Subject: Re: how to OPLOT (x,y) over CW_ZOOM?
Posted by davidf on Sun, 09 May 1999 07:00:00 GMT
View Forum Message <> Reply to Message

Octavi Fors (octavi@fajnm1.am.ub.es) writes:

>> I should think it would be much easier to write
>> your own zoom window. And I would certainly write this
>> compound widget as an object if I were going to do it,
>> so that I could draw the points in the window by calling
>> the DRAW_POINTS method.
>
> I'm not familiar with objects, but if you have any template-code
> to begin with, I could work about.
>
> Where can I find DRAW_POINTS method? I don't know anything
> about it.

Humm. I was speaking too abstractly, I see.

But here is the object definition and INIT method of a
"smart" graphics window compound widget that Dick and I
implemented for a customer. The purpose of the window
was to lay any number of "plot objects" in a grid
in the window. Plus, the window had to have certain
"selection" methods. For example, it had to be able
to select one of the multiple plot objects for
processing, etc.

Because there were a lot of things we wanted to do
with the window (e.g., change the grid, release and
select plots, draw crosshairs, etc, etc.), it seemed
to make a lot more sense to write this as an object
rather than as a normal compound widget. This way,
we could have separate methods to do anything we liked
with the object without having to worry about the
traditional technique of "setting the value" of
a compound widget. For us, "getting the value" of the
compound widget simply means obtaining the window's
object reference. Once we have that, we can do anything
we like with the window by calling the appropriate
method. If we want more capability, we just add
another method. So it makes code development simple.

```
 ;---------------------------------------------------------- -------------
FUNCTION Plot_Window::Init, $
   parent, $                  ; The parent of the Plot Window top-level base.
   Plots=plots, $             ; A pointer to a vector of Plot Objects.
   XSize=xsize, $            ; The X size of the display window.
   YSize=ysize, $            ; The Y size of the display window.
   Grid=grid, $              ; The grid layout of the window.
   Color=color, $            ; The color index for drawing lines.
   Status=status, $          ; If set, include status window.
   CurrentTool=currentTool, $ ; The index of the currently selected tool.
   Tracking=tracking         ; If set, cursor tracking is on.

; This window method initializes the Window Object. The window is
; conceived as a compound widget. When you get the "value" of the
; compound widget, you are returned the window object. Instead of
; "setting values" of the compound widget, you call window object
; methods.

Catch, error
IF error NE 0 THEN BEGIN
   ok = Dialog_Message(['Error in INIT method:', !Err_String, 'Returning...'])
   RETURN, 0
ENDIF

   ; A parent parameter is required.

IF N_Params() EQ 0 THEN BEGIN
   ok = Dialog_Message('A "parent" parameter is required. Returning...')
   RETURN, -1
ENDIF

   ; Assign parameters.

self.color = color
self.grid = grid
self.fullGrid = grid
IF (grid[0] * grid[1]) EQ 1 THEN BEGIN
   self.oneplot = 1
   self.activePlot = (*plots)[0]
   self.activeNum = 0
ENDIF ELSE self.oneplot = 0
self.xsize = xsize
self.ysize = ysize
self.plots = plots
self.currentTool = currentTool
self.tracking = Keyword_Set(tracking)
self.drawEventName = 'Plot_Window_Draw_Events'

   ; Create the Plot Window widgets.

self.tlb = parent
self.drawID = Widget_Draw(self.tlb, XSize=self.xsize, YSize=self.ysize, $
   UValue=self, Notify_Realize='Plot_Window_Realize_DrawID', $
   Button_Events=1, Motion_Events=self.tracking, $
   Event_Pro='GA_Plot_Window_Draw_Events')

IF Keyword_Set(status) THEN BEGIN
   rowbase = Widget_Base(self.tlb, Row=1)
   labelID = Widget_Label(rowbase, Value='Status:')
   self.status = Widget_Text(rowbase, YSize=1, Scr_XSize=self.xsize - 50)
ENDIF

   ; Create a pixmap for erasing window drawing actions.

Window, XSize=self.xsize, YSize=self.ysize, /Pixmap, /Free
self.pixID = !D.Window

RETURN, 1
END
 ;------------------------------------------------------------ -------------



PRO Plot_Window__Define

; This is the object definition structure of the Plot_Window object.

struct = { Plot_Window, $
   tlb:0L, $               ; The pseudo-TLB of the compound widget.
   drawID:0L, $            ; The draw widget identifier.
   wid:0, $                ; The window index number of the draw widget.
   pixID:0, $              ; The window index number of the pixmap.
   xsize:0, $              ; The X size of the window.
   ysize:0, $              ; The Y size of the window.
   plots:Ptr_New(), $      ; A pointer to a vector of plot objects.
   grid:IntArr(2), $       ; The grid associated with this window.
   fullGrid:IntArr(2), $   ; The full grid to return to if it is changed
   color:0, $              ; Index to a cursor drawing color.
   oneplot:0, $            ; A flag to indicate a single plot in window.
   currentTool:0, $        ; The index of the active tool.
   tracking:0, $           ; A flag to indicate cursor tracking mode.
   currentSelection:Obj_New(), $ ; The currently selected plot object.
   selectionID:-1, $       ; The window number of the selection pixmap.
   selectionMix:-1, $      ; The window number of the mixing pixmap.
   buttonDown:0, $         ; A flag to indicate a window button is DOWN.
   status:-1L, $           ; Widget ID of status window.
   drawEventName:'', $     ; The name of the draw widget event handler.
   activePlot:Obj_New(), $  ; The currently active plot.
   activeNum:-1, $         ; The number of the currently active plot.
   sx:-1, $                ; The static X corner of a selection box.
   sy:-1 }                ; The static Y corner of a selection box.

END
 ;--------------------------------------------------------- -------------
```

Once we have the compound widget defined. We simply catch the events and dispatch them to the appropriate method. For example, here is the event handler for the draw widget in this compound widget:

```
 ;--------------------------------------------------------- -------------
PRO Plot_Window_Draw_Events, event

; This is the main draw widget event handler. Its purpose is
; to interpret and respond to draw widget events by calling
; the appropriate Window Object methods.

   ; Only respond to DOWN, MOTION, and UP events.

IF event.type GE 3 THEN RETURN

   ; Get the self object.

Widget_Control, event.id, Get_UValue=self

   ; What kind of an event?

thisEvent = (['DOWN', 'UP', 'MOTION'])[event.type]
CASE thisEvent OF
   'DOWN': self->DownButton, event.x, event.y
   'UP': self->UpButton, event.x, event.y
   'MOTION': self->CursorMotion, event.x, event.y
ENDCASE
END
 ;--------------------------------------------------------- --------
```

But we can also do other kinds of things. For example, we can get the window to print its contents to a PostScript file. All we do is get the "value" of the compound widget, which is the object reference, and call the PSFile method. The method just opens a PostScript file and draws all of the "plot objects" that are currently in the window to the PostScript file. The PSFile method looks like this:

```
 ;--------------------------------------------------------- --------
PRO Plot_Window::PSFile

; This method produces a PostScript file of the window contents.
```

```
   ; Method error handling.

Catch, error
IF error NE 0 THEN BEGIN
   Catch, /Cancel
   self->Error_Report, !Err_String
   RETURN
ENDIF

   ; Set up the PostScript device.

thisDevice = !D.Name
deviceKeywords = PSWindow()
Set_Plot, 'PS'
Device, _Extra=deviceKeywords, Filename='plot_window.ps', /Inches

   ; Draw the window plots.

FOR j=0,N_Elements(*self.plots)-1 DO (*self.plots)[j]->Draw

   ; Clean up.

Device, /Close
Set_Plot, thisDevice
END
 ;---------------------------------------------------------- --
```

This is probably not the "template" you were looking for,
but I think it illustrates the power of compound widgets
as objects.

The point, really, is that a zoom compound widget that
can draw boxes or points in its window, send itself
to a PostScript file, remember the current zoom
factor and perhaps which coordinate system it is in,
etc. is a pretty nice compound widget to have around. :-)

Hope this gives you some ideas, at least. This is the kind
of information I am putting into my current book. If it ever
gets finished, I'll let you know. :-(

Cheers,

David
--
David Fanning, Ph.D.
Fanning Software Consulting

Phone: 970-221-0438 E-Mail: davidf@dfanning.com
Coyote's Guide to IDL Programming: http://www.dfanning.com/
Toll-Free IDL Book Orders: 1-888-461-0155