Subject: Re: A third-party example
Posted by davidf on Mon, 09 Apr 2001 14:48:03 GMT
View Forum Message <> Reply to Message

Ben Tupper (btupper@bigelow.org) writes:

> Liam suggested that I post the GUI object I made with Martin's
> GUIobject. I have pasted it below.

I had just a little bit of trouble getting this to work.
First of all, I had an older version of MGS_Container
on my path, that I had to get rid of. Then, I found
a typo on line 850 of HBB_TwoLists__Define.pro:

```
  S_label = Self.LabelsNames[1]
```

Should be:

```
   S_label = Self.LabelNames[1]
```

Now it works, but the two lists have taken over my computer!
You must have some BIG default lists. :-)

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

Subject: Re: A third-party example
Posted by btt on Mon, 09 Apr 2001 15:26:48 GMT
View Forum Message <> Reply to Message

Hi,

Thanks for the correction. What a great debut (thud!)

> Now it works, but the two lists have taken over my computer!
> You must have some BIG default lists. :-)

Oh geez! That was a fix I added at the last minute, nuts! I explicitly
set the size of the second list from the geometry of the first, as

```
 FirstList = Widget_List(BlahBlahBlah)
 Geo = Widget_info(FirstList, /Geometry)
 SecondList = Widget_LIST(blahblahblah, $
  xSize = Geo.Scr_xSize, ySize = Geo.Scr_ySize) <---- Bad Idea
```

This step was an attempt to endrun around a sizing difference between Mac and Unix widget sizing. Endrun! Hmph, I ran right off the field!

The updated code follows. On this Mac the two lists occupy the same screen space. I would be interested in knowing if that is not true on other platforms.

Thanks again,

Ben

P.S. I am reading 'Arctic Dreams' by Barry Lopez... Wow!

```
;------STARTHERE
;+
; NAME:
; HBB_TwoList
;
; PURPOSE:
; The purpose of this widget/object to provide the user an interactive
means of
; of adjusting the contents of a list. Items from a 'primary' list may
be copied to a
;'secondary' list.  Items in the secondary list may be removed or 'moved
up' or
;'moved down' relative to each other.
;
; REQUIREMENTS:
; INHERITS MGS_GUIObject
;
; CATEGORY:
; Widgets.
;
; CALLING SEQUENCE:
; objref = OBJ_NEW('HBB_TwoList')
;
; ARGUMENTS:
; None required
;
; INITIALIZATION KEYWORDS: (see MGS_GUIobject)
; P_LIST (Get/Set) An vector of values for the primary list.
;  The datatype is maintained as provided, but the List_Widget
;  requires that the values be string.  So most numeric types
```

```
;  are acceptable when converted to STRING.  The items in the
;  the secondary list are selected from this list. If not provided,
;  a dummy list (list of dummies?) is provided automatically.
; S_LIST (Get/Set) A vector of items for the secondary list. Default = ['']
; P_LABEL (Get/Set)  A label for the primary list.  Default = ''
; S_LABEL (Get/Set) A label for the secondary list. Default = ''
; MAX_LENGTH (Get/Set) The maximum number of rows shown in the list,
Default = 30
; MIN_LENGTH (Get/Set) The minimum number of rows shown in the list ,
Default = 10
; UPDOWN (Get/Set) Set this keyword to permit up/down shuffling of items
in
;  the secondary list using the MoveUp/MoveDown buttons.
; MULTIPLE (Get/Set) Set this keyword to permit multiple-item selection
;  in either list.
;  NO_DUPLICATES (Get/Set) Set this keyword to prevent duplicate
;  items from the primary list from appearing in the secondary list.
;
;
; EXAMPLE:
; o = Obj_New('HBB_TWOLIST',/UPDOWN)
; o->GUI
;  ;fiddle with the lists... over items left/right/up/down.
;  ;close the dialog
; o->GUI
;  ;note your earlier chioces are preserved
; Obj_Destroy, o
;  ;cleanup
;
; MODIFICATION HISTORY:
; written 3 APR 2001 Ben Tupper
; Bigelow Laboratory for Ocean Science
; btupper@bigelow.org
;  Based upon PRC_DualList widget.
;
; 4 APR 2001 Fixed bug when moving the top item DOWN. BT
;  Changed the keyword SHUFFLE to UPDOWN for clarity.
; 9 APR 2001 Changed 'Add' to 'Copy' and placed UpDown buttons under the
;  secondary list.   Forced buttons to same size. BT
;
;
;-



;------
; CopyItem
;------
FUNCTION HBB_TwoList::CopyItem, event
```

```
Catch, Error
If Error NE 0 then Begin
 Self->ErrorMessage
 Catch, /Cancel
 Return,0
 EndIF

p_select = widget_info(self.listid[0],/List_select)
If p_select[0] NE -1 then begin

  ;if duplicates are not permitted, then
  ;check for duplication otherwise skip this step
 If Self.No_Duplicates EQ 1 Then Begin
  A = where(*Self.list[1] EQ (*self.list[0])[p_select], count)
  If Count NE 0 then Return, 0
  EndIf


 if n_elements(*self.List[1]) GT 1 Then Begin
  *self.List[1] = [*self.list[1] , (*self.List[0])[p_select] ]
  EndIf Else begin
   if (*self.list[1])[0] EQ '' Then $
   *self.List[1] = (*self.List[0])[p_select] Else $
   *self.List[1] = [*self.list[1], (*self.List[0])[p_select] ]
  EndELSE ;the secondary list is empty

 Widget_control, Self.ListID[1], Set_Value = String(*Self.List[1])

 EndIF ; the p_select[0] ne -1

Return,0
END ;CopyItem event

;------
; RemoveItem
;------
FUNCTION HBB_TwoList::RemoveItem, event
Catch, Error
If Error NE 0 then Begin
 Self->ErrorMessage
 Catch, /Cancel
 Return,0
 EndIF


 ;if the secondary list is empty, then return immediately
If n_elements(*(Self.List)[1]) EQ 0 then Return,0
```

```
s_select = widget_info(self.listID[1], /List_select)

If s_select[0] NE -1 Then Begin
 OldList = *(Self.List)[1]
 If n_elements(OldList) GT 1 Then Begin
  index = replicate(1L, n_elements(OldList))
  Index[s_select] = 0
  A = where(index EQ 1, count)
  if Count gt 0 then *(Self.List)[1] = oldList[a] Else $
   *(self.List)[1] = ''
  EndIf Else begin
   *(self.List)[1] = ''
  EndElse ; OldList only had one item to remove
 Widget_control, Self.ListID[1], Set_Value = STRING(*(Self.List)[1])
 EndIf ;s_select[0] ne -1

Return, 0
END ;RemoveItem event

;------
; MoveUp
;------
FUNCTION HBB_TwoList::MoveUp, event
Catch, Error
If Error NE 0 then Begin
 Self->ErrorMessage
 Catch, /Cancel
 Return,0
 EndIF


s = widget_info(Self.ListID[1], /List_select)
If s[0] EQ -1 Then Return,0
 ;return if the list is 'empty'
if (*(Self.List)[1])[s[0]] EQ '' Then Return,0

nsel = n_elements(s)
nList = n_elements(*(Self.List)[1])

 ;can't allow a higher index than elements in s_list
if s[nsel-1] GT (nlist-1) Then Return,0

if s[0] EQ 0 Then Begin ;move the top to the bottom
 *(self.list)[1] = Shift(*(self.list)[1], -1)
 EndIf Else begin
 Index = Lindgen(Nlist)
 Index[(s[0]-1L) : (s[nSel-1])] = SHIFT(Index[(s[0]-1L) : (s[nSel-1])] , -1)
 (*(Self.List)[1])= (*(Self.List)[1])[Index]
```

```
 EndElse

Widget_control, Self.ListID[1], Set_Value = STRING( *(self.List)[1] )

Return, 0
END ;MoveUp event

;------
; MoveDown
;------
FUNCTION HBB_TwoList::MoveDown, event
Catch, Error
If Error NE 0 then Begin
 Self->ErrorMessage
 Catch, /Cancel
 Return,0
 EndIF

s = widget_info(Self.ListID[1], /List_select)
If s[0] EQ -1 Then Return,0
 ;return if the list is 'empty'
if (*(Self.List)[1])[s[0]] EQ '' Then Return,0

nsel = n_elements(s)
nList = n_elements(*(Self.List)[1])

 ;can't allow a higher index than elements in s_list
if s[nsel-1] GT (nlist-1) Then Return,0


If s[nsel-1] EQ nList-1 then Begin
 *(self.list)[1] = Shift(*(self.list)[1], 1) ;move the bottom to the top
 EndIf Else begin
 Index = Lindgen(Nlist)
 If nsel Eq 1 then $
  Index[s[0]:s[0]+1L] = Shift(Index[s[0]:s[0]+1L],1) Else $
  Index[(s[0]-1L ) : (s[nSel-1] + 1L)] = SHIFT(Index[(s[0]-1L) :
(s[nSel-1] + 1L)] ,1)

 (*(Self.List)[1])= (*(Self.List)[1])[Index]

 EndElse

Widget_control, Self.ListID[1], Set_Value = STRING( *(self.List)[1] )
Return, 0
END ;MoveDown event

;------
```

```
; ListSelect - doesn't do anything (yet) except receive events
;------
FUNCTION HBB_TwoList::ListSelect, event
Catch, Error
If Error NE 0 then Begin
 Self->ErrorMessage
 Catch, /Cancel
 Return,0
 EndIF

Return, 1
END ;ListSelect


;------
; BuildGUI
;------
PRO HBB_TwoList::BuildGUI
Catch, Error
If Error NE 0 then Begin
 Self->ErrorMessage
 Catch, /Cancel
 Return
 EndIF

colbase = widget_base(self.layoutid, /base_align_center, column = 3)

width = (MAX(STRLEN(*self.list[0]))+5)>20
Height = self.ListLength[0] > (n_elements(*(Self.List)[0])+3) < Self.ListLength[1]
 ;column base for Primary List
PID = widget_base(colbase, /base_align_center, column = 1, frame = 1)
pLabel = widget_Label(PID, Value = self.LabelNames[0], /align_center)
pList = Widget_List(PID, Value = STRING(*(Self.List)[0]), $
 ySize = height,xsize = width, $
 Multiple = Self.Multiple,  $
 uValue = {Object:Self, Method: 'ListSelect' })

list_geo = widget_info(pList, /Geometry)

ButtonBase = Widget_Base(colbase, /base_align_center, column = 1, space
= 10)
LabelID = Widget_Label(ButtonBase, value = 'Modify '+self.labelnames[1]+
' Field', $
 /align_center)

Move =   '>>> Move >>>'
Copy =  '>>> Copy >>>'
Remove =  '<< Remove <<'
```

```
MoveUp =  '   Move Up  '
MoveDown =  '  Move Down '

TransferID = Widget_Base(ButtonBase, Column = 1, /base_align_center)
 AddID = Widget_Button(TransferID, Value = copy,$
  uValue = {Object:self, Method:'CopyItem'})

But_geo = widget_info(AddID, /Geometry)

 RemoveID= Widget_Button(TransferID, Value = remove,$
  xsize = but_geo.xsize, $
  uValue = {Object:self, Method:'RemoveItem'})



SID = widget_Base(ColBase, /Base_align_center, Column = 1, Frame = 1)
sLabel = widget_Label(SID, Value = self.LabelNames[1], /align_center)

 ;create the list with or without a value
if n_elements(*(self.list)[1]) NE 0 then $
 sList = Widget_List(SID, Value = STRING(*(Self.List)[1]), $
  ySize = height,xsize = width, $
  Multiple = Self.Multiple, $
  uValue = {Object:Self, Method: 'ListSelect'}) Else $
 sList = Widget_List(SID, $
  ySize = height,xsize = width, $
  Multiple = Self.Multiple,  $
  uValue = {Object:Self, Method: 'ListSelect' })

Self.ListID = [pList, sList]

UpID = Widget_Button(SID, Value = moveup,$
  xsize = but_geo.xsize, $
  uValue = {Object:self, Method:'MoveUp' }, $
  sensitive = self.UPDOWN )

DownID = Widget_Button(SID, value = movedown,$
  xsize = but_geo.xsize, $
  uValue = {Object:self, Method:'MoveDown' }, $
  sensitive = self.UPDOWN )


END ;BuildGUI

;------
; GetValue
;------
FUNCTION HBB_TwoList::GetValue, event
```

```
Catch, Error
If Error NE 0 then Begin
 Self->ErrorMessage
 Catch, /Cancel
 Return,0
 EndIF


Value = {S_LIST: *Self.List[1]}
Return, Value
END ;GetValue


;------
; GetState
;------
FUNCTION HBB_TwoList::GetState, event
Catch, Error
If Error NE 0 then Begin
 Self->ErrorMessage
 Catch, /Cancel
 Return,0
 EndIF

If Widget_Info(self.ListID[1], /Valid_ID) Then $
 S = Widget_Info(self.ListID[1], /List_Select) Else $
 S = -1

State = {S_LIST:*Self.List[1], S_Select:S }
Return, State
END ;GetState

;------
; GetProperty
;------
PRO HBB_TwoList::GetProperty, P_list=P_List, S_list = S_List, $
 P_Label = P_Label, S_Label = S_Label, $
 UPDOWN = UPDOWN, Multiple = Multiple, $
 Min_Length = Min_length, Max_length = Max_Length, $
 _Ref_Extra = Extra

Catch, Error
If Error NE 0 then Begin
 Self->ErrorMessage
 Catch, /Cancel
 Return
 EndIF
```

```
If arg_present(p_list) then P_List = *(Self.List)[0]
If Arg_present(S_LIST) Then s_List = *(Self.List)[1]

P_Label = Self.LabelNames[0]
S_label = Self.LabelNames[1]

UPDOWN =  Self.UPDOWN
Multiple = Self.Multiple
No_Duplicates = Self.No_Duplicates

Min_Length = Self.ListLength[0]
Max_length = Self.ListLength[1]

Self->MGS_GUIObject::GetProperty, _Extra = Extra

END ;SetProperty


;------
; SetProperty
;------
PRO HBB_TwoList::SetProperty, P_list=P_List, S_list = S_List, $
 P_Label = P_Label, S_Label = S_Label, $
 UPDOWN = UPDOWN, Multiple = Multiple, $
 Min_Length = Min_length, Max_length = Max_Length, $
 No_Duplicates = No_Duplicates, $
 _Ref_Extra = Extra

Catch, Error
If Error NE 0 then Begin
 Self->ErrorMessage
 Catch, /Cancel
 Return
 EndIF

If N_Elements(P_LIST) NE 0 Then *(Self.List)[0] = P_LIST
If N_elements(S_LIST) NE 0 Then *(Self.List)[1] = S_LIST

If N_elements(P_label) NE 0 Then Self.LabelNames[0] = P_Label[0]
If N_elements(S_Label) NE 0 Then Self.LabelsNames[1] = S_Label[0]

If N_Elements(UPDOWN) NE 0 Then Self.UPDOWN = Keyword_Set(UPDOWN)
If n_elements(Multiple) NE 0 Then Self.Multiple = KeyWord_Set(Multiple)
If n_elements(No_Duplicates) NE 0 Then $
  Self.No_Duplicates = KeyWord_Set(No_Duplicates)

If n_elements(Min_Length) NE 0 Then Self.ListLength[0] = Min_Length
```

```
If N_elements(Max_Length) NE 0 Then Self.ListLength[1] = Max_length

Self->MGS_GUIObject::SetProperty, _Extra = Extra

END ;SetProperty

;------
; INIT
;------
FUNCTION HBB_TwoList::Init, $
 P_list=P_List, S_list = S_List, $
 P_Label = P_Label, S_Label = S_Label, $
 UPDOWN = UPDOWN, Multiple = Multiple, $
 Min_Length = Min_length, Max_length = Max_Length, $
 No_Duplicates = No_Duplicates, $
 _Ref_Extra = Extra

IF NOT Self->MGS_GUIobject::INIT(_Extra = Extra) Then Return, 0

Catch, Error
If Error NE 0 then Begin
 Self->ErrorMessage
 Catch, /Cancel
 Return,0
 EndIF

self.list = PtrArr(2,/Allocate)
If n_elements(P_List) EQ 0 then $
 p_list = ['Martin', 'Goofy', 'David', 'DubYa', 'Mark', 'Mickey',$
  'Craig', 'Minnie','JD',  'Henry', 'Pavel']
*Self.List[0] = P_List

If n_elements(S_list) EQ 0 then $
 *Self.List[1] = [''] Else *Self.list[1] = S_list

If n_elements(P_label) NE 0 then Self.LabelNames[0] = P_Label[0]
If n_elements(S_label) NE 0 Then Self.LabelNames[1] = S_Label[0]

Self.UPDOWN = keyword_set(UPDOWN)
Self.Multiple = keyword_set(Multiple)
Self.No_Duplicates = Keyword_Set(No_Duplicates)

If n_elements(Min_length) EQ 0 then min_L = 10 Else min_L = Min_Length[0]
If n_elements(Max_Length) EQ 0 Then max_L = 30 Else Max_L = Max_Length[0]
Self.ListLength = [min_L, max_L]


Return, 1
```

```
END ;INIT

;-----
; CleanUp
;-----
PRO HBB_TwoList::CleanUp

 Ptr_Free, Self.List

 Self->MGS_GUIObject::Cleanup

END ;CleanUp

;-----
; Defintion
;-----
PRO HBB_TwoList__Define

 struct = {HBB_TwoList, $
  INHERITS MGS_GUIOBJECT, $

  LabelNames:strArr(2), $;Labels for each list
  LabelID:LonArr(2), $ ;labelIDs
  List:PtrArr(2), $   ;list contents
  ListID: LonArr(2), $ ;widgetIDs for lists
  UPDOWN:0, $     ;1 = allow up/down movement
  Multiple:0,$    ;1 = allow multiple selections
  No_Duplicates: 0, $  ;are duplicates permitted in s_list?
  ListLength:IntArr(2)} ;minmax list length (10, 30 default)
END
;------ENDHERE
```

--
Ben Tupper
Bigelow Laboratory for Ocean Sciences
180 McKown Point Rd.
W. Boothbay Harbor, ME 04575
btupper@bigelow.org