
Subject: Programs that Work on 24-bit Color Displays (Long)

Posted by [davidf](#) on Sun, 08 Nov 1998 08:00:00 GMT

[View Forum Message](#) <> [Reply to Message](#)

Hi Folks,

By far and away the most useful program on my web page is XCOLORS. It has a number of useful features, but one of the most important is that it can be used to write programs that work correctly on 24-bit color displays. People are still confused (or unaware) of how this works, so I thought I would write a short tutorial.

The XCOLORS program itself can be found here:

<http://www.dfanning.com/programs/xcolors.pro>

The problem with a 24-bit color display is that colors are expressed directly and are not coupled to indices in the color table, as they are on an 8-bit color display. Thus, when you change the color table values (using, for example, a program like XLOADCT) the colors in your display windows remain unchanged. To update the colors in the window, you must re-execute the graphics command that put the graphic in the window in the first place. IDL will--as part of the process of displaying the graphic--run the pixel values through the color table vectors to get the specific color for the graphic.

So, quite simply, to make programs on 24-bit displays act like the same programs on 8-bit displays you have to know when the color table vectors have been changed so that you can re-display your graphic. But a program like XLOADCT can't notify you when you it changes the color table vectors. (Although I hear that the IDL 5.2 version will, finally, be able to do this, although it doesn't yet in the beta versions I have seen.)

The program XCOLORS *can* notify you. It is currently set up to do this in two different ways. First, it works in widget programs by notifying a widget of your choice. It does this by sending that widget an event structure (i.e., it works like all other widgets work). The event handler for that widget gets this event structure and can do whatever is

appropriate with it.

The event structure is defined like this:

```
event = { XCOLORS_LOAD, ID:0L, TOP:0L, HANDLER:0L, $  
R:BytArr(256), G:BytArr(256), B:BytArr(256) }
```

What you have to do is tell XCOLORS what values to put in the ID and TOP fields of this structure. The R, G, and B fields have the current color table vectors in them. You can use this information or not. Quite often that information can be safely ignored.

Here is an example of how it works. Here is a short widget program that displays a shaded surface. This is the widget definition module for the program. Save this program in a file named "shader.pro".

```
*****  
PRO Shader, data  
  
; Need data?  
  
IF N_Elements(data) EQ 0 THEN data = Dist(30)  
  
; 24 Bit Color Display?  
  
Device, Get_Visual_Depth=thisDepth  
IF thisDepth GT 8 THEN BEGIN  
    Device, Decomposed=0  
    truecolor = 1  
ENDIF ELSE truecolor = 0  
  
; Load color table.  
  
LoadCT, 5, /Silent  
  
; Create widgets.  
  
tlb = Widget_Base(Column=1)  
colors = Widget_Button(tlb, Value='XColors', $  
    Event_Pro='Shader_Colors')  
drawID = Widget_Draw(tlb, XSize=400, YSize=400)  
Widget_Control, tlb, /Realize  
  
; Get window index number, display graphic.  
  
Widget_Control, drawID, Get_Value=wid
```

```

WSet, wid
Shade_Surf, data

; Store info structure.

info = {data:data, wid:wid, truecolor:truecolor}
Widget_Control, tlb, Set_UValue=info, /No_Copy

; Event loop.

XManager, 'shader', tlb, /No_Block
END
*****

```

This program, Shader, has a button that loads different color tables by calling XCOLORS. How should that event handler be written?

Well, it should be written so that it handles two possible events: the one from the XCOLORS button and a second one that comes from XCOLORS when it loads new color table vectors.

Here is the event handler. (Add this to the "shader.pro" file, but be sure you place it FIRST in the file, before the SHADER procedure. Do you know why?)

```

*****
PRO Shader_Colors, event

; What kind of event is this?

thisEvent = Tag_Names(event, /Structure_Name)

; Do the right thing. :-)

CASE thisEvent OF

'WIDGET_BUTTON': XColors, Group_Leader=event.top, $
    NotifyID=[event.id, event.top]

'XCOLORS_LOAD': BEGIN
    Widget_Control, event.top, Get_UValue=info, /No_Copy
    IF info.truecolor THEN BEGIN
        WSet, info.wid
        Shade_Surf, info.data
    ENDIF
    Widget_Control, event.top, Set_UValue=info, /No_Copy

```

ENDCASE

ENDCASE

END

Notice that if the event is a button event that XCOLORS is called with the NOTIFYID keyword set. The first widget identified in this two-element vector is the one that will fill the ID field of the XCOLORS_LOAD event structure. The second widget in the two-element vector identifies the widget that will be placed in the TOP field of that event structure.

If the event structure in this event handler is an XCOLORS_LOAD event structure and the display is a 24-bit display (indicated by the truecolor flag, which was set in the widget definition module), then the graphic is re-displayed.

Try running this program on a 24-bit color display. It will act identically to the same program run on an 8-bit display. (There may be a short delay while the event is processed, but this should be hardly noticeable.)

The second way XCOLORS is useful is with objects. I write a lot of graphic objects these days and I am often want to use color tables with these objects. In fact, I often want to work with these objects at the IDL command line.

Here is an extremely simple object that does a shaded surface plot like the program above. (Save this code in a file named "shader__define.pro". Note, there are **two** underscore characters in that name.)

```
PRO Shader::Draw
Shade_Surf, *self.data
END;-----
```

```
PRO Shader::CleanUp
Ptr_Free, self.data
END;-----
```

```

FUNCTION Shader::lint, data
IF N_Elements(data) EQ 0 THEN data = Dist(30)
self.data = Ptr_New(data)
RETURN, 1
END;-----

```

```

PRO Shader__Define
struct = {SHADER, data:Ptr_New()}
END;-----

```

On a 24-bit color display I use the object like this:

```

IDL> Device, Decomposed=0
IDL> thisObject = Obj_New('SHADER')
IDL> Window
IDL> thisObject->Draw

```

Now, what I want is to load different color tables and see the shaded surface updated immediately. In this case, I use the NOTIFYOBJ keyword to XCOLORS. The value of this keyword must be a structure that has three fields: OBJECT, METHOD, and WID. The OBJECT field should have an object reference; the METHOD field should have the name of a method of that object I want to call when the color vectors change (in this case, I want the "Draw" method to be called); and the WID field should have the graphic window index number of the window I want the graphic output to be displayed in. I call XCOLORS like this:

```

IDL> XCOLORS, NotifyObj={OBJECT:thisObject, $
    Method:'DRAW', WID:!D.Window}

```

There you go! A color table loading tool that works on 24-bit displays just like it works on 8-bit displays. :-)

I'll leave it to you to discover the other useful properties of XCOLORS, but I'll give you a hint: it doesn't use COMMON blocks. How could that be useful, do you think?

Cheers,

David

David Fanning, Ph.D.
Fanning Software Consulting
E-Mail: davidf@dfanning.com
Phone: 970-221-0438, Toll-Free Book Orders: 1-888-461-0155
Coyote's Guide to IDL Programming: <http://www.dfanning.com/>
