Subject: Re: How do I create a plot which looks something like a Matrix Plot in IDL ?
Posted by James[3] on Sun, 03 Jan 2016 09:40:57 GMT
View Forum Message <> Reply to Message

On Sunday, January 3, 2016 at 2:40:03 AM UTC+1, Jim P wrote:
> On Saturday, January 2, 2016 at 2:30:29 PM UTC-7, James wrote:
>> On Saturday, January 2, 2016 at 9:51:16 PM UTC+1, Jim P wrote:
>>> On Saturday, January 2, 2016 at 10:31:50 AM UTC-7, James wrote:
>>>> I have two dataset scenarios. Each scenario is an output from a model and each scenario contains an array of values in the x (horizontal) axis and 10 arrays of values in the y (vertical) axis. I am trying to create a plot which looks like this http://www.mathworks.com/matlabcentral/answers/uploaded_file s/42412/3.png
>>>> I have been looking for a plot function in ENVI IDL which can help make these plots but so far I did not find any. Can anybody help? Even better if someone can suggest a better graphical/visualization/plot option to display these two multivariate datasets so that the differences are conspicuous enough. With line graphs the visualization is not clear enough because the same values overlap on top of each other.
>>>>
>>>> Following are my scenarios.
>>>>
>>>> SCENARIO A
>>>>
>>>> x= [0, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]
>>>> y1= [0, 0, 0.02, 0.01, 0, 0, 0, 0, 0, 0, 0, 0]
>>>> y2= [0.01, 0, 0.05, 0.1, 0.19, 0.6, 0.87, 1, 1, 1, 1, 1]
>>>> y3= [0.02, 0.05, 0.2, 0.69, 0.99, 1, 1, 1, 1, 1, 1, 1]
>>>> y4= [0.02, 0.12, 0.25, 0.97, 1, 1, 1, 1, 1, 1, 1, 1]
>>>> y5= [0, 0.12, 0.68, 1, 1, 1, 1, 1, 1, 1, 1, 1]
>>>> y6= [0, 0.2, 0.84, 1, 1, 1, 1, 1, 1, 1, 1, 1]
>>>> y7= [0.01, 0.49, 0.97, 1, 1, 1, 1, 1, 1, 1, 1, 1]
>>>> y8= [0.01, 0.51, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
>>>> y9= [0.01, 0.82, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
>>>> y10= [0, 0.84, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
>>>>
>>>> SCENARIO B
>>>>
>>>> x= [0, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]
>>>> y1= [0.01, 0.03, 0.01, 0, 0.01, 0, 0, 0, 0, 0, 0, 0]
>>>> y2= [0.01, 0.07, 0.04, 0.13, 0.23, 0.5, 0.92, 1, 1, 1, 1, 1]
>>>> y3= [0.01, 0.03, 0.2, 0.61, 0.99, 1, 1, 1, 1, 1, 1, 1]
>>>> y4= [0.02, 0.06, 0.4, 0.99, 1, 1, 1, 1, 1, 1, 1, 1]
>>>> y5= [0, 0.24, 0.61, 1, 1, 1, 1, 1, 1, 1, 1, 1]
>>>> y6= [0, 0.26, 0.88, 1, 1, 1, 1, 1, 1, 1, 1, 1]
>>>> y7= [0, 0.51, 0.99, 1, 1, 1, 1, 1, 1, 1, 1, 1]
>>>> y8= [0.02, 0.64, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
>>>> y9= [0.02, 0.87, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
>>>> y10= [0.01, 0.94, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
>>>

>>> The simplest way may be to use a widget_table with cell colors, then take a screen capture.
>>>
>>> Jim P.
>> ** ****************
>>
>> Dear Jim,
>>
>> Thanks a lot for your helpful comments. I have never used WIDGET_TABLE function before since I am quite new to IDL Programming. The idea seems interesting and I want to try it. I am right now looking at the help page http://www.exelisvis.com/docs/WIDGET_TABLE.html
>> Do you know whether this function allows automatic addition of cell colours based on the range of values present or do I need to manually assign cell colours to the values? Additionally, if I assign colours to the cells which hold the values, will the values be still visible or I can replace the values just with colours? Apologies if I am asking some basic questions which may be otherwise common knowledge among the IDL Programmers' community. It would be a great help if I can find an example. I will search the web for it. If you have any other suggestions they are welcome.
>>
>> James
>
> Below is an example I wrote a number of years ago, with additional input from Dr. Mike Galloy. On the left is an image and on the right is a tabular form of the image with the cells colored according to the selected color table, using the same color table as the pixels in the image.
>
> You would be responsible for setting the color of each cell.
>
> "Showing" and "hiding" the text value in a cell is simply a matter of setting the foreground color to be the same as the background color.
>
> There are lots of comments that, along with the online help, should get you where you want to be.
>

```
> ;+
> ; This procedure updates the image window display based
> ; on the current contents of the table widget, the
> ; selected cells (if any) and the current color table.
> ;
> ; @Param
> ;   TLB {in}{required}{type=long}
> ;       Set this parameter to the ID of the top-level base.
> ;
> ; @Hidden
> ;
> ; @Author
> ;   JLP, RSI Global Services
> ;
> ; @History
> ;   March 29, 2005 - Initial version
> ;-
>  Pro ImageTable_62_UpdateImage, TLB
>  Compile_Opt StrictArr
>  On_Error, 2
>  ;
>  ; Get the raw data values from the table.
>  ;
>  Table = Widget_Info(TLB, Find_by_UName='ValueTable')
>  Widget_Control, Table, Get_Value = Values
>  ;
>  ; Get the color table number from the combobox.
>  ;
>  CTCombobox = Widget_Info(TLB, Find_by_UName = 'ColorTableCombobox')
>  Current = Widget_Info(CTCombobox, /Combobox_GetText)
>  Widget_Control, CTCombobox, Get_Value = AllValues
>  CTIndex = (Where(AllValues eq Current))[0]
>  ;
>  ; Save the Direct Graphics state.
>  ;
>  WSave = !d.window
>  Device, Get_Decomposed = WasDecomposed
>  TVLCT, R, G, B, /Get
>  ;
>  ; Draw the table values to the image window using
>  ; the current color table.
>  ;
>  Draw = Widget_Info(TLB, Find_by_UName = 'ImageDraw')
>  Widget_Control, Draw, Get_Value = DrawID
>  WSet, DrawID
>  LoadCT, CTIndex, /Silent
>  Device, Decomposed = 0
>  ;
>  ; Remember that the table values are flipped in Y
>  ; relative to our image so they orient the same way
>  ; on the screen.
>  ;
>  TVScl, Reverse(Values, 2)
>  ;
>  ; Read the image buffer back from the window.
>  ;
>  Device, Decomposed = 1
>  Image = TVRd(True = 1)
>  ;
>  ; Restore the Direct Graphics state.
>  ;
>  Device, Decomposed = WasDecomposed
>  If (WSave ne -1) then Begin
>     Device, Window_State = WState
>     If (WState[WSave]) then Begin
>         WSet, WSave
>     EndIf
>  EndIf
>  TVLCT, R, G, B
>  ;
>  ; Save the image data for the purpose of reapplying the
>  ; selection box.
>  ;
>  Widget_Control, TLB, Set_UValue = Image
>  ;
>  ; Flip the image data about Y back into the orientation of the
>  ; table.
>  ;
>  TableImage = Reverse(Temporary(Image), 3)
>  ;
>  ; The background color of each cell corresponds to each pixel
>  ; value.  We use "Update = 0" to prevent excessive flashing.
>  ;
>  Widget_Control, Table, $
>     Background_Color = TableImage, Update = 0
>  ;
>  ; If we're "hiding" the text, this just means we draw the
>  ; table values using the same color as the background.
>  ;
>  HideShow = Widget_Info(TLB, Find_by_UName = 'HideShowCombobox')
>  If (Widget_Info(HideShow, /Combobox_GetText) eq 'Show') then Begin
>  ;
>  ; If we're showing table values, render the text in either black
>  ; or white, depending on the better contrast with the background
>  ; color in the cell.
>  ;
>     ImageTable_62_EnhanceText, TableImage
>  EndIf
>  Widget_Control, Table, $
>     Foreground_Color = TableImage, Update = 0
>  ;
>  ; Update the base now with the accumulated changes.
>  ;
>  Widget_Control, TLB, /Update
>  If (~Widget_Info(TLB, /Map)) then Begin
>     Widget_Control, TLB, Map = 1
>  EndIf
>  End
>
>
>  ;+
>  ; This procedure converts an input array of table cell (image
>  ; pixel) colors to an array of black and/or white colors to
>  ; be used as the text color of each cell.  The color black
>  ; or white is chosen to increase contrast with the background
>  ; color in each cell.
>  ;
>  ; @Param
>  ;   RGB {inout}{required}{type=BYTARR(3, N, M)}
>  ;       Set this parameter to the RGB colors of the pixels
>  ;       displayed in the image using the current color table.
>  ;       On output, the array will contain the color, either
>  ;       white or black, to be used for the foreground (text)
>  ;       color for the corresponding table cells.
>  ;
>  ; @Hidden
>  ;
>  ; @Author
>  ;   JLP, RSI Global Services
>  ;
>  ; @History
>  ;   March 29, 2005 - Initial version
>  ;-
>  Pro ImageTable_62_EnhanceText, RGB
>  Compile_Opt StrictArr
>  On_Error, 2
>  ;
>  ; Convert the individual pixel colors from RGB space to
>  ; hue, lightness and saturation space.
>  ;
>  Color_Convert, Reform(RGB[0, *, *]), Reform(RGB[1, *, *]), $
>     Reform(RGB[2, *, *]), H, L, S, /RGB_HLS
>  ;
>  ; We say that a pixel is "dark" if its lightness is less
>  ; than 50%.
>  ;
>  Dark = Where(L lt .50, NDark)
>  ;
>  ; Light pixels will use black text.  This is out initial
>  ; default.
>  ;
>  RGB[*] = 0b
>  ImageSize = Size(RGB, /Dimensions)
>  If (NDark ne 0) then Begin
>  ;
>  ; Dark pixels will have white text.  We use reform here
>  ; to aid in addressing our 2-D image space with the 1-dimensional
>  ; vectors returned by Color_Convert.  We use Overwrite so
>  ; we don't make extra copies of the data.
>  ;
>     RGB = Reform(RGB, 3, N_elements(RGB)/3, /Overwrite)
>     RGB[*, Dark] = 255b
>     RGB = Reform(RGB, 3, ImageSize[1], ImageSize[2], /Overwrite)
>  EndIf
>  End
>
>
>  ;+
>  ; This procedure manages all events from the application's
>  ; widgets.
>  ;
>  ; @Param
>  ;   Event {in}{required}{type=widget event structure}
>  ;       Set this parameter to the event structure to be
>  ;       acted upon.
>  ;
>  ; @Hidden
>  ;
>  ; @Author
>  ;   JLP, RSI Global Services
>  ;
>  ; @History
>  ;   March 29, 2005 - Initial version
>  ;-
>  Pro ImageTable_62_Event, Event
>  Compile_Opt StrictArr
>  On_Error, 2
>  ;
>  ; What type of event is it?
>  ;
>  EventType = Tag_Names(Event, /Structure_Name)
>  If (EventType eq 'WIDGET_BASE') then Begin
>  ;
>  ; This is a base resize event.  We adjust the size of the
>  ; table accordingly (and leave all the other widgets alone.)
>  ;
>     TLBGeom = Widget_Info(Event.Top, /Geometry)
>  ;
>  ; The base on the left hand side containing the image, comboboxes
>  ; and spinner, remains fixed in size.
>  ;
>     Left = Widget_Info(Event.Top, Find_by_UName = 'LeftBase')
>     LeftGeom = Widget_Info(Left, /Geometry)
>  ;
>  ; The table widget occupies "everything else".
>  ;
>     NewX = Event.X - 2*TLBGeom.XPad - LeftGeom.Scr_XSize > 10
>     NewY = Event.Y - 2*TLBGeom.YPad > 10
>     Table = Widget_Info(Event.Top, Find_by_UName = 'ValueTable')
>  ;
>  ; Update the table size and return.
>  ;
>     Widget_Control, Table, Scr_XSize = NewX, Scr_YSize = NewY
>     Return
>  EndIf
>  ;
>  ; All other events are managed according to the UVALUE of the
>  ; widget that produced the event.
>  ;
>  Widget_Control, Event.ID, Get_UValue = BranchCode
>  Case BranchCode of
>     'ValueVisibility' : Begin
>  ;
>  ; Hide or show the values in the table.
>  ;
>        Widget_Control, Event.Top, Get_UValue = Image
>        Table = Widget_Info(Event.Top, Find_by_UName = 'ValueTable')
>        Case Widget_Info(Event.ID, /Combobox_GetText) of
>           'Show' : Begin
>  ;
>  ; If we're showing the text in the table, make sure the text color
>  ; contrasts well with the cell color.
>  ;
>              ImageTable_62_EnhanceText, Image
>  ;
>  ; Remember that the table values are "upside down" in Y to
>  ; correspond to our image orientation, so we need to flip
>  ; the data.
>  ;
>              Widget_Control, Table, $
>                 Foreground_Color = Reverse(Temporary(Image), 3)
>              End
>           'Hide' : Begin
>  ;
>  ; If we're hiding the text in the table, then we set the text
>  ; color to be the same as the image color at that pixel.  Again,
>  ; we need to flip the data to match the image orientation.
>  ;
>              Widget_Control, Table, $
>                 Foreground_Color = Reverse(Temporary(Image), 3)
>              End
>        EndCase
>        End
>     'ColorTable' : Begin
>  ;
>  ; A change in color table means we must update the image as well
>  ; as the cell colors in the table.
> ;
>         ImageTable_62_UpdateImage, Event.Top
>         End
>     'Table' : Begin
>         EventType = Tag_Names(Event, /Structure_Name)
>         If (EventType eq 'WIDGET_CONTEXT') then Begin
> ;
> ; If we have a right click, this is a context menu event
> ; request, so show it at the position of the clicked cell.
> ; (Note that we have previously received at least one and
> ; maybe two WIDGET_TABLE_CELL_SEL events as a result of
> ; the right-click selection, if the table was editable.)
> ;
>             ContextMenu = Widget_Info(Event.ID, $
>                 Find_by_UName = 'TableContextMenu')
>             Widget_DisplayContextMenu, Event.ID, $
>                 Event.X, Event.Y, ContextMenu
>             Return
>         EndIf
> ;
> ; The user has selected cells in the table.  First,
> ; update the image window.
> ;
>         Widget_Control, Event.Top, Get_UValue = Image
>         Draw = Widget_Info(Event.Top, Find_by_UName = 'ImageDraw')
>         Widget_Control, Draw, Get_Value = DrawID
> ;
> ; Save the Direct Graphics environment.
> ;
>         WSave = !d.window
>         Device, Get_Decomposed = WasDecomposed
> ;
> ; Display the image to the draw widget.  This will erase any
> ; previously overplotted line.  A cleverer solution might use
> ; a pixmap instead.
> ;
>         WSet, DrawID
>         Device, Decomposed = 1
>         TV, Image, True = 1
> ;
> ; Overplot the bounds of the region selected in the table
> ; if it's more than 1 cell.
> ;
>         Selected = Widget_Info(Event.ID, /Table_Select)
>         MinX = Min(Selected[0, *], Max = MaxX)
>         MinY = Min(Selected[1, *], Max = MaxY)
>         dX = MaxX - MinX + 1
>         dY = MaxY - MinY + 1
>         If ((dX gt 0) && (dY gt 0)) then Begin
>  ;
>  ; We're going to plot the bounding box in the image in green.
>  ; We need to remember that the cell numbers in the table
>  ; are flipped top to bottom with respect to the image.
>  ;
>            PlotS, MinX + [0, 1, 1, 0, 0]*dX, $
>               (Size(Image, /Dimensions))[2] - 1 - $
>               (MinY + [0, 0, 1, 1, 0]*dY), $
>               /Device, Color = '00ff00'x
>         EndIf
>  ;
>  ; Restore the Direct Graphics environment.
>  ;
>         If (WSave ne -1) then Begin
>            Device, Window_State = WState
>            If (WState[WSave]) then Begin
>               WSet, WSave
>            EndIf
>         EndIf
>         Device, Decomposed = WasDecomposed
>         End
>      'PixelSize' : Begin
>  ;
>  ; Change the number of pixels square each cell in the table
>  ; should occupy according to the value in the spinner.  This number
>  ; actually includes the dividers between cells.
>  ;
>         Table = Widget_Info(Event.Top, Find_by_UName = 'ValueTable')
>  ;
>  ; There are no bounds on the spinner, so we need to manually clamp
>  ; it to a reasonable range.
>  ;
>         Value = Long(Event.Value)
>         Value >= 2
>         Value <= 256
>  ;
>  ; In case we needed to clamp the value, update the spinner with
>  ; the current value.
>  ;
>         Widget_Control, Event.ID, Set_Value = Value, Update = 0
>  ;
>  ; Adjust the cell dimensions in the table, then update
>  ; everything on the interface at one time.
>  ;
>         Widget_Control, Table, Row_Heights = Value, $
>            Column_Widths = Value, Update = 0
>         Widget_Control, Event.Top, /Update
>         End
>     'Context_SelectedStatistics' : Begin
>  ;
>  ; This is an event from the table's context menu.
>  ; Calculate statistics on the selected cells.
>  ;
>         ParentContextBase = Widget_Info(Event.ID, /Parent)
>         Table = Widget_Info(Event.Top, Find_by_UName = 'ValueTable')
>         Selected = Widget_Info(Table, /Table_Select)
>         Widget_Control, Table, Get_Value = TableValues
>         If (N_elements(Selected)/2 gt 1) then Begin
>            TableSize = Size(TableValues, /Dimensions)
>            Selected = Reform(Selected[0, *] + Selected[1, *]*TableSize[0])
>            Stats = Moment(TableValues[Selected], MDev = MDev, SDev = SDev)
>            MinValue = Min(TableValues[Selected], Max = MaxValue)
>            Stats = StrTrim([MinValue, MaxValue, Stats, MDev, SDev], 2)
>            StatsLabels = ['Minimum', 'Maximum', 'Mean', 'Variance', 'Skewness', $
>               'Kurtosis', 'Mean Absolute Deviation', $
>               'Standard Deviation']
>            StatsLabel = StatsLabels + ' : ' + Stats
>         EndIf Else Begin
>            StatsLabel = 'Pixel value = ' + $
>               StrTrim(TableValues[Selected[0, 0], Selected[1, 0]], 2)
>         EndElse
>         v = Dialog_Message(StatsLabel, /Information, $
>            Dialog_Parent = Event.Top, $
>            Title = 'ImageTable 6.2 ROI Statistics')
>         End
>     'Context_SelectedArea' : Begin
>  ;
>  ; This is an event from the table's context menu.
>  ; Calculate the area of the selected cells
>  ;
>         ParentContextBase = Widget_Info(Event.ID, /Parent)
>         Table = Widget_Info(Event.Top, Find_by_UName = 'ValueTable')
>         Selected = Widget_Info(Table, /Table_Select)
>         NPixels = Long(N_elements(Selected)/2)
>         AreaLabel = 'Area = ' + $
>            StrTrim(NPixels, 2) + $
>            ' pixel' + (NPixels gt 1 ? 's' : '')
>         v = Dialog_Message(AreaLabel, /Information, $
>            Dialog_Parent = Event.Top, $
>            Title = 'ImageTable 6.2 ROI Area')
>         End
>     Else :
>  EndCase
>  End
>
```

```
>
> ;+
> ; This procedure creates a table widget whose cell values are
> ; displayed in colors according to a color look-up table,
> ; much like an image.
> ;
> ; @Param
> ;   Val {in}{optional}{type=integer array dimensions n by m}
> ;       Set this parameter to a 2-dimensional array of integral
> ;       values between the values of -99 and 999.  (This limitation
> ;       is related to the format statement used to display the
> ;       cell values; feel free to modify the code according to
> ;       your data's needs.) The default value is a 64-by-64 shifted
> ;       DIST image.
> ;
> ; @Examples <pre>
> ;   IDL> ImageTable_62 </pre>
> ;
> ; @Categories
> ;   Widget_Table, 6.2
> ;
> ; @Author
> ;   Jim Pendleton & MG, RSI Global Services
> ;
> ; @History
> ;   March 31, 2005 - Initial version
> ;
> ; @File_Comments
> ;   This procedure highlights some new features of WIDGET_TABLE
> ;   in IDL 6.2, in particular the ability to set foreground
> ;   and background colors in individual cells, and to display
> ;   a context menu within a table. <br>
> ;
> ;-
> Pro ImageTable_62, Val
> Compile_Opt StrictArr
> On_Error, 2
> ;
> ; Create a copy of the input data or create the default "bulls-eye"
> ; image.
> ;
> iVal = N_Elements(Val) eq 0 ? Fix(Shift(Dist(512), 256, 256)) : Val
> ImageSize = Size(iVal, /Dimensions)
> ;
> ; Create our widget tree.  We'll have some controls on the left
> ; and a table widget on the right.
> ;
> TLB = Widget_Base(/Row, /TLB_Size_Events, UName = 'Top', $
>     Title = 'ImageTable 6.2')
>  Left = Widget_Base(TLB, /Column, UName = 'LeftBase')
>  D = Widget_Draw(Left, XSize = ImageSize[0], YSize = ImageSize[1], $
>     UName = 'ImageDraw')
>  ;
>  ; By default, we use the "Hardcandy" color table.  It shows good
>  ; contrast in the upper corner that is initiall displayed in the
>  ; table.
>  ;
>  CTBase = Widget_Base(Left, /Row, /Align_Left)
>  CTLabel = Widget_Label(CTBase, Value = 'Color Table : ')
>  LoadCT, /Silent, Get_Names = CTNames
>  ColorTable = 28
>  CTCombobox = Widget_Combobox(CTBase, Value = CTNames,$
>      UName = 'ColorTableCombobox', UValue = 'ColorTable')
>  Widget_Control, CTCombobox, Set_Combobox_Select = ColorTable
>  VisibilityBase = Widget_Base(Left, /Row, /Align_Left)
>  VisibilityLabel = Widget_Label(VisibilityBase, $
>     Value = 'Table Values : ')
>  VisibilityCombobox = Widget_Combobox(VisibilityBase, $
>     Value = ['Show', 'Hide'], $
>     UValue = 'ValueVisibility', $
>     UName = 'HideShowCombobox')
>  ;
>  ; We'll initially display each cell as an 18x18 square.  The
>  ; spinner will allow the user to adjust this.  See the
>  ; idl62/lib/itools/ui_widgets directory for the source to
>  ; cw_itupdownfield.pro, an undocumented, but highly useful
>  ; compound widget.
>  ;
>  PixelSize = 18
>  Spinner = CW_itUpDownField(Left, Increment = 1, $
>     Label = 'Pixel Size : ', $
>     Value = pixelsize, $
>     UName = 'PixelSizeSpinner', $
>     UValue = 'PixelSize')
>  ;
>  ; Create a table with our data values.  The default orientation
>  ; places the cell [0, 0] at the upper left.  But we want it
>  ; to be oriented the same as our image, with [0, 0] in the
>  ; lower left..  So we need to flip the data in Y.
>  ;
>  Table = Widget_Table(TLB, Value = Reverse(IVal, 2), $
>     X_Scroll = 25,$
>     Y_Scroll = 25, $
>     Row_Heights = PixelSize, Column_Widths = PixelSize, $
>     Format = '(i3)', $
>     UName = 'ValueTable', $
>     UValue = 'Table', $
>  ;
>  ; We choose a font appropriate to Windows or Linux.  We want it
>  ; to be small so our cell sizes can be small, too.
>  ;
>     Font = !version.os_family eq 'Windows' ? $
>     'Helvetica*8' : 'timr08', $
>     Alignment = 1, $
>  ;
>  ; Label the rows to show that we flipped the image in Y.
>  ; Row 0 is at the bottom.
>  ;
>     Row_Labels = StrTrim(Reverse(Lindgen(ImageSize[1])), 2), $
>     /All_Events, $
>  ;
>  ; Also capture context menu events (i.e., right-clicks.)
>  ;
>     /Context_Events, $
>     /Disjoint)
>  ;
>  ; Create a context menu for the table.
>  ;
>  ContextBase = Widget_Base(Table, /Context_Menu, $
>     UName = 'TableContextMenu')
>  ContextStatisticsButton = Widget_Button(ContextBase, $
>     Value = 'Show Selected Statistics', $
>     UValue = 'Context_SelectedStatistics')
>  ContextStatisticsArea = Widget_Button(ContextBase, $
>     Value = 'Show Selected Area', $
>     UValue = 'Context_SelectedArea')
>  ;
>  ; We initially hide the TLB until after we update the colors
>  ; in the table the first time.
>  ;
>  Widget_Control, TLB, Map = 0
>  Geom = Widget_Info(TLB, /Geometry)
>  ScreenSize = Get_Screen_Size()
>  Widget_Control, TLB, $
>     TLB_Set_XOffset = (ScreenSize[0] - Geom.Scr_XSize)/2., $
>     TLB_Set_YOffset = (ScreenSize[1] - Geom.Scr_YSize)/2.
>  Widget_Control, TLB, /Realize
>  ;
>  ; Update the colors in the cells and start the event handler.
>  ;
>  ImageTable_62_UpdateImage, TLB
>  XManager, 'ImageTable_62', TLB, /No_Block
>  End
```

Dear Jim,

Thank you very much for providing the code. If I finally use parts of the code for my research work, I will credit your name and Dr. Mike Galloy's. But before I implement it for my example, I have a question. If I have to set the color of each cell, do you think it is practical as I have more than 30 dataset scenarios. If there is an option of setting the cell colours automatically then there would be consistency of colour usage across all the 30 scenarios. By the way, I don't see any image on the left and neither do I see is a tabular form of the image on the right?

-James
