Modal Widgets

- Modal Widgets also known as pop-up widgets are a means to simplify a GUI interface by allowing an input window to be presented to the user at an appropriate time.
- Because a new window is created and destroyed, there needs to be a mechanism to pass the values specified to the used back to the calling widget.

Variables inside a Widget

- Remember that variables defined in a widget only exist as long as that widget exists.
- We have to somehow pass out any values defined to a calling function.

pop_fslider Case Study

- Problem: We want to create a simple slider that a user can call from the IDL prompt and return to the calling routine a floating point value reflecting the position of the floating point slider (CW_FSLIDER).

IDL> a = pop_fslider()

pop_fslider.pro

(initial version - Widget Definition)

function pop_fslider
  base = Widget_Base( /column )
  fslider = CW_Fslider( base, /edit )
  quit_button = Widget_Button( base, value="Quit", event_pro='quit_event' )
  Widget_Control, base, /realize
  global_data={ fslider_id: fslider, fslider_value: 0.0 }
  Widget_Control, base, set_uvalue=global_data
  Xmanager, 'pop_fslider', base, event_handler='pop_fslider_event'
  Widget_Control, base, get_uvalue=global_data
  return, global_data.fslider_value
end

(initial version output)

IDL> a = pop_fslider()

In Event Handler =       44.0000
In Event Handler =       39.0000
In Event Handler =       39.0000
In Event Handler =       42.0000
% WIDGET_CONTROL: Invalid widget identifier: 1.
% Execution halted at: POP_FSLIDER        28

pop_fslider.pro

(initial version - Event Handler Definition)

pro pop_fslider_event, event
  Widget_Control, event.top, get_uvalue=global_data
  Widget_Control, global_data.fslider_id, get_value=fslider_value
  global_data.fslider_value=fslider_value
  print,'In Event Handler =', global_data.fslider_value
  Widget_Control, event.top, set_uvalue=global_data
end

pro quit_event, event
  Widget_Control, event.top, /destroy
end
**pop_fslider.pro**
(initial version - Widget Definition)

```idl```
function pop_fslider
  base = Widget_Base( /column )
  fslider = CW_Fslider( base, /edit )
  quit_button = Widget_Button( base, value="Quit",
    event_pro='quit_event' )
  Widget_Control, base, /realize
  global_data={ fslider_id: fslider, fslider_value: 0.0 }
  Widget_Control, base, set_uvalue=global_data
  Xmanager, 'pop_fslider', base, event_handler='pop_fslider_event'
  Widget_Control, base, get_uvalue=global_data
  print,'In Widget Definition=', global_data.fslider_value
RETURN, global_data.fslider_value
```

**pop_fslider.pro**
(initial version - Event Definition)

- This does not work because the widget has already been destroyed and along with it the uvalue that was being stored
- To make this work, we need to use what IDL calls handles (or in C/C++ the concept of pointers)

**IDL Handles (Pointers)**

- **What are Handles?**
  - Handles (pointers) are special variables that point to memory locations that are explicitly allocated by the user.
  - It is different from other variables in that the memory location pointed to by a handle are persistent (i.e., they exist from one routine to another)
  - It is also the user’s responsibility to destroy the allocated memory

- **Uses of Handles**
  - Management of persistent data
  - Minimizes passing around large data structures
  - Linked lists data structures
  - Tree data structures

- **Routines for managing Handles**
  - **Handle_Create (Create a pointer)**
    IDL> image_pointer = Handle_Create()
  - **Handle_Value (Point a handle to data)**
    IDL> image = bindgen(5,5)
    IDL> handle_value, pointer, image, /set
  - **Handle_Value (Copy data pointed to by the handle to a new variable)**
    IDL> handle_value, pointer, a
Sample use of Handle Routines

IDL> image_pointer=handle_Create()
IDL> image=bindgen(5,5)
IDL> handle_value, image_pointer, image, /set
IDL> help, image
IMAGE     BYTE      = Array[5, 5]
IDL> help, image_pointer
IMAGE_POINTER   LONG      =            1
IDL> handle_value, image_pointer,a
IDL> help,a
A             BYTE      = Array[5, 5]

pop_fslider.pro
(new version - Widget Definition)

function pop_fslider
Widget_Control, base, /realize
pointer = Handle_Create()
global_data={ fslider_id: fslider, handle: pointer }
Widget_Control, base, set_uvalue=global_data
Xmanager, 'pop_fslider', base,
event_handler='pop_fslider_event'
Widget_Control, base, get_uvalue=global_data
print,'In Widget Definition=', global_data.fslider_value
handle_value, pointer, fslider_value
return, fslider_value
end

pop_fslider.pro
(new version - Event Handler Definition)

pro pop_fslider_event, event
Widget_Control, event.top, get_uvalue=global_data
Widget_Control, global_data.fslider_id,
get_value=fslider_value
pointer = global_data.handle
print,'In Event Handler =', fslider_value
handle_value, pointer, fslider_value, /set
end

pro quit_event, event
Widget_Control, event.top, /destroy
end

Result of pop_fslider.pro

IDL> a=pop_fslider()
In Event Handler = 40.0000
In Event Handler = 43.0000
In Event Handler = 42.0000
IDL> print,a
42.0000

CW_DEFROI

• An example of a pop-up compound widget is the CW_DEFROI
• It is a widget designed to provide a means to define a region of interest for selective processing.
### CW_DEFROI Example

**Problem:**
- Display an image
- Define a region of interest
- Make the negative of that region

**CW_DEFROI usage specifics**
- Works on a draw_widget
- draw_widget must be button and motion event enabled
- Returns indices of pixels defining region of interest

### CW_DEFROI Widget View

```pro
pro cw_defroi_example
  base = Widget_Base(/column)
  draw = Widget_Draw( base, xsize=256, ysize=256, /button_events,/motion_events )
  roi_button = Widget_Button( base, value='Define ROI', event_pro='roi_event')
  Widget_Control, base, /realize
  Widget_Control, draw, get_value=window
  image=bindgen(256,256)
  global_data={ draw_id: draw, window_id:window, image_data:image, roi_data: 0 };
  Widget_Control, base, set_uvalue=global_data
  Xmanager,'cw_defroi_example',base
end
```

### CW_DEFROI Event Handler

```pro
pro cw_defroi_example_event, event
end
pro roi_event, event
  Widget_Control,event.top,get_uvalue=global_data
  window=global_data.window_id & wset, window
  image=global_data.image_data & tvscl,image
  draw_id=global_data.draw_id
  roi=CW_DEFROI( draw_id )
  image(roi) = 1-image(roi) & tvscl, image
  new_data={draw_id:global_data.draw_id, window_id:global_data.window_id, $ image_data:image, roi_data:roi}
  Widget_Control,event.top,set_uvalue=new_data
end
```

### CW_DEFROI Widget Definition

```pro
pro cw_defroi_example
  base = Widget_Base(/column)
  draw = Widget_Draw( base, xsize=256, ysize=256, /button_events,/motion_events )
  roi_button = Widget_Button( base, value='Define ROI', event_pro='roi_event')
  Widget_Control, base, /realize
  Widget_Control, draw, get_value=window
  image=bindgen(256,256)
  global_data={ draw_id: draw, window_id:window, image_data:image, roi_data: 0 };
  Widget_Control, base, set_uvalue=global_data
  Xmanager,'cw_defroi_example',base
end
```