
Subject: Re: Save 2D conversion matrix

Posted by peter.albert@gmx.de on Wed, 09 Nov 2005 10:36:09 GMT

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

Hi David,

> Uh, well, it is Button and Motion events (along with their
> cousins PRESS and RELEASE) that actually mirror and extend

o.k., thanks. Actually, a long while ago I dropped working with draw widgets because I wanted to be able to resize the draw window, which appeared to be not actually trivial with draw widgets. Then there came FSC_WINDOW and resizing the window as no longer a problem :-)

However, now I am using a plot command which can take several seconds to be completed. The current version of FSC_WINDOW sets the KBRD_Focus_Events keyword to WIDGET_CONTROL to 1, with the consequence that dragging the window accross the screen can easily take several minutes... Well, this can be changed, and now I also wanted button and motion events. Imho the appropriate event handling routine should not be within the FSC_WINDOW source code, as any user might do something different, so I added a keyword USER_EVENT_PRO, which should be the name of a routine which just takes one parameter, namely the event itself.

So there are 4 new keywords: BUTTON_EVENTS and MOTION_EVENTS, which are passed through to WIDGET_DRAW, USER_EVENT_PRO which is used in the main event handler, and KBRD_Focus_Events, which is passed through to WIDGET_CONTROL.

I added the modified code in case you think it's a useful extension.

Cheers,

Peter

```
;+
; NAME:
;   FSC_WINDOW
;
; PURPOSE:
;
;   This routine implements a "smart" resizeable graphics window.
;   It is used as a wrapper for built-in IDL graphics procedures
;   such as SURFACE, CONTOUR, PLOT, SHADE_SURF, etc. In additon,
;   it can be used to display any user-written graphics procedure
```

so long as that procedure follows three simple rules: (1) It does not open its own graphics windows, (2) It is defined with no more than three positional arguments (an unlimited number of keyword arguments are allowed), and (3) It uses no device-specific commands, such as "WSet", "Device, Decomposed=1", etc.

Keyword arguments permit the window to have its own portion of a color table and to be able to change the colors loaded in that portion of the color table. Colors are updated automatically on both 8-bit and 24-bit color displays. In addition, the window colors will "protect" themselves. I mean by this that the window will re-load its own colors into the color table when the window gains keyboard focus. This prevents other applications from changing the colors used to display data in this window. (This is an issue mainly in IDL 5 applications where widget applications can run concurrently with commands from the IDL command line.)

Keyword arguments also permit the window to create output files of its contents. These files can be color and gray-scale PostScript, and color BMP, GIF, JPEG, PICT, PNG, TIFF, or JPEG files. Output can also be sent directly to the default printer.

AUTHOR:

FANNING SOFTWARE CONSULTING
David Fanning, Ph.D.
1645 Sheely Drive
Fort Collins, CO 80526 USA
Phone: 970-221-0438
E-mail: davidf@dfanning.com
Coyote's Guide to IDL Programming: <http://www.dfanning.com>

CATEGORY:

Widgets, Graphics.

CALLING SEQUENCE:

FSC_WINDOW, command, P1, P2, P3

REQUIRED INPUTS:

COMMAND: The graphics procedure command to be executed. This parameter

must be a STRING and the command must be a procedure.

Examples

```

; are 'SURFACE', 'CONTOUR', 'PLOT', etc.
;
;
; OPTIONAL INPUTS:
;
; P1: The first positional parameter appropriate for the graphics
; command.
;
; P2: The second positional parameter appropriate for the
graphics
; command.
;
; P3: The third positional parameter appropriate for the graphics
; command.
;
; INPUT KEYWORD PARAMETERS:
;
; WBACKGROUND: The background color index for the window. Setting
this color
; along with the WERASEIT keyword causes the window to be
erased with
; this color. Set to !P.Background by default.
;
; WERASEIT: Setting this keyword "erases" the contents of the
current
; graphics window before re-executing the graphics command. For
example,
; this keyword might need to be set if the graphics "command" is
TVSCL.
; The default is to NOT erase the display before reissuing the
graphics
; command.
;
; _EXTRA: This keyword forms an anonymous structure of any
unrecognized
; keywords passed to the program. The keywords must be
appropriate
; for the graphics command being executed.
;
; GROUP_LEADER: The group leader for this program. When the group
leader
; is destroyed, this program will be destroyed.
;
; METHOD: Set this keyword to indicate that the method of an
object
; should be called, instead of a graphics procedure command. If
this
; keyword is set, the COMMAND parameter should be the name of an
object

```

```

;    procedure method, and the P1 parameter MUST be an object
reference.
;
;    TVORDER: This keyword corresponds the the !Order system
variable. It
;    is not used in this program, but is carried along for the call
to
;    TVREAD when windows are saved as ouput files. It will affect
the
;    transfer of window contents into the output data file. It
should be
;    used if the output file contents appear upside down.
;
;    WTITLE: This is the window title. It is the string "COMMAND
Window (1)"
;    by default, where COMMAND is the input parameter. And the
number
;    (1 in this case) is the window index number of the draw widget.
;
;    WXPOS: This is the initial X offset of the window. Default is
to
;    position the window in the approximate middle of the display.
;
;    WYPOS: This is the initial Y offset of the window. Default is
to
;    position the window in the approximate middle of the display.
;
;    WPOSTSCRIPT: Set this keyword to 1 to include a PostScript File
button under
;    the Save As button. This keyword is set automatically on 24-bit
display
;    devices. To turn the button OFF on 24-bit devices, set the
keyword value to 0.
;    There is no guaranteed way to create perfect PostScript output
when the program
;    is run on 8-bit displays. This will depend entirely on how the
"graphics command"
;    is written. Hence the button is turned off automatically on
8-bit devices.
;
;    WPRINT: Set this keyword to 1 to include a Print button under
the File button.
;    This keyword is set automatically on 24-bit display devices. To
turn the
;    button OFF on 24-bit devices, set the keyword value to 0. There
is no
;    guaranteed way to print output correctly when the program is
run on

```

```

; 8-bit displays. This will depend entirely on how the "graphics
command"
; is written. Hence the button is turned off automatically on
8-bit devices.
;
; WXSIZ: This is the initial X size of the window. Default is
400
; pixels.
;
; WYSIZ: This is the initial Y size of the window. Default is
400
; pixels.
;
; WCOLORS: Using this keyword adds a "Colors..." button to the
; "File" menu. Set this keyword to the number of colors available
; in the window and the starting index of the first color. For
example,
; to allow the window access to 100 colors, starting at color
index 50
; (i.e., color indices 50 to 149), use WColors=[100, 50]. If you
use the
; keyword syntax "/WColors", all the colors available will be
used, not just
; one color. If the keyword is set to a scalar value greater than
1, the
; starting color index is set to 0. The default value for this
keyword
; is [(!D.Table_Size, 0].
;
; KBRD_FOCUS_EVENTS: This keyword is passed through to
; WIDGET_CONTROL. If set, events are issued whenever the keyboard
focus
; of the widget changes. ATTENTION: Until now, the default value
of this
; keyword was 1!
;
; BUTTON_EVENTS: If set, the draw widget will create button
events.
;
; MOTION_EVENTS: If set, the draw widget will create motion
events.
;
; USER_EVENT_PRO: If either BUTTON_EVENTS or MOTION_EVENTS are
set, the
; appropriate events must be passed to an external routine. The
only
; parameter of this routine is the event itself.
;
;

```

```

; COMMON BLOCKS:
;
;   None.
;
; RESTRICTIONS:
;
;   This program requires additional programs from the Fanning
;   Software Consulting library:
;
;   CENTERTLB.PRO
;   ERROR_MESSAGE.PRO
;   FSC_PSCONFIG__DEFINE.PRO
;   FSC_DROPLIST.PRO
;   FSC_FIELD.PRO
;   FSC_FILESELECT.PRO
;   FSC_INPUTFIELD.PRO
;   FSC_PLOTWINDOW.PRO
;   PSCONFIG.PRO
;   PSWINDOW.PRO
;   TVREAD.PRO
;   XCOLORS.PRO
;
;   If the "command" program requires keywords that are also
keywords
;   to FSC_WINDOW, then you must use the keyword twice on the
command line.
;
; EXAMPLE:
;
;   If the program is called with no parameters whatsoever, it will
load
;   example data.
;
;   IDL> FSC_WINDOW
;
;   To use the program with an IDL PLOT command, for example:
;
;   IDL> FSC_WINDOW, 'PLOT', Findgen(11), Charsize=1.5,
Title='Example Plot'
;
;   To build your own graphics display command, you can do
something like this.
;   Here is a command program that takes an image, a column number,
and a row number,
;   and plots a column and row profile next to one another:
;
;   PRO COL_ROW_PLOT, image, column, row, _Extra=extra
;   ; Check parameters.

```

```

;      IF N_Elements(image) EQ 0 THEN image = DIST(200)
;      IF N_Elements(column) EQ 0 THEN column = 100
;      IF N_Elements(row) EQ 0 THEN row = 100
;      ; Set up plots.;
;      !P.Multi = [0, 2, 1]
;      Plot, image[column, *], Title='Row Profile',
YRange=[Min(image), Max(image)], $
;      XStyle=1, XTitle='At Column No: ' + StrTrim(column,2),
Extra=extra
;      Plot, image[* , row], Title='Column Profile',
YRange=[Min(image), Max(image)], $
;      XStyle=1, XTitle='At Row No: ' + StrTrim(row,2),
Extra=extra
;      !P.Multi = 0
;      END
;
;      This command program is used with FSC_WINDOW, like this:
;
;      IDL> Demo_GetData, image, Filename='ctscan.dat'
;      IDL> FSC_WINDOW, 'COL_ROW_PLOT', image, 30, 185, YTitle='Image
Value'
;
; TIPS FOR WRITING GRAPHICS DISPLAY PROGRAMS TO USE WITH FSC_WINDOW:
;
;      It is *exceedingly* difficult to write a graphics display
routine that uses
;      color and get it to display properly on your display, in a
PostScript file, and
;      when printed. This is because: (1) your display is a 24-bit
device and the PostScript
;      and PRINTER devices are both 8-bit devices, and (2) the printer
is different from your
;      display and PostScript device in not being able to load colors
on the fly. (A single
;      color table can only be loaded when you SET_PLOT to a PRINTER,
and then the colors
;      cannot be changed.)
;
;      Since FSC_Window knows *nothing* about your graphics display
routine, the chances of
;      it doing all three of these things correctly are just about
zero, unless you follow
;      the recommendations below. (In which case, your changes improve
to about 50/50.)
;
;      RECOMMENDATIONS
;
;      1. Use FSC_COLOR to specify your colors. If you don't do this,

```

```

please don't call
;      me for help. This is the FIRST thing I will recommend you
try. :-) FSC_COLOR *exists* to
;      solve these particular problems! And _get the latest
version_! Things may have changed
;      since you last downloaded it.
;
;
;      2. Consider using a white background color for your graphics
display. This is what
;      you are going to get with PostScript whether you like it or
not. It will make things
;      a LOT simpler for you to do it this way. Otherwise, it is up
to you to write your
;      graphics display program in such a way that if you are in
the PostScript device, you
;      will fill the page with the background color *before* you
draw your graphics. Something
;      like this:
;
;
;      IF !D.Name EQ 'PS' THEN Polyfill, [0,0,1,1, 0],
[0,1,1,0,0], /Normal, $
;      Color=FSC_Color(backgroundColor)
;      Plot, data, Color=FSC_Color(plotColor),
Background=FSC_Color(backgroundColor)
;
;
;      3. If you plan to print the contents of your FSC_WINDOW, you
*must* load your
;      colors *immediately* before you call the FSC_WINDOW program.
This will ensure
;      that the program will load *these* colors before it calls
the PRINTER device.
;      Unless you have specific requirements, I would load the
colors like this:
;
;
;      TVLCT, FSC_Color(/AllColors, /Triple, NColors=ncolors),
!D.table_Size - ncolors - 2
;      FSC_Window, 'yourprogram', ...
;
;
;      4. In your graphics display program, use FSC_Color to specify
*all* colors and DO *NOT*
;      use a color index parameter in the call. (The colorIndex
parameter is the second
;      positional parameter to FSC_COLOR.) You code might look
something like this:
;
;
;      Plot, mydata, Color=FSC_Color('Dodger Blue'),
Background=FSC_Color('White')
;
;

```

```

;   Good luck! If you have any problems (and you have followed
recommendation 1 already),
;   then please contact me for help.
;
; MODIFICATION HISTORY:
;
;   Written by: David Fanning, Sept 2000. Based on previous XWINDOW
program.
;   Whoops! Left out the line to resize draw widgets on UNIX
machines. Fixed. 12 Oct 2000, DWF.
;   Removed support for GIF files for IDL 5.4. 18 Jan 2001. DWF.
;   Beefed up documentation. 27 March 2001. DWF.
;   Added TVORDER keyword. 25 March 2002. DWF.
;   Added METHOD keyword so that an object method could be used as the
graphics
;   display routine name. 6 July 2003 KaRo
;   Added tips for writing graphics display programs. 26 Aug 2004.
DWF.
;-
;
; #####
;
; LICENSE
;
; This software is OSI Certified Open Source Software.
; OSI Certified is a certification mark of the Open Source Initiative.
;
; Copyright © 2000-2004 Fanning Software Consulting.
;
; This software is provided "as-is", without any express or
; implied warranty. In no event will the authors be held liable
; for any damages arising from the use of this software.
;
; Permission is granted to anyone to use this software for any
; purpose, including commercial applications, and to alter it and
; redistribute it freely, subject to the following restrictions:
;
; 1. The origin of this software must not be misrepresented; you must
;    not claim you wrote the original software. If you use this
software
;    in a product, an acknowledgment in the product documentation
;    would be appreciated, but is not required.
;
; 2. Altered source versions must be plainly marked as such, and must
;    not be misrepresented as being the original software.
;
; 3. This notice may not be removed or altered from any source
distribution.

```

```

;
; For more information on Open Source Software, visit the Open Source
; web site: http://www.opensource.org.
;
; #####

```

PRO FSC_Window_Execute, info

; This module executes the command.

; Error handling for all trapped errors.

```

Catch, theError
IF theError NE 0 THEN BEGIN
    Catch, /Cancel
    ok = Error_Message(Traceback=1, /Error)
    RETURN
ENDIF

```

; Current window, if supported.

IF (!D.Flags AND 256) NE 0 THEN WSet, info.wid

; Color protection on? Load color vectors if appropriate.

TVLCT, info.r, info.g, info.b

; Need an erase before drawing graphics?

IF info.weraseit THEN Erase, Color=info.wbackground

; Execute the command, based on the number of parameters
; and whether the keyword pointer points to a valid variable.

```

if info.cmdStruct.type eq 0 then $
IF N_Elements(*info.cmdStruct.keywords) EQ 0 THEN BEGIN
    CASE info.cmdStruct.nparams OF
        0: Call_Procedure, info.cmdStruct.command
        1: Call_Procedure, info.cmdStruct.command, *info.cmdStruct.p1
        2: Call_Procedure, info.cmdStruct.command, *info.cmdStruct.p1,
*info.cmdStruct.p2
        3: Call_Procedure, info.cmdStruct.command, *info.cmdStruct.p1,
*info.cmdStruct.p2, $
        *info.cmdStruct.p3
    ENDCASE
ENDIF ELSE BEGIN
    CASE info.cmdStruct.nparams OF

```

```

    0: Call_Procedure, info.cmdStruct.command,
    _Extra=*info.cmdStruct.keywords
    1: Call_Procedure, info.cmdStruct.command, *info.cmdStruct.p1,
    _Extra=*info.cmdStruct.keywords
    2: Call_Procedure, info.cmdStruct.command, *info.cmdStruct.p1,
    *info.cmdStruct.p2, $
        _Extra=*info.cmdStruct.keywords
    3: Call_Procedure, info.cmdStruct.command, *info.cmdStruct.p1,
    *info.cmdStruct.p2, $
        *info.cmdStruct.p3, _Extra=*info.cmdStruct.keywords
    ENDCASE
ENDELSE $
ELSE $
IF N_Elements(*info.cmdStruct.keywords) EQ 0 THEN BEGIN
    CASE info.cmdStruct.nparams OF
        0: Call_Method, info.cmdStruct.command
        1: Call_Method, info.cmdStruct.command, *info.cmdStruct.p1
        2: Call_Method, info.cmdStruct.command, *info.cmdStruct.p1,
        *info.cmdStruct.p2
        3: Call_Method, info.cmdStruct.command, *info.cmdStruct.p1,
        *info.cmdStruct.p2, $
            *info.cmdStruct.p3
    ENDCASE
ENDIF ELSE BEGIN
    CASE info.cmdStruct.nparams OF
        0: Call_Method, info.cmdStruct.command,
        _Extra=*info.cmdStruct.keywords
        1: Call_Method, info.cmdStruct.command, *info.cmdStruct.p1,
        _Extra=*info.cmdStruct.keywords
        2: Call_Method, info.cmdStruct.command, *info.cmdStruct.p1,
        *info.cmdStruct.p2, $
            _Extra=*info.cmdStruct.keywords
        3: Call_Method, info.cmdStruct.command, *info.cmdStruct.p1,
        *info.cmdStruct.p2, $
            *info.cmdStruct.p3, _Extra=*info.cmdStruct.keywords
    ENDCASE
ENDELSE

END
;-----

```

PRO FSC_Window_Example, _Extra=extra

; An example graphics display routine. Called if no parameters
; are passed to FSC_Window.

```

; Windows supported?

IF (!D.Flags AND 256) NE 0 THEN $
  Device, Get_Decomposed=decomposedState, Decomposed=0

; Fake data and Shade_Surf display.

data = Dist(40)
Surface, data, Shades=BytScl(data)

; Color decompositon back to entry value.

IF (!D.Flags AND 256) NE 0 THEN $
  Device, Decomposed=decomposedState
END
;-----

PRO FSC_Window_PostScript, event

; This event handler executes PostScript output.

; Error handling.

Catch, theError
IF theError NE 0 THEN BEGIN
  Catch, /Cancel
  IF !Error_State.Name EQ 'IDL_M_UPRO_UNDEF' THEN BEGIN
    ok = Dialog_Message(['Cannot find PSConfig. Please download
now', $
    'or add PSConfig directory to path.'])
    Widget_Control, event.top, Set_UValue=info, /No_Copy
    RETURN
  ENDIF ELSE BEGIN
    ok = Error_Message(Traceback=1, /Error)
    Widget_Control, event.top, Set_UValue=info, /No_Copy
    RETURN
  ENDELSE
ENDIF

; Get the info structure.

Widget_Control, event.top, Get_UValue=info, /No_Copy

; Allow user to configure the PostScript device.

```

```

info.psconfiguration->GUI,Cancel=cancelled, Group_Leader=event.top
keywords = info.psconfiguration->GetKeywords()
IF cancelled THEN BEGIN
    Widget_Control, event.top, Set_UValue=info, /No_Copy
    RETURN
ENDIF

```

; Display the graphics by executing the command.

```

thisDevice = !D.Name
Set_Plot, 'PS'
Device, _Extra=keywords
FSC_Window_Execute, info
Device, /Close_File
Set_Plot, thisDevice

```

; Return the info structure.

```

Widget_Control, event.top, Set_UValue=info, /No_Copy

```

```

END

```

```

;-----

```

```

PRO FSC_Window_Print, event

```

; This event handler executes the command in the Printer device.

; Set up the printer.

```

ok = Dialog_PrinterSetup()
IF NOT ok THEN RETURN

```

; Get info structure and printer orientation.

```

Widget_Control, event.top, Get_UValue=info, /No_Copy
Widget_Control, event.id, Get_UValue=orientation

```

; Load the program's color vectors.

```

TVLCT, info.r, info.g, info.b

```

; Save the current graphics device.

```

thisDevice = !D.Name

```

; Set up the printer. You may have to adjust the fudge factors
; to account for the printable area offset.

CASE orientation OF

'PORTRAIT': BEGIN

keywords = PSWindow(/Printer, Fudge=0.25)

Set_Plot, 'PRINTER', /Copy

Device, Portrait=1

ENDCASE

'LANDSCAPE': BEGIN

keywords = PSWindow(/Printer, /Landscape, Fudge=0.25)

Set_Plot, 'PRINTER', /Copy

Device, Landscape=1

ENDCASE

ENDCASE

; Display the graphics by executing the command.

Device, _Extra=keywords

FSC_Window_Execute, info

Device, /Close_Document

Set_Plot, thisDevice

; Restore the info structure.

Widget_Control, event.top, Set_UValue=info, /No_Copy

END

;-----

PRO FSC_Window_SaveAs, event

; Saves the current display window as output files.

; Get the info structure and the appropriate file extension.

Widget_Control, event.top, Get_UValue=info, /No_Copy

Widget_Control, event.id, Get_UValue=file_extension

; Base name for file output.

basename = 'fsc_window'

; Take a snapshot of the display window and write file.

```

WSet, info.wid
CASE file_extension OF
  'BMP' : image = TVREAD(Filename = basename, /BMP, Order=info.order)
  'GIF' : image = TVREAD(Filename = basename, /GIF, Order=info.order)
  'PICT' : image = TVREAD(Filename = basename, /PICT,
Order=info.order)
  'JPG' : image = TVREAD(Filename = basename, /JPEG,
Order=info.order)
  'TIF' : image = TVREAD(Filename = basename, /TIFF,
Order=info.order)
  'PNG' : image = TVREAD(Filename = basename, /PNG, Order=info.order)
ENDCASE

```

```

; Restore the info structure.

```

```

Widget_Control, event.top, Set_UValue=info, /No_Copy

```

```

END

```

```

;-----

```

```

PRO FSC_Window_Command__Define

```

```

; The definition of the command structure.

```

```

struct = { FSC_Window_Command, $
  command: "", $ ; The command to execute.
  p1: Ptr_New(), $ ; The first parameter.
  p2: Ptr_New(), $ ; The second parameter.
  p3: Ptr_New(), $ ; The third parameter.
  nparams: 0, $ ; The number of parameters.
  keywords: Ptr_New(), $ ; The command keywords.
  type: 0, $ ; =0 call_procedure =1 call_method
  user_event_pro: "" $ ; User defined event handling
routine for
  ; button or motion events
}

```

```

END

```

```

;-----

```

```

PRO FSC_Window_Quit, event

```

```

; This event handler destroys the program.

```

```

Widget_Control, event.top, /Destroy
END
;-----

```

```

PRO FSC_Window_Cleanup, tlb

```

```

; The cleanup routine for the program.

```

```

Widget_Control, tlb, Get_UValue=info, /No_Copy
IF N_Elements(info) EQ 0 THEN RETURN

```

```

; Free up the pointers and objects used in the program.

```

```

Ptr_Free, info.cmdStruct.p1
Ptr_Free, info.cmdStruct.p2
Ptr_Free, info.cmdStruct.p3
Ptr_Free, info.cmdStruct.keywords
Obj_Destroy, info.psconfiguration
END
;-----

```

```

PRO FSC_Window_TLB_Events, event

```

```

; The event handler for top-level base events.

```

```

Widget_Control, event.top, Get_UValue=info, /No_Copy

```

```

; What kind of event is this:

```

```

thisEvent = Tag_Names(event, /Structure_Name)

```

```

CASE thisEvent OF

```

```

    'WIDGET_BASE': BEGIN

```

```

        ; Resize the draw widget.

```

```

        IF StrUpCase(!Version.OS_Family) NE 'UNIX' THEN BEGIN

```

```

            Widget_Control, info.drawid, XSize=event.x, YSize=event.y
            info.xsize = event.x

```

```

        info.ysize = event.y

ENDIF ELSE BEGIN

        ; This code added to work-around UNIX
resize bug when
        ; TLB has a menu bar in IDL 5.2.

        Widget_Control, event.top, TLB_GET_Size=newsize
        xdiff = newsize[0] - info.tlboxsize
        ydiff = newsize[1] - info.tlbysize
        info.tlboxsize = event.x
        info.tlbysize = event.y
        info.xsize = info.xsize + xdiff
        info.ysize = info.ysize + ydiff
        Widget_Control, info.drawid, XSize=info.xsize,
YSize=info.ysize

ENDELSE

        ; Execute the command.

        FSC_Window_Execute, info

ENDCASE

'WIDGET_KBRD_FOCUS': BEGIN

        ; Keyboard focus events if color
protection is turned on.

        IF event.enter EQ 0 THEN BEGIN
            Widget_Control, event.top, Set_UValue=info, /No_Copy
            RETURN
        ENDIF

        ; Load colors and execute.

        TVLCT, info.r, info.g, info.b
        FSC_Window_Execute, info

ENDCASE

ELSE: call_procedure, info.cmdStruct.user_event_pro, event

ENDCASE
Widget_Control, event.top, Set_UValue=info, /No_Copy

```

END

;-----

PRO FSC_Window_Colors, event

; This event handler handles color events.

Widget_Control, event.top, Get_UValue=info, /No_Copy

; What kind of event is this?

thisEvent = Tag_Names(event, /Structure_Name)

CASE thisEvent OF

'WIDGET_BUTTON': BEGIN

; Call XColors to change colors.

TVLCT, info.r, info.g, info.b

XColors, Group_Leader=event.top, NColors=info.wcolors[0],

Bottom=info.wcolors[1], \$

Title=info.wtitle + ' Colors', NotifyID=[event.id, event.top]

ENDCASE

'XCOLORS_LOAD': BEGIN

; New color tables are loaded. Save them.

; Redisplay graphics on 24-bit displays.

Device, Get_Visual_Depth=theDepth

info.r = event.r

info.g = event.g

info.b = event.b

IF theDepth GT 8 THEN FSC_Window_Execute, info

ENDCASE

ENDCASE

Widget_Control, event.top, Set_UValue=info, /No_Copy

END

;-----


```

Catch, theError
IF theError NE 0 THEN BEGIN
  Catch, /Cancel
  ok = Error_Message(Traceback=1, /Error)
  RETURN
ENDIF

; Check for availability of GIF files.

thisVersion = Float(!Version.Release)
IF thisVersion LT 5.4 THEN haveGif = 1 ELSE haveGIF = 0

; Check keywords and define values if required.

IF N_Elements(method) EQ 0 THEN method = 0
IF N_Elements(tvorder) EQ 0 THEN tvorder = !Order
IF N_Elements(wxsize) EQ 0 THEN wxsize = 400
IF N_Elements(wysize) EQ 0 THEN wysize = 400
IF N_Elements(wxpos) EQ 0 THEN wxpos = -1
IF N_Elements(wypos) EQ 0 THEN wypos = -1
IF N_Elements(wtitle) EQ 0 THEN wtitle = "NOTITLE"
IF N_Elements(wbackground) EQ 0 THEN wbackground=!P.Background
Device, Get_Visual_Depth=theDepth
IF theDepth GT 8 AND N_Elements(needPS) EQ 0 THEN needPS = 1
IF theDepth GT 8 AND N_Elements(needPrint) EQ 0 THEN needPrint = 1
needPS = Keyword_Set(needPS)
needPrint = Keyword_Set(needPrint)

IF N_Elements(command) EQ 0 THEN BEGIN
  command = 'FSC_Window_Example'
  wtitle = 'Example Data'
  LoadCT, 4, /Silent
  wcolors = !D.Table_Size
  needPrint = 1
  needPS = 1
ENDIF

; Check for color handling.

IF Keyword_Set(wcolors) THEN BEGIN
  needColors = 1
  IF N_Elements(wcolors) EQ 1 THEN BEGIN
    IF wcolors[0] EQ 1 THEN wcolors = [!D.Table_Size, 0] ELSE wcolors
= [wcolors, 0]
  ENDIF
  wcolors[0] = (wcolors[1] + wcolors[0]) < (!D.Table_Size -
wcolors[1])
  IF wcolors[0] EQ 0 THEN $

```

```

    Message, 'Problem with COLORS keyword. Check calling sequence.',
/NoName
ENDIF ELSE BEGIN
    needColors = 0
    wcolors = 1
ENDELSE
weraseit = Keyword_Set(weraseit)

; Parse command and build a command structure.

IF Size(command, /TName) NE 'STRING' THEN $
    Message, 'First argument must be a string. Returning...', /NoName
cmdStruct = {FSC_WINDOW_COMMAND}
cmdStruct.command = command
cmdStruct.type = method
nparams = 0
IF N_Elements(p1) NE 0 THEN BEGIN
    nparams = nparams + 1
    cmdStruct.p1 = Ptr_New(p1)
ENDIF
IF N_Elements(p2) NE 0 THEN BEGIN
    nparams = nparams + 1
    cmdStruct.p2 = Ptr_New(p2)
ENDIF
IF N_Elements(p3) NE 0 THEN BEGIN
    nparams = nparams + 1
    cmdStruct.p3 = Ptr_New(p3)
ENDIF
cmdStruct.nparams = nparams
IF N_Elements(extra) NE 0 THEN cmdStruct.keywords = Ptr_New(extra) $
    ELSE cmdStruct.keywords = Ptr_New(/Allocate_Heap)

IF N_Elements(Button_events) NE 0 or N_Elements(Motion_Events) NE 0
THEN BEGIN
    IF N_Elements(user_event_pro) NE 0 THEN $
        cmdStruct.user_event_pro = user_event_pro $
    ELSE $
        Message, 'You have to set "user_event_pro" when Button or Motion
events are issued. Returning...', /NoName
    ENDIF

; Build the widgets.

tlb = Widget_Base(/TLB_Size_Events, Column=1, MBar=mbarID)
fileID = Widget_Button(mbarID, Value='File')

; Print button, if needed.

```

```

IF needPrint THEN BEGIN
    printID = Widget_Button(fileID, Value='Print',
Event_Pro='FSC_Window_Print', /Menu)
    dummy = Widget_Button(printID, Value='Landscape Orientation',
UVALUE='LANDSCAPE')
    dummy = Widget_Button(printID, Value='Portrait Orientation',
UVALUE='PORTRAIT')
ENDIF

; Save As menu.

saveID = Widget_Button(fileID, Value='Save As',
Event_Pro='FSC_Window_SaveAs', /Menu)
dummy = Widget_Button(saveID, Value='BMP File', UValue='BMP')
IF havegif THEN dummy = Widget_Button(saveID, Value='GIF File',
UValue='GIF')
dummy = Widget_Button(saveID, Value='PICT File', UValue='PICT')
dummy = Widget_Button(saveID, Value='PNG File', UValue='PNG')
dummy = Widget_Button(saveID, Value='JPEG File', UValue='JPG')
dummy = Widget_Button(saveID, Value='TIFF File', UValue='TIF')
IF needPS THEN dummy = Widget_Button(saveID, Value='PostScript File', $
Event_Pro='FSC_Window_PostScript')

; Colors button, if needed.

IF needcolors THEN dummy = Widget_Button(fileID, Value='Colors...', $
/Separator, Event_Pro='FSC_Window_Colors')

; Quit button.

quitID = Widget_Button(fileID, Value='Quit',
Event_Pro='FSC_Window_Quit', /Separator)

; Draw widget.

drawID = Widget_Draw(tlb, XSize=wxsize, YSize=wysize, $
    Button_Events=Button_Events, $
    Motion_Events=Motion_Events $
)

; Position the program on the display.

IF wxpos LT 0 OR wypos LT 0 THEN CenterTLB, tlb ELSE $
Widget_Control, tlb, XOffset=wxpos, YOffset=wypos
Widget_Control, tlb, /Realize
Widget_Control, drawID, Get_Value=wid

; Set unique title for the program.

```

```

IF wtitle EQ 'NOTITLE' THEN $
  wtitle = StrUpCase(command) + ' Window (' + StrTrim(wid,2) + ')'
ELSE $
  wtitle = wtitle + ' (' + StrTrim(wid,2) + ')'
Widget_Control, tlb, TLB_Set_Title=wtitle

; Set current graphics window.

WSet, wid

; Get some information for a window resize bug.

Widget_Control, tlb, TLB_Get_Size=tlbsizes

; Store the program's colors.

TVLCT, r, g, b, /Get

; Build the info structure.

info = { cmdStruct: cmdStruct, $
  xsize:wysize, $           ; X size of window.
  ysize:wysize, $           ; Y size of window.
  tlbxsize:tlbsizes[0], $    ; X size of TLB.
  tlbysize:tlbsizes[1], $    ; Y size of TLB.
  order:tvorder, $          ; The order of window
transfer.
  r: r, $                   ; The red color vector.
  g: g, $                   ; The green color vector.
  b: b, $                   ; The blue color vector.
  psconfiguration:Obj_New(), $ ; The window's PostScript
configuration.
  wid: wid, $               ; The window index number.
  drawid: drawid, $         ; The draw widget
identifier.
  wcolors:wcolors, $        ; The window's color
information.
  wtitle:wtitle, $          ; The window's title.
  wbackground: wbackground, $ ; The window's background
color.
  weraseit: weraseit $      ; The window's erase flag.
}

; If PostScript output is required, initialize the PSConfig object.

IF needPS THEN info.psconfiguration = Obj_New("FSC_PSConfig")

```

; If we are going to have problems executing the command, catch it here.

Catch, theError

IF theError NE 0 THEN BEGIN

 Catch, /Cancel

 Widget_Control, tlb, /Destroy

 ok = Error_Message('Problem executing command. Check spelling and syntax', /Error, /Traceback)

 RETURN

ENDIF

; Execute the command.

IF info.cmdStruct.type EQ 0 THEN \$

 IF N_Elements(*info.cmdStruct.keywords) EQ 0 THEN BEGIN

 CASE info.cmdStruct.nparams OF

 0: Call_Procedure, info.cmdStruct.command

 1: Call_Procedure, info.cmdStruct.command, *info.cmdStruct.p1

 2: Call_Procedure, info.cmdStruct.command, *info.cmdStruct.p1, *info.cmdStruct.p2

 3: Call_Procedure, info.cmdStruct.command, *info.cmdStruct.p1, *info.cmdStruct.p2, \$

 *info.cmdStruct.p3

 ENDCASE

 ENDIF ELSE BEGIN

 CASE info.cmdStruct.nparams OF

 0: Call_Procedure, info.cmdStruct.command,

 _Extra=*info.cmdStruct.keywords

 1: Call_Procedure, info.cmdStruct.command, *info.cmdStruct.p1,

 _Extra=*info.cmdStruct.keywords

 2: Call_Procedure, info.cmdStruct.command, *info.cmdStruct.p1, *info.cmdStruct.p2, \$

 _Extra=*info.cmdStruct.keywords

 3: Call_Procedure, info.cmdStruct.command, *info.cmdStruct.p1, *info.cmdStruct.p2, \$

 *info.cmdStruct.p3, _Extra=*info.cmdStruct.keywords

 ENDCASE

 ENDELSE \$

ELSE \$

 IF N_Elements(*info.cmdStruct.keywords) EQ 0 THEN BEGIN

 CASE info.cmdStruct.nparams OF

 0: Call_Method, info.cmdStruct.command

 1: Call_Method, info.cmdStruct.command, *info.cmdStruct.p1

 2: Call_Method, info.cmdStruct.command, *info.cmdStruct.p1, *info.cmdStruct.p2

 3: Call_Method, info.cmdStruct.command, *info.cmdStruct.p1, *info.cmdStruct.p2, \$

```

        *info.cmdStruct.p3
    ENDCASE
ENDIF ELSE BEGIN
    CASE info.cmdStruct.nparams OF
        0: Call_Method, info.cmdStruct.command,
        _Extra=*info.cmdStruct.keywords
        1: Call_Method, info.cmdStruct.command, *info.cmdStruct.p1,
        _Extra=*info.cmdStruct.keywords
        2: Call_Method, info.cmdStruct.command, *info.cmdStruct.p1,
        *info.cmdStruct.p2, $
        _Extra=*info.cmdStruct.keywords
        3: Call_Method, info.cmdStruct.command, *info.cmdStruct.p1,
        *info.cmdStruct.p2, $
        *info.cmdStruct.p3, _Extra=*info.cmdStruct.keywords
    ENDCASE
ENDELSE

; Save the info structure and turn keyboard focus events on.

Widget_Control, tlb, Set_UValue=info, /No_Copy,
KBRD_Focus_Events=KBRD_Focus_Events

; Start er up!

XManager, 'fsc_window', tlb, /No_Block,
Event_Handler='FSC_Window_TLB_Events', $
Group_Leader=group_leader, Cleanup='FSC_Window_Cleanup'

END
;-----

```
