words Byte, Integer, Long, Float, and Double into the initial values attribute field. It should look like this when you are done.

Hint: press Ctrl+Enter after typing in a word to start a new line.



Name	Value
Name	WID_DROPLIST_0
Component Sizing	Default
Frame	False
Initial Values	Byte Integer Long Float Double
Sensitive	
Title	
X Offset	
X Size	

Initial Values
The initial values contained in the droplist.

6. In order to add functionality to the GUI we need to modify the events properties for each of the buttons. Click the events tab under the Browse buttons properties menu. Change the OnPress event value to OnPressBrowse and close the properties menu. Modify the OnPress event for each of the button widgets. Be sure to use a consistent naming convention (i.e. OnPressDisplay, OnPressExit, OnPressErase, etc.).

Modifying the OnPress event for each button tells the GUI to execute a program when the button is pressed. For example, pressing the Exit button tells the GUI to search for and execute program called OnPressExit, which we will write later in this lab.

After your GUI looks like the one above, and the button OnPress events have been modified, save the .prc file and generate the program (as in 4. above). The names for each file should be read_image.prc and read_image.pro. After the program has been geterate open the read_image.pro and read_image_eventcb.pro files.

7. Examine read_image.pro. Somewhere in the file you will see code that will generate each widget in the GUI. For example the code used to create the erase button should look like this:

```
WID_BUTT_ERASE = Widget_Button(WID_BASE_0, UNAME='WID_BUTT_ERASE'  $
    ,XOFFSET=83 ,YOFFSET=122 ,SCR_XSIZE=72 ,SCR_YSIZE=22  $
    ,/ALIGN_CENTER ,VALUE='Erase Display')
```

This code was automatically generated by IDL based upon the values you entered into the properties dialog for the erase button.

8. Examine read_image_eventcb.pro. This program is comprised of numerous smaller programs corresponding to each button event you modified in 6. above. For example, the program corresponding to the Browse button looks like this:

```
pro OnPressBrowse, Event

end
```

Code will need to be added to each event program to make it functional.

Compile both programs and run read_image.pro. Your GUI will appear. However, the buttons are not yet functional (nothing happens when you press them).

9a. Add code to make each button perform a task. Let’s start with the exit button. Open read_image_eventcb.pro and locate the following code:

```
pro OnPressExit, Event

end
```

Add the following line of code between the pro and end statements above:

```
Widget_Control, event.top, /DESTROY
```

This code tells IDL to close the topmost widget in the GUI hierarchy. This is the base widget in the GUI we have created. Compile both programs and run read_image.pro. Pressing the exit button should cause the GUI to close.

9b. OK that was the easy one. We will now modify the Browse button code so it reads in an image via the ***DIALOG_PICKFILE*** function (see lab 4). Locate the Browse button code in read_image_eventcb.pro.

```
pro OnPressBrowse, Event

end
```

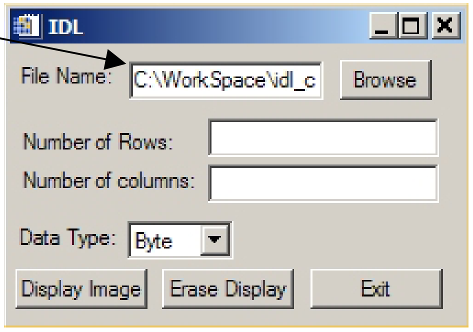
Modify the code to look like this:

```
pro OnPressBrowse, Event

;select image file via DIALOG_PICKFILE(see lab 4)
image1 = DIALOG_PICKFILE()

;Set the text box value equal to image location (variable name image1)
FileLocationText = WIDGET_INFO(event.TOP, FIND_BY_UNAME = 'WID_TEXT_FILE')
WIDGET_CONTROL, FileLocationText, SET_VALUE = image1

;save variables created so we can pass them to other events
SAVE, /VARIABLES, FILENAME = 'imagebutton_variables.sav'

End
```

Let's dissect the above code. The first line (***image1 = DIALOG_PICKFILE()***) creates a new variable (image1) and sets it equal to the path of file selected by the user (e.g., 'C:\ATRS\IMAGE1').

The next two lines are used to pass the path name (from the image1 variable) to the text widget named ***'WID_TEXT_FILE'***. A new variable (***FileLocationText***) containing information about the ***'WID_TEXT_FILE'*** widget is created via the ***WIDGET_INFO*** command. As written, the ***WIDGET_INFO*** command searches the base widget (***event.TOP***) for a widget with the unique name (***UNAME***) ***'WID_TEXT_FILE'.*** Note this name will correspond to the name you gave the file text widget in step 5 above. The ***WIDGET_CONTROL*** command sets the value attribute of the ***'WID_TEXT_FILE'*** widget equal to the image path (the image1 variable). The last line of code saves all the variables created in this portion of the code to a file named ***'imagebutton_variables.sav'***. IDL will not pass variables between sections of the read_imange_eventcb.pro. Saving them to a file will allow us to utilize the same variables for different buttons on the GUI.

Compile both programs and run read_image.pro. Your GUI will appear. Click on the browse button and browse the ATRS image we used in lab 3. After selecting the image the path name should appear the file name text field. The exit button should still work.



9c. Add functionality to the Display Image Button. Locate the code for the Display image event button. Modify the code to look like this:

```
pro OnPressDisplay, Event

RESTORE, 'imagebutton_variables.sav'

;get user values for image rows
RowWidg = WIDGET_INFO(event.TOP, FIND_BY_UNAME = 'WID_TEXT_ROWS')
WIDGET_CONTROL, RowWidg, GET_VALUE = nrows

; convert nrows variable from string to integer
nrows=FIX(nrows)

;get user values for image columns
ColWidg = WIDGET_INFO(event.TOP, FIND_BY_UNAME = 'WID_TEXT_COLS')
WIDGET_CONTROL, ColWidg, GET_VALUE = ncols

; convert ncols variable from string to integer
ncols=FIX(ncols)

;Read in file
file1 = READ_BINARY (image1, DATA_DIMS=[nrows, ncols], DATA_TYPE = 4)

;Display the image
TV, file1

End
```

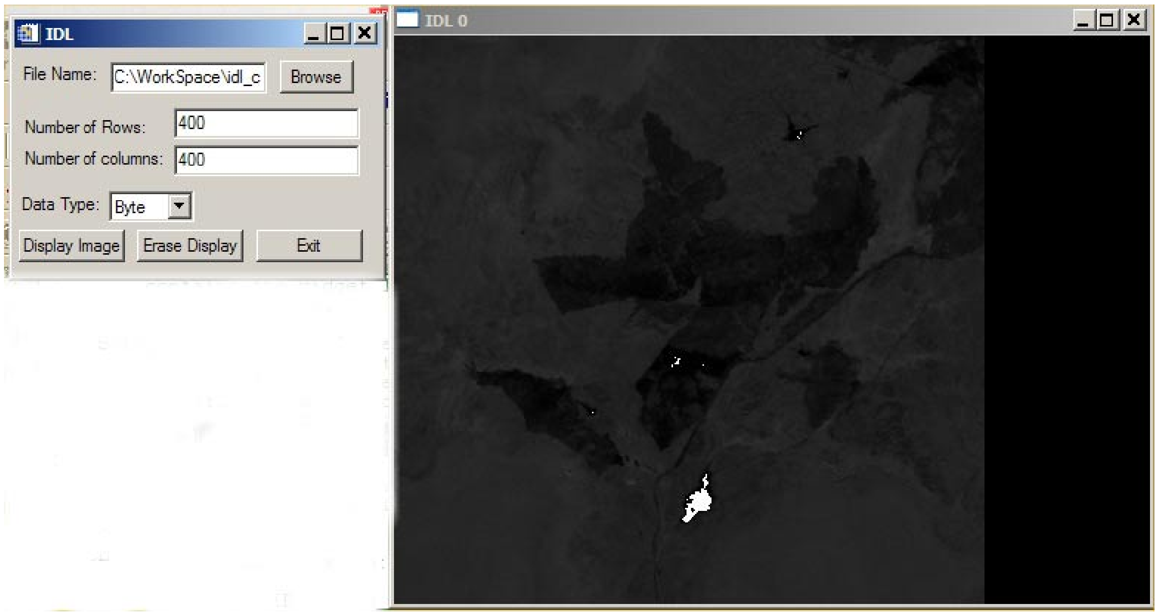
Code dissection:

The RESTORE command restores all variable created by the browse button in step 9b. The ***WIDGET_INFO*** command is used to obtain the number of rows and columns specified by the user. As in 9b., the code searches for a widget with a unique name (e.g., ***'WID_TEXT_ROWS'*** for the number of row text widget). The ***WIDGET_CONTROL*** command creates a new variable (e.g., ***nrows***) equal to the user input via the ***GET_VALUE*** keyword. User inputs are automatically set as strings. We want them as integers in our case, so we convert them from string to integer using the FIX command. The next line of code uses the ***READ_BINARY*** command to read in the file. A new variable (***file1***) is created. The ***image1*** variable is equal to the file path name created with the browse button (it was passed from step 9b. via the ***SAVE*** and ***RSTORE*** commands. The ***DATA_DIMS*** keyword specifies the number of rows and columns specified by the user (***nrows*** and ***ncols*** variables), and the ***DATA_TYPE*** keyword tells IDL the image is an INTEGER array (data type 4). Finally, the image is displayed in a graphics window using the TV command.

9d. The last guided step will be to make the Erase Display button functional. Locate the code responsible the Erase Display Button event. Modify it to look like this:

```
pro OnPressErase, Event

ERASE

End
```

Compile both programs and run read_image.pro. Your GUI will appear. Click on the browse button and browse the ATRS image we used in lab 3. After selecting the image the path name should appear the file name text field. Enter 400 in the Number of rows text field and 400 in the number of columns text field. Press the Display Image button. You image should appear in a graphics window. Pressing the Erase Image button should erase your image. The exit button will exit the GUI. Note – the data type drop down list is not yet functional.



Task 3. Currently the data type drop down list is not functional. Make it functional. Create a new variable (or variables) equal to the data type specified by the user. Pass the value of the variable to the DATA_TYPE keyword in READ_BINARY command in part 9c. This will not be an easy task as drop lists are the most difficult widgets to create in IDL. Use the IDL help menu. There are also numerous programs available on RSI's website the will help you with this task. Hint see table below for data type codes.

Type Code	Data Type	Type Code	Data Type
0	Undefined	8	Structure
1	Byte	9	Double-precision complex floating
2	Integer (16-bit)	10	Pointer
3	Longword integer (32-bit)	11	Object reference
4	Floating point	12	Unsigned integer (16-bit)
5	Double-precision floating	13	Unsigned longword integer (32-bit)
6	Complex floating	14	64-bit integer
7	String	15	Unsigned 64-bit integer