Subject: Re: Updateable Message Widget
Posted by knighton on Wed, 02 Aug 1995 07:00:00 GMT
View Forum Message <> Reply to Message

In <3vlg0b$het@ncar.ucar.edu> cavanaug@uars1.acd.ucar.edu (Charles Cavanaugh) writes:

> I've been trying to develop a clean method of having an updateable window
> in some of my larger programs. The plan is to use this window to keep the program
> user current on the program's processing status (rather than just printing to the
> xterm). I finally settled on the method below, but it's not as clean as I would like.
> Notice that the widget is realized but not registered (and therefore, I assume, needs
> no event handler), and that to destroy the widget, I have to pass widget id - 1 to
> widget_control. Weird. Does anyone have a better way of handling this?

Some ways of getting around using wId-1:

Use WIDGET_INFO(wId, /PARENT)

or use the PRO_SET_VALUE keyword when making the base widget to cause
the WIDGET_CONTROL, wBase, SET_VALUE='...' to call a specified procedure
that will then locate the label widget and change its value.

I have appended a message box compound widget that I use extensively to
do just about everything, including what you described. If you call it
with the /INFO_BOX keyword, it will do just what you want.

Disclaimer: Software is provided as is, etc...

---------- Snip ----------

```
;cw_msgbox.pro
;
;
;
;
;+
; NAME:
; CW_MSGBOX - message box compound widget.
;
; PURPOSE:
; This compound widget function creates a message box to display information
; and to get a simple response from the user. This function can be used to
; display error messages and other information as well as to get simple
; button activated responses such as "OK", "CANCEL", "SAVE", etc. In its
; simplest form, this widget displays a message and allows the user to
; acknowledge this message by pressing an "OK" button. This can be used to
; provide a simple means of displaying error messages.
;
```

```
; CATEGORY:
; Compound Widgets
;
; CALLING SEQUENCE:
; widget_id = CW_MSGBOX(Parent, Button_values)
;
; INPUTS:
; Parent:  The ID of the parent widget.
;
; KEYWORD PARAMETERS:
; VALUE:  A string array containing the message to be displayed
;   in the message box.  After the message box is created,
;   this message can be changed but the new value string
;   array can not have any more elements than the original.
;   i.e. the message box will not grow.  Default:  If no
;   message is specified, then a snide remark of my choosing
;   will be placed in the box instead.  If you really want
;   a blank box, then set VALUE="   ".
; BUTTON_VALUES: A string array, containing one string per button to be
;                   displayed for the user's response.  Also, if this
;   keyword is specified, then events will be returned by
;   the compound widget.  Default:  One "OK"
;   button will be displayed if this keyword is not used and
;   no events will be returned.
; TITLE:  A title to be displayed in the window's title box.  The
;   default is "Message".
; DEFAULT_VALUE: The value returned in the event sent as a result of the
;   widget being destroyed by means other than the normal
;   processing sequence.  (i.e. by the window manager, or
;   as a result of the group leader being destroyed.)  If
;   BUTTON_VALUES is not used, then no events are returned.
;   Default:  "DESTROYED"
; RESOURCE_NAME: X-windows resource name to be given to this widget.
; /INFO_BOX: Creates a box with no buttons for displaying messages
;   only.  Sets /PERSIST and if not overridden by the
;   EVENTS keyword, no events are returned.  BUTTON_VALUES
;   is ignored.  The XMANAGER is not called.
; DISPLAY_TIME: Causes message box to be automatically destroyed after
;   a specified number of seconds.  Overrides /PERSIST
;   keyword and turns it off.  Seconds should be specified
;   as a floating point value.  This keyword can be used to
;   cause an INFO_BOX to display for a specified time.  It
;   can also be used to force a user response to a message
;   box within a specific time period.  If the user doesn't
;   respond, then the DEFAULT_VALUE is returned with any
;   events sent back to the parent.  Default is for no
;   timeout period.
; EVENTS:  1 => events are returned.  0 => events are not returned.
```

```
;   If present, this keyword overrides any other features
;   that cause events to be or not to be returned.
;   Default:  Dependent on BUTTON_VALUES.
; /SENSITIVE: Causes the widget hierarchy rooted at Parent to remain
;   sensitive to events.  Default:  The hierarchy rooted at
;   Parent is desensitized.  For example:  An error message
;   box may have the top level base as its Parent.  Thus,
;   the entire window would be desensitized.
; /PERSIST: The first button press does not cause the message box to
;   be destroyed.  It will hang around until explicitly
;   destroyed.  Default:  The message box is destroyed after
;   the first button press.
; /BELL:  A bell sounds when the box is created and any time the
;   value is changed.  Default: no bell is sounded.
; UVALUE:  A user value to give to the widget.
; FONT:  A font to use when creating the widget.
; XOFFSET: The X offset of the widget relative to its parent.
;   default: based on xsize and position of parent's top
;   level base.
; YOFFSET: The Y offset of the widget relative to its parent.
;   default: based on ysize and position of parent's top
;   level base.
;
; OUTPUTS:
; The ID of the created widget is returned.
;
; SIDE EFFECTS:
;
; This routine starts the XMANAGER if it isn't already running unless INFO_BOX
; is specified.
;
; This widget may generate events with the following structure definition:
;
; event = {ID:parent, TOP:tlbase, HANDLER:0L, ORIGIN:cw_msgbox_id, $
;    VALUE:0 [,UVALUE:0]}
;
; VALUE is the string value of the button that was pressed.  If /PERSIST
; is not specified, then the user value of the compound widget will be
; returned as part of the event structure.
; NOTE:  If BUTTON_VALUES is not specified, then no events are returned,
; rather, when the single "OK" button is pressed, the message box is
; destroyed, and the event handler returns without generating events.
;
;                               WARNING!
;
; Prior to IDL 4.0:
; When a modal widget is registered with the XMANAGER, it starts a second
; event processing loop.  Any further calls to the XMANAGER to register
```

```
; a widget (modal or non-modal) will cause the XMANAGER to start another
; loop.  Furthermore, the third, fourth, etc. loops don't end until all
; of the widgets registered after the modal widget and the original modal
; widget itself is destroyed.  Because of this effect, CW_MSGBOX should not
; be used if a modal widget is registered.  If it is used under these
; circumstances, then it won't return to the calling program until after
; all of the modal widgets are destroyed.  This can cause weird things
; to happen with your program.
;
;
; PROCEDURE:
; EXAMPLE:
;
; message = CW_MSGBOX(Parent, $
;   VALUE=['File not saved', $
;   'Exit anyway?'], $
;   BUTTON_VALUES=['OK', 'CANCEL'], $
;   TITLE='Warning')
;
; Produces:
;
;       ------------------------------------
;      | - |         Warning            |o|O|
;      |------------------------------------|
;      |                                    |
;      |          File not saved.           |
;      |           Exit anyway?             |
;      |                                    |
;      |     ---------     --------------   |
;      |    |  OK     |   |   CANCEL     |  |
;      |     ---------     --------------   |
;       ------------------------------------
;
; or some reasonable facsimile thereof, desensitizes the parent,
; disappears when one of the buttons is pressed, and sends an event
; to parent.
;
; MODIFICATION HISTORY:
;  Written by: Ken Knighton
;   knighton@gav.gat.com
;   Fusion Division
;   General Atomics
;   San Diego, CA
;
; Date:  2/6/95
;
; 2/16/95 KEK Added /INFO_BOX keyword.
;
```

```
; 2/20/95 KEK Added EVENTS keyword.
;
; 4/10/95 KEK Added DISPLAY_TIME keyword.
;
; 4/20/95 KEK Put XMANAGER call at very end of CW_MSGBOX procedure
;
; 4/25/95 KEK Changed so that XMANAGER is not called if /INFO_BOX is
;   specified.
;
; 5/9/95  KEK Added RESOURCE_NAME keyword
;
;-

;This function receives events from the message box and processes them
;as necessary.  This function is the event handler for the message box
;compound widget.
PRO CW_MSGBOX_EVENT, ev

  ON_ERROR, 2   ;Return to caller

  ;Get the state information for this compound widget.
  stateholder = WIDGET_INFO(ev.top, /CHILD)

  WIDGET_CONTROL, stateholder, GET_UVALUE=state, /NO_COPY

  ;If this is not a timer event,
  IF ev.id NE state.cw_msgbox_id THEN BEGIN

    ;Get the value of the button that created the event
    WIDGET_CONTROL, ev.id, GET_VALUE=value, /NO_COPY

  ENDIF ELSE BEGIN

    ;Use the default event value for any events returned to the parent
    value = state.default_value

  ENDELSE

  ;If events are to be generated,
  IF state.events NE 0 THEN BEGIN

    ;Get the uvalue from the message box
    WIDGET_CONTROL, ev.top, GET_UVALUE=uvalue, /NO_COPY

    ;Send an event back to the parent.
    WIDGET_CONTROL, state.parent, $
  SEND_EVENT= {ID:state.parent, TOP:state.tlbase, $
    HANDLER:0L, ORIGIN:ev.top, VALUE:value, $
```

```
    UVALUE:uvalue}

    ;Restore the uvalue for the message box
    WIDGET_CONTROL, ev.top, SET_UVALUE=uvalue, /NO_COPY

  ENDIF

  ;If the message box is to be destroyed after the first event,
  IF state.persist EQ 0 THEN BEGIN

    ;Disable the KILL_NOTIFY procedure for the stateholder to prevent
    ;calling it when the message box is destroyed.
    WIDGET_CONTROL, stateholder, KILL_NOTIFY=""

    ;Destroy the message box.
    WIDGET_CONTROL, ev.top, /DESTROY

  ENDIF

  ;If the parent was desensitized,
  IF state.sensitive EQ 0 THEN $ ;Resensitize it.
    WIDGET_CONTROL, state.parent, /SENSITIVE, /HOURGLASS

  ;If the message box wasn't destroyed,
  IF state.persist NE 0 THEN BEGIN

    ;Restore the state information
    WIDGET_CONTROL, stateholder, SET_UVALUE=state, /NO_COPY

    ;Restore the button value
    WIDGET_CONTROL, ev.id, SET_VALUE=value, /NO_COPY

  ENDIF

  RETURN

END  ;CW_MSGBOX_EVENT

;Call back procedure that is called if widget is destroyed by something other
;than the CW_MSGBOX_EVENT function.  This procedure takes care of loose ends so
;that the calling application is able to keep track of the widget destruction.
PRO CW_MSGBOX_DIED, id

  ON_ERROR, 2

  ;Get the state information from the widget
  WIDGET_CONTROL, id, GET_UVALUE=state, /NO_COPY
```

```
  ;If events are to be returned, then return one.
  IF state.events NE 0 THEN BEGIN

    ;Get the user value of the message box.
    WIDGET_CONTROL, state.cw_msgbox_id, GET_UVALUE=uvalue

    ;If the parent exists,
    IF WIDGET_INFO(state.parent, /VALID_ID) THEN $ ;Send the event to the parent.
      WIDGET_CONTROL, state.parent, $
  SEND_EVENT={ID:state.parent, TOP:state.tlbase, $
                  HANDLER:0L, ORIGIN:state.cw_msgbox_id, $
                  VALUE:state.default_value, UVALUE:uvalue }

  ENDIF

  ;If the parent exists and was desensitized,
  IF WIDGET_INFO(state.parent, /VALID_ID) AND state.sensitive EQ 0 THEN $
    ;Resensitize it.
    WIDGET_CONTROL, state.parent, /SENSITIVE, /HOURGLASS

  RETURN

END  ;CW_MSGBOX_DIED

;This procedure sets the value of the text in the message box.
PRO CW_MSGBOX_SET_VAL, cw_msgbox_id, value

  ON_ERROR, 2

  ;Get the state information from the compound widget to obtain the
  ;label widget id.
  stateholder=WIDGET_INFO(cw_msgbox_id, /CHILD)
  WIDGET_CONTROL, stateholder, GET_UVALUE=state, /NO_COPY

  ;Determine the number of lines of text currently displayed
  num_labels = N_ELEMENTS(state.label)

  ;Determine the number of lines to be displayed.
  value_size = N_ELEMENTS(value)

  ;For all of the lines of text currently displayed,
  FOR i = 0, num_labels-1 DO BEGIN

    ;If there are new lines left to display,
    IF i LT value_size THEN $ ;display them
      WIDGET_CONTROL, state.label(i), SET_VALUE=value(i) $
    ELSE $   ;display a blank line
      WIDGET_CONTROL, state.label(i), SET_VALUE=""
```

```
  ENDFOR

  ;If the bell was rung when the box was created,
  IF state.bell NE 0 THEN $ ;ring it again
    PRINT, STRING(7B)

  ;Restore the state information
  WIDGET_CONTROL, stateholder, SET_UVALUE=state, /NO_COPY

  RETURN

END  ;CW_MSGBOX_SET_VAL

;This function gets the value of the text in the message box.
FUNCTION CW_MSGBOX_GET_VAL, cw_msgbox_id

  ON_ERROR, 2

  ;Get the state information from the compound widget to obtain the
  ;label widget id.
  stateholder=WIDGET_INFO(cw_msgbox_id, /CHILD)
  WIDGET_CONTROL, stateholder, GET_UVALUE=state, /NO_COPY

  ;Determine the number of text lines in the widget.
  num_labels = N_ELEMENTS(state.label)

  ;Make an array into which to place the text lines.
  value=STRARR(num_labels)

  ;For each of the lines
  FOR i = 0, num_labels - 1 DO BEGIN

    ;Get the value of the text.
    WIDGET_CONTROL, state.label(i), GET_VALUE=value(i)

  ENDFOR

  ;Restore the state information
  WIDGET_CONTROL, stateholder, SET_UVALUE=state, /NO_COPY

  RETURN, value

END  ;CW_MSGBOX_GET_VAL


;This function creates the CW_MSGBOX widget.
FUNCTION CW_MSGBOX, parent, $
```

```
 VALUE=value, $
 BUTTON_VALUES=button_values, $
 TITLE=title, $
 DEFAULT_VALUE=default_value, $
 RESOURCE_NAME=resource_name, $
 INFO_BOX=info_box, $
 DISPLAY_TIME=display_time, $
 EVENTS=events_setting, $
 SENSITIVE=sensitive, $
 PERSIST=persist, $
 BELL=bell, $
 UVALUE=uvalue, $
 FONT=font, $
 XOFFSET=xoffset, $
 YOFFSET=yoffset

ON_ERROR, 2   ;Return to the calling routine

;Check the keyword parameters and set them to reasonable values if necessary.
IF (KEYWORD_SET(value) EQ 0) THEN value = "This box intentionally left blank"
IF (KEYWORD_SET(button_values) EQ 0) THEN BEGIN
  button_values="OK"
  num_buttons=1
  events=0
ENDIF ELSE BEGIN
  num_buttons=N_ELEMENTS(button_values)
  events=1
ENDELSE
IF (KEYWORD_SET(title) EQ 0) THEN title = "Message"
IF (KEYWORD_SET(default_value) EQ 0) THEN default_value="DESTROYED"
IF (KEYWORD_SET(resource_name) EQ 0) THEN resource_name="msgbox"
IF (KEYWORD_SET(sensitive) EQ 0) THEN sensitive=0
IF (KEYWORD_SET(persist) EQ 0) THEN persist=0
IF (KEYWORD_SET(bell) EQ 0) THEN bell=0
IF (KEYWORD_SET(uvalue) EQ 0) THEN uvalue = 0
IF (KEYWORD_SET(font) EQ 0) THEN font = ""
IF (KEYWORD_SET(info_box) NE 0) THEN BEGIN
  num_buttons=-1
  events=0
ENDIF
IF KEYWORD_SET(display_time) EQ 0 THEN BEGIN
  display_time=0.0
ENDIF ELSE BEGIN
  persist=0
ENDELSE
IF (KEYWORD_SET(events_setting) NE 0) THEN events = events_setting

;Get the location of the upper left hand corner of the top level base
```

```
;for of the parent widget.
WIDGET_CONTROL, parent, TLB_GET_SIZE=xy_size, TLB_GET_OFFSET=xy_offset

;Give a reasonable default for the message box location.
IF (KEYWORD_SET(xoffset) EQ 0) THEN xoffset = xy_offset(0)+xy_size(0)/2
IF (KEYWORD_SET(yoffset) EQ 0) THEN yoffset = xy_offset(1)+xy_size(1)/2

;Find the top level base of the parent's widget hierarchy by traversing upwards
;until there are no more parents.
temp = parent
REPEAT BEGIN
  tlbase = temp
  temp = WIDGET_INFO(tlbase, /PARENT)
ENDREP UNTIL temp EQ 0

;Create the top level base widget for the message box
cw_msgbox_id = WIDGET_BASE(TITLE=title, $
   GROUP_LEADER=parent, $
   RESOURCE_NAME=resource_name, $
   PRO_SET_VAL='CW_MSGBOX_SET_VAL', $
   FUNC_GET_VAL='CW_MSGBOX_GET_VAL', $
   UVALUE=uvalue, $
   /COLUMN, $
   XOFFSET=xoffset, $
   YOFFSET=yoffset, $
   SPACE=0, $
   XPAD=0, $
   YPAD=0)

;Determine the number of value lines to display
value_size=N_ELEMENTS(value)

;Make an array to hold the label widget ids
label = LONARR(value_size)

;If the FONT keyword was specified,
IF FONT NE "" THEN BEGIN ;Create the widgets with the given font.

  ;Put blank lines around the value to ensure good looks and a minimum
  ;width for the message box.
  stateholder=WIDGET_LABEL(cw_msgbox_id, $
   VALUE='                                    ', $
   KILL_NOTIFY="CW_MSGBOX_DIED", $
   FONT=font)

  ;For all of the lines of text,
  FOR i = 0, value_size-1 DO BEGIN
```

```
    ;Display the text
    label(i)=WIDGET_LABEL(cw_msgbox_id, VALUE=value(i), FONT=font)

  ENDFOR

  ;Put blank lines around the value to ensure good looks and a minimum
  ;width for the message box.
  junk=WIDGET_LABEL(cw_msgbox_id, $
   VALUE='                                    ', $
   FONT=font)

  ;Create a base into which to put the buttons.
  button_base=WIDGET_BASE(cw_msgbox_id, COLUMN=num_buttons)

  ;For all of the buttons.
  FOR i=0, num_buttons-1 DO BEGIN

    ;Create the button widgets
    junk=WIDGET_BUTTON(button_base, VALUE=button_values(i), $
   FONT=font)
  ENDFOR

ENDIF ELSE BEGIN  ;Create the widgets without specifying a font.

  ;Put blank lines around the value to ensure good looks and a minimum
  ;width for the message box.
  stateholder=WIDGET_LABEL(cw_msgbox_id, $
   VALUE='                                    ', $
   KILL_NOTIFY="CW_MSGBOX_DIED")

  ;For all of the lines of text,
  FOR i = 0, value_size-1 DO BEGIN

    ;Display the text
    label(i)=WIDGET_LABEL(cw_msgbox_id, VALUE=value(i))

  ENDFOR

  ;Put blank lines around the value to ensure good looks and a minimum
  ;width for the message box.
  junk=WIDGET_LABEL(cw_msgbox_id, $
   VALUE='                                    ')

  ;Create a base into which to put the buttons.
  button_base=WIDGET_BASE(cw_msgbox_id, COLUMN=num_buttons)

  ;For all of the buttons.
  FOR i=0, num_buttons-1 DO BEGIN
```

```
      ;Create the button widgets
      junk=WIDGET_BUTTON(button_base, VALUE=button_values(i))

    ENDFOR

  ENDELSE

  ;Save state information for the compound widget.
  cw_state={PARENT:parent, CW_MSGBOX_ID:cw_msgbox_id,  LABEL:label, $
     SENSITIVE:sensitive, $
     PERSIST:persist, EVENTS:events, BELL:bell, $
     TLBASE:tlbase, DEFAULT_VALUE:default_value }

  ;Store the message box state info in the first child of the message box
  WIDGET_CONTROL, stateholder, SET_UVALUE=cw_state, /NO_COPY

  ;Realize the widgets.
  WIDGET_CONTROL, cw_msgbox_id, /REALIZE

  ;If the user wants the message box to display for a fixed time period.
  IF display_time NE 0.0 THEN BEGIN

    ;Create a time event to happen at the specified time
    WIDGET_CONTROL, cw_msgbox_id, TIMER=display_time

  ENDIF

  ;If a bell is to be sounded,
  IF bell NE 0 THEN $ ;sound it.
    PRINT, STRING(7B)

  ;If the parent hierarchy is to be desensitized,
  IF sensitive EQ 0 THEN $ ;desensitize it and clear any outstanding events
    WIDGET_CONTROL, parent, SENSITIVE=0, /CLEAR_EVENTS, /HOURGLASS

  ;If there are buttons or events are to be returned,
  IF num_buttons GE 0 OR events NE 0 THEN BEGIN

    ;Register the message box with the XMANAGER.
    XMANAGER, 'cw_msgbox', cw_msgbox_id

  ENDIF

  RETURN, cw_msgbox_id

END
```