Subject: Re: Slicer3 followup
Posted by thompson on Wed, 20 Jan 1999 08:00:00 GMT
View Forum Message <> Reply to Message

Bob Yantosca <bmy@io.harvard.edu> writes:

> I am running IDL 5.1 on the SGI-Origin platform, and have noticed a couple
> of things about the library routine "slicer3.pro"

> (1) slicer3.pro contains the following call to XMANAGER with the /MODAL
>   keyword:

```
>      XMANAGER, 'Slicer3', wMainBase, EVENT_HANDLER='Viz3D_Event', $
>         /MODAL, CLEANUP='Viz3D_KillMain'
```

> However, in IDL 5.1, the /MODAL keyword in XMANAGER has been superseded by
> a call to /MODAL in WIDGET_BASE. Am not sure what side effects this will
> cause, but at some point perhaps RSI should tidy this up.

I hope that the way RSI tidies this up is to change their mind and decide that
/MODAL in XMANAGER isn't obsolete after all. As discussed previously in this
newsgroup, the newly recommended way of using /MODAL in WIDGET_BASE does not
provide some crucially important functionality that has always been available
with XMANAGER.

I don't think your problem is caused by the /MODAL keyword, but RSI did give me
some directions as to how to fix xmanager.pro in IDL/v5.1 and /v5.2. The fixed
version is below. See if this solves your problems.

William Thompson

```
 ==============================================================
===================
; $Id: xmanager.pro,v 1.42 1998/01/21 22:09:16 lubos Exp $
;
; Copyright (c) 1991-1998, Research Systems, Inc.  All rights reserved.
; Unauthorized reproduction prohibited.

;+
; NAME:
; XMANAGER
;
; PURPOSE:
; Provide management for widgets client applications created using IDL.
;
; CATEGORY:
```

```
; Widgets.
;
; CALLING SEQUENCE:
; XMANAGER [, Name, ID]
;
; OPTIONAL INPUTS:
; NAME: A string giving the name of the application that is being
;  registered.
;
; ID: The widget ID of the top level base of the new client.
;
; KEYWORD PARAMETERS:
; BACKGROUND:
;   ------------------------------------------------------------ -
;  | PLEASE NOTE: This keyword is OBSOLETE. It's functionality |
;  | is provided by the TIMER keyword to the WIDGET_CONTROL    |
;  | procedure.                                                |
;   ------------------------------------------------------------ -
;
; CATCH: If TRUE, tells XMANAGER to use CATCH when dispatching
;  widget events. If FALSE, CATCH is not used and execution
;  halts on error. The default is TRUE. If CATCH is specified,
;  the internal state of XMANAGER is updated and it returns
;  immediately without taking any further action. CATCH
;  is only effective if XMANAGER is blocking to dispatch
;  errors. If active command line event dispatching is in
;  use, it has no effect.
;
; CLEANUP: This keyword contains a string that is the name of the
;  routine called when the widget dies.  If not specified,
;  no routine is called.  The cleanup routine must accept one
;  parameter which is the widget id of the dying widget. This
;  routine is set as the KILL_NOTIFY routine for the widget.
;
; EVENT_HANDLER: The name of the event handling routine that is to be
;  called when a widget event occurs in the registered
;  application. If this keyword is not supplied, the Xmanager
;  will construct a default name by adding the "_EVENT" suffix
;  to the NAME argument. See below for a more detailed
;  explanation.
;
; GROUP_LEADER: The widget id of the group leader for the application
;  being registered.  When the leader dies, all widgets that have
;  that leader will also die.
;
;  For example, a widget that views a help file for a demo
;  widget would have that demo widget as it's leader.  When
;  the help widget is registered, it sets the keyword
;  GROUP_LEADER to the widget id of the demo widget. If
;  the demo widget is destroyed, the help widget led by
;  the it would be killed by the XMANAGER.
;
; JUST_REG:
;  This keyword tells the manager to just register the widget
;  but not to start doing the event processing.  This is useful
;  when you want to register a group of related top level widgets
;  but need to regain control immediately afterwards.
;
;  NOTE: JUST_REG does not do the same thing as NO_BLOCK. This is
;  explained in detail below under "SIDE EFFECTS".
;
;  MODAL:
;   ---------------------------------------------------------- --
;  | PLEASE NOTE: This keyword is OBSOLETE. It's functionality  |
;  | is provided by the MODAL keyword to the WIDGET_BASE        |
;  | procedure.                                                  |
;   ---------------------------------------------------------- --
;
;  When this keyword is set, the widget that is being registered
;  traps all events and desensitizes all the other widgets.  It
;  is useful when input from the user is necessary before
;  continuing. Once the modal widget dies, the others are
;  resensitized and the normal event processing is restored.
;  XMANAGER is therefore using sensitivity to provide the
;  illusion of modality. The WIDGET_BASE keyword is a newer
;  IDL feature that provides the real thing.
;
; NO_BLOCK: If set, tells XMANAGER that the registering client
;  does not require XMANAGER to block if active command line
;  event processing is available. If active command line
;  event processing is available *AND* every current XMANAGER
;  client specifies NO_BLOCK, then XMANAGER will not block
;  and the user will have access to the command line while
;  widget applications are running.
;
;  NOTE: NO_BLOCK does not do the same thing as JUST_REG. This is
;  explained in detail below under "SIDE EFFECTS".
;
; OUTPUTS:
; No outputs.
;
; COMMON BLOCKS:
; MANAGED
; XMANAGER_LOCAL:
;  Common blocks used for module state maintenance. These common
;  blocks are considered private to this module and should not
;  be referenced outside RSI supplied routines. They are
;  subject to change without notice.
;
;
; SIDE EFFECTS:
;
;   JUST_REG vs NO_BLOCK
;   --------------------
;      Although their names imply a similar function, the JUST_REG and
; NO_BLOCK keywords perform very different services. It is important
; to understand what they do and how they differ.
;
;      JUST_REG tells XMANAGER that it should simply register a client
; and then return immediately. The result is that the client becomes
; known to XMANAGER, and that future calls to XMANAGER will take this
; client into account. Therefore, JUST_REG only controls how the
; registering call to XMANAGER should behave. The registered client
; can still be registered as requiring XMANAGER to block by not setting
; NO_BLOCK. In this case, future calls to XMANAGER will block.
;
; NO_BLOCK tells XMANAGER that the registered client does not
; require XMANAGER to block if the command processing front end
; is able to support active command line event processing (described
; below). XMANAGER remembers this attribute of the client until
; the client exits, even after the call to XMANAGER that registered the
; client returns. NO_BLOCK is just a "vote" on how XMANAGER should
; behave. The final decision is made by XMANAGER by considering the
; NO_BLOCK attributes of all of its current clients as well as the
; ability of the command front end in use to support the active command
; line.
;
;   Blocking vs Non-blocking
;   ------------------------
; The issue of blocking in XMANAGER requires some explanation.
; IDL places incoming widget events into a queue of pending events.
; The only way to get these events processed and dispatched is to
; call the WIDGET_EVENT function. Arranging for WIDGET_EVENT to be
; called properly is the primary job of XMANAGER. XMANAGER offers
; two different modes of operation:
;
;     - The first (outermost) XMANAGER processes events by calling
;       WIDGET_EVENT as necessary until no managed clients remain on
;       the screen. This is referred to as "blocking", because XMANAGER
;       does not return to the caller until it is done, and the IDL
;       command line is not available.
;
;     - XMANAGER does not block, and instead, the part of IDL
;       that reads command input also watches for widget events
;       and calls WIDGET_EVENT as necessary while also reading
;       command input. This is referred to as "non-blocking" or
;       "active command line" mode.
;
; The default is to block. However, if every currently active
; application specified the NO_BLOCK keyword to XMANAGER, non-blocking
; mode is used, if possible.
;
; There are currently 5 separate IDL command input front end
; implementations:
;
;  - Apple Macintosh IDE
;  - Microsoft Windows IDE
;  - Motif IDE (Unix and VMS)
;  - Unix plain tty
;  - VMS plain tty
;
; Except for the VMS plain tty, all of these front ends are able to
; support the non-blocking active command line. VMS users can have
; an active command line by using the IDLde interface. The decision
; on whether XMANAGER blocks to process widget events is determined
; by the following rules, in order of precedence:
;
;     - Use of the MODAL keyword will cause XMANAGER to block.
;     - Setting JUST_REG to 1 ensures that XMANAGER will not block.
;     - If using the VMS plain tty interface, XMANAGER will block.
;     - If none of the previous rules apply, XMANAGER will block
;       if any of its currently active clients were registered without
;       specifying NO_BLOCK. If NO_BLOCK is specified for every client,
;       XMANAGER will not block and will instead return and allow
;       active command line processing to take place.
;
; When possible, applications should set the NO_BLOCK keyword.
; This allows the IDL command line to be active while events are
; being processed, which is highly desirable.
;
;
; RESTRICTIONS:
; The implementation of XMANAGER may change in the future. Details
; of its internal implementation must not be relied upon --- only
; its external definition can be considered stable.
;
; XMANAGER uses several undocumented features provided by the
; internal WIDGET routines. These features are private to RSI, and
; are not guaranteed to remain in IDL or to remain unchanged. They
; exist only to support XMANAGER and should not be used elsewhere:
;
;  WIDGET_CONTROL, /XMANAGER_ACTIVE_COMMAND
;  WIDGET_CONTROL, /MODAL
;  WIDGET_EVENT,   /BREAK_ON_EXPOSE
;  WIDGET_EVENT,   /EVENT_BREAK
;  WIDGET_EVENT,   /XMANAGER_BLOCK
;  WIDGET_INFO,    /XMANAGER_BLOCK
;
; These features are undocumented because they are not considered
; permanent. Research Systems reserves the right to remove or alter
; these features at any time.
;
; EXAMPLE USE:
; To create a widget named Example that is just a base widget with a done
; button using the XMANAGER you would do the following:
;
;
; ;------ first - the event handler routine ------;
;
;     PRO example_event, ev   ;this is the routine that
;      ;deals with the events in the
;      ;example widget.
;
; WIDGET_CONTROL, ev.id, GET_UVALUE = uv ;the uservalue is retrieved
;      ;from the widget where the
;      ;event occurred
;
; if(uv eq 'DONE') then $   ;if the event occurred in the
;   WIDGET_CONTROL, ev.top, /DESTROY ;done button then kill the
;     END     ;widget example
;
;
; ;------ second - the main routine ------;
;
;     PRO example    ;this is the main routine
;      ;that builds the widget and
;      ;registers it with the Xmanager
;
; base = WIDGET_BASE(TITLE = 'Example') ;first the base is created
;
; done = WIDGET_BUTTON(base, $  ;next the done button is
;        TITLE = 'DONE', $ ;created and it's user value
;        UVALUE = 'DONE') ;set to "DONE"
;
; WIDGET_CONTROL, base, /REALIZE  ;the widget is realized
;
; XManager, 'example', base  ;finally the example widget
;      ;is registered with the
;      ;Xmanager
;     END
```

```
;
; notes: First the event handler routine is listed.  The handler
;  routine has the same name as the main routine with the
;  characters "_event" added.  If you would like to use another
;  event handler name, you would need to pass it's name in as
;  a string to the EVENT_HANDLER keyword.  Also notice that the
;  event routine is listed before the main routine.  This is
;  because the compiler will not compile the event routine if
;  it was below the main routine.  This is only needed if both
;  routines reside in the same file and the file name is the same
;  as the main routine name with the ".pro" extension added.
;
;
; PROCEDURE:
; When the first widget is registered, initialize the lists and then
; start processing events.  Continue registering widgets and dispatching
; events until all the widgets have been destroyed.  When a widget is
; killed, destroy all widgets that list the destroyed widget as their
; leader, if any.
;
; RELATED FUNCTIONS AND PROCEDURES:
; XREGISTERED, XMTOOL
;
; MODIFICATION HISTORY: Written by Steve Richards, November, 1990
; SMR, Mar,  1991 Added a cleanup routine keyword to allow dying
;     widgets to clean themselves up when dying.
; SMR, May,  1991 Fixed a bug found by Diane Parchomchuk where an error
;     occurred when registering a widget  ight after destroying another.
; SMR & ACY, July, 1991
;     Fixed a bug found by Debra Wolkovitch where lone widgets being
;     destroyed and new ones created caused problems.
; SMR, Sept, 1991 Changed cleanup to use the new WIDGET_INFO routine.
; SMR & ACY, Oct,  1991
;     Fixed a bug where a background event that unregistered itself
;     after a time would result in an XMANAGER error.
;  SMR, Mar.  1992 Changed XMANAGER to use enhanced widget functions for
;     event processing.
; SMR, Nov.  1992 Changed modal widget handling allowing nesting of
;     modal widgets.  The first modal desensitizes all current widgets
;     and subsequent modals only desensitize the modal that called them.
; JIY, Apr.  1993 Changed modal widget handling process to not run the
;     event loop for nested modal widgets. Allowed for multiple modal
;     widgets.
; AB & SMR, 17 November 1993
;     Added ID validity checking to desensitizing of modal widgets to
;     fix a bug where already dead widgets were being accessed.
; DJE, Feb, 1995
;     Made it so that non-modal widgets created from a modal widget have
;     events processed in the modal widget's event loop. This fixes a
;     bug where xmanager wouldn't return immediately if there was a
;     modal widget somewhere in the nesting, even though a non-modal
;     widget was being added. The nesting level could get _very_ deep.
; DJE, Apr 1995
;     Pass a local variable to WIDGET_EVENT in the MODAL case, instead
;     of passing the common block variable modalList. This avoids a bug
;     where modalList gets changed behind WIDGET_EVENT's back.
; DJE, Apr 1996
;     Changes for handling asynchronous widget event dispatching.
;     Complete rewrite. Background tasks are no longer supported. The
;     MODAL keyword is now obsolete. Added CATCH and BLOCK keywords.
; AB, May 1996
;     Made changes so that XMANAGER always blocks under VMS with the
;     non-GUI interface. This is due to the fact that the SMG$ system
;     routines used by IDL in the plain tty case cannot support
;     interleaving of X events with tty input.
; AB, 9 January 1997
;     Changed the meaning of the CATCH keyword so that catching is the
;     default. Removed BLOCK and replaced with NO_BLOCK. Switched
;     default action back to blocking from unblocking based on feedback
;     from the IDL 5 beta. Added the ability to block only as long as a
;     client without NO_BLOCK is running, and then revert to the active
;     command line.
; AB, 10 February 1997
;     Cleaned up code to make it easier to understand and maintain.
;     Also cleaned up the distinction between real modality (MODAL
;     keyword to WIDGET_BASE) and XMANAGER's older fake modality
;     (MODAL keyword to XMANAGER), and fixed bugs in the current
;     implementation of fake modality.
; William Thompson, 25-Nov-1998, GSFC
;     Changed to allow CDS software to work in IDL/v5.1.1.
;-



PRO XmanagerPrintError
  ; Called when a client error is caught to print the error out for
  ; the user. Unfortunately no stack trace is available, but that's
  ; why XMANAGER,CATCH=0 exists.

  err = !err_string
  syserr = !syserr_string
  printf, -2, format='(A, A)', !ERROR_STATE.MSG_PREFIX, $
'XMANAGER: Caught unexpected error from client application. Message follows...'
  help,/last_message
END
```

```
PRO ValidateManagedWidgets
  ; Makes sure all the widgets in the list of managed widgets are still
  ; valid, and removes those that aren't.

  COMMON managed, ids, $  ; IDs of widgets being managed
     names, $ ; and their names
   modalList ; list of active modal widgets

  ; initialize the lists
  IF (NOT keyword_set(ids)) THEN BEGIN
    ids = 0L
    names = 0
  ENDIF

  ; if the list is empty, it's valid
  IF (ids[0] EQ 0L) THEN RETURN

  ; which ones are valid?
  valid = where(widget_info(ids, /managed))

  ; build new lists from those that were valid in the old lists
  IF (valid[0] EQ -1) THEN BEGIN
    ids = 0L
    names = 0
  ENDIF ELSE BEGIN
    ids = ids[valid]
    names = names[valid]
  ENDELSE

END



PRO AddManagedWidget, name, id
  ; Adds the given widget with its name to the list of managed widgets
  ;
  ; The list of managed widgets is kept as a convenience for applications
  ; that want to register their functionality by name. For instance, an app
  ; may not want to bring up a particular dialog if there is already one up.
  ; They can find out if the dialog is running by calling the XREGISTERED
  ; routine

  COMMON managed

  ValidateManagedWidgets
  IF (ids[0] EQ 0L) THEN BEGIN
    ; create new lists
    ids = [ id ]
    names = [ name ]
  ENDIF ELSE BEGIN
    ; insert at the beginning of the lists
    ids = [ id, ids ]
    names = [ name, names ]
  ENDELSE

END



FUNCTION LookupManagedWidget, name
  ; Returns the widget id of the named widget, or 0L if not found

  COMMON managed

  ValidateManagedWidgets

  IF (ids[0] NE 0L) THEN BEGIN
    found = where(names EQ name)
    IF (found[0] NE -1) THEN BEGIN
      RETURN, ids[found[0]]
    ENDIF
  ENDIF

  RETURN, 0L
END



PRO XUNREGISTER, corpse
  ;  ------------------------------------------------------------ ------
  ; | PLEASE NOTE: This routine is OBSOLETE. It's functionality is   |
  ; | is no longer necessary.                                        |
  ;  ------------------------------------------------------------ ------
  ;
  ; This procedure used to remove a dead widget from the Xmanagers common
  ; block, but that information is now maintained internally by IDL.

  COMMON XUNREGISTER_OBSOLETE, obsolete

  IF (NOT keyword_set(obsolete)) THEN BEGIN
    obsolete = 1
    message, /info, 'this routine is obsolete'
  END
  ; Might as well validate the list now (even though it would happen later)
  ValidateManagedWidgets

END





PRO XMANAGER_EVLOOP_STANDARD
  ; This is the standard XMANAGER event loop. It works by dispatching
  ; events for all managed widgets until there are none left that require
  ; blocking. In the best case, the command line is able to dispatch events
  ; and there are no clients that require blocking (specified via the
  ; NO_BLOCK keyword to XMANAGER) and we are able to return immediately.

  COMMON xmanager_local, fake_modal_obsolete, xmanager_catch


  ; WARNING: Undocumented feature. See RESTRICTIONS above for details.
  active = widget_info(/XMANAGER_BLOCK)
  WHILE (active NE 0) DO BEGIN
    err = 0
    IF (xmanager_catch) THEN catch, err
    IF (err EQ 0) THEN BEGIN
      ; WARNING: Undocumented feature. See RESTRICTIONS above for details.
      tmp = widget_event(/XMANAGER_BLOCK)
    ENDIF ELSE XmanagerPrintError
    IF (xmanager_catch) THEN catch, /cancel
    ; WARNING: Undocumented feature. See RESTRICTIONS above for details.
    active = widget_info(/XMANAGER_BLOCK)
  ENDWHILE

END



PRO XMANAGER_EVLOOP_REAL_MODAL, modal_id
  ; This version of the XMANAGER event loop is used when a client with
  ; the MODAL keyword set on its TLB has been passed in. It dispatches
  ; events for that client until it is done. Events for other clients
  ; are also flushed at critical points so that expose events are not
  ; delayed unnecessarily.

  COMMON xmanager_local
  active = 1
  WHILE (active NE 0) DO BEGIN
    err = 0
    IF (xmanager_catch) THEN catch, err
      IF (err EQ 0) THEN BEGIN
        ; WARNING: Undocumented feature. See RESTRICTIONS above for details.
        tmp = widget_event(MODAL_ID, bad_id=bad, /BREAK_ON_EXPOSE)
      ENDIF ELSE XmanagerPrintError
      IF (xmanager_catch) THEN catch, /cancel
      active = widget_info(MODAL_ID, /managed)

      ; Modal event handling returned. Flush events for other widgets
      ; so we do not keep expose events (among others) blocked.
      IF (active) THEN BEGIN
        err = 0
        IF (xmanager_catch) THEN catch, err
        IF (err EQ 0) THEN BEGIN
          tmp = widget_event(/NOWAIT)
        ENDIF ELSE XmanagerPrintError
        IF (xmanager_catch) THEN catch, /cancel
      ENDIF
  ENDWHILE
END



PRO XMANAGER_EVLOOP_FAKE_MODAL, ID
  ; This version of the XMANAGER event loop is used when a client is
  ; registered with the MODAL keyword to XMANAGER. It fakes the appearance
  ; of real modality by making the other existing clients insensitive while
  ; the modal widget exists.

  COMMON managed
  COMMON xmanager_local


  ; Remember the current modal list so it can be restored afterwards
  oldModalList = modalList
  modalList = [ ID ]
  ; WARNING: Undocumented feature. See RESTRICTIONS above for details.
;
;  This line was commented out, as advised by RSI.
;
;  WIDGET_CONTROL, ID, /MODAL

  ; Get list of clients that should be desensitized to mimic modality.
  ; If this is the outermost modal, then the list of all currently
  ; managed widgets is used. If this is a nested inner modal, then
  ; use the oldModalList.
  IF (keyword_set(oldModalList)) THEN BEGIN
    senslist = oldModalList
  ENDIF ELSE BEGIN
    WIDGET_CONTROL, ID, managed=0    ; So won't show up in following statement
    senslist = WIDGET_INFO(/MANAGED)
    WIDGET_CONTROL, ID, /MANAGED     ; Put it back
  ENDELSE
  for i = 0, n_elements(senslist) - 1 do $
    WIDGET_CONTROL, BAD_ID=ignore_bad, senslist[i], SENSITIVE=0


  ; Process events only for clients in the modal list. This list may gain
  ; members if event processing leads to other applications being registered
  ; via a recursive call to XMANAGER.
  tmp = where(widget_info(modalList, /managed), active)
  WHILE (active NE 0) DO BEGIN
    err = 0
    IF (xmanager_catch) THEN catch, err
    tmp = modalList
    IF (err EQ 0) THEN BEGIN
      ; WARNING: Undocumented feature. See RESTRICTIONS above for details.
      tmp = widget_event(tmp, bad_id=bad, /BREAK_ON_EXPOSE)
    ENDIF ELSE XmanagerPrintError
    IF (xmanager_catch) THEN catch, /cancel
    tmp = where(widget_info(modalList, /managed), active)
    IF (active NE 0) THEN modalList = modalList[tmp]
    ;
    ; Modal event handling returned, flush events for other widgets
    ; if any so we do not keep expose events etc. blocked
    ;
    IF (active) THEN BEGIN
      err = 0
      IF (xmanager_catch) THEN catch, err
      IF (err EQ 0) THEN BEGIN
        tmp = widget_event(/NOWAIT)
      ENDIF ELSE XmanagerPrintError
      IF (xmanager_catch) THEN catch, /cancel
    ENDIF
  ENDWHILE

  for i = 0, n_elements(senslist) - 1 do $
    WIDGET_CONTROL, BAD_ID=ignore_bad, senslist[i], /SENSITIVE

  ; restore the outer XMANAGER's list of modal widgets
  modalList = oldModalList

END
```

```
PRO XMANAGER, NAME, ID, BACKGROUND = background, CATCH = catch, $
  CLEANUP = cleanup, EVENT_HANDLER = event_handler, $
  GROUP_LEADER = group_leader, JUST_REG = just_reg, $
  MODAL = modal, NO_BLOCK = no_block

  COMMON managed
  COMMON xmanager_local


  isFakeModal = keyword_set(modal)

  ; print out obsolete keyword messages
  IF (keyword_set(background)) THEN BEGIN
    message, "The BACKGROUND keyword to the XMANAGER procedure is " + $
      "obsolete. It is superseded by the TIMER keyword to " + $
      "the WIDGET_CONTROL procedure.", /info
  ENDIF
  IF (isFakeModal AND (NOT keyword_set(fake_modal_obsolete))) THEN BEGIN
    fake_modal_obsolete = 1
    message, "The MODAL keyword to the XMANAGER procedure is " + $
      "obsolete. It is superseded by the MODAL keyword to " + $
      "the WIDGET_BASE function.", /info
  ENDIF


  ; Initialization
  if (n_elements(catch) ne 0) THEN BEGIN
    xmanager_catch = catch ne 0
    message, /INFO, 'Error handling is now ' + (['off', 'on'])[xmanager_catch]
    return
  ENDIF ELSE if (n_elements(xmanager_catch) EQ 0) then xmanager_catch = 1;
  isRealModal = 0
  if (N_ELEMENTS(just_reg) eq 0) then just_reg = 0
  IF (isFakeModal) THEN just_reg = 0;
  IF (NOT keyword_set(modalList)) THEN modalList = 0
  ValidateManagedWidgets


  ; Argument setup
  if (N_PARAMS() EQ 0) THEN BEGIN
    IF (ids[0] EQ 0L) THEN BEGIN
      message, 'No widgets are currently being managed.', /info
      RETURN
    ENDIF
  ENDIF ELSE IF (N_PARAMS() NE 2) THEN BEGIN
    message, 'Wrong number of arguments, usage: XMANAGER [, name, id]'
  ENDIF ELSE BEGIN ;2 argument case

    ; Check the arguments
    IF (NOT widget_info(id, /valid)) THEN message, 'Invalid widget ID.'
    nameinfo = size(name)
    IF ((nameinfo[0] NE 0) OR (nameinfo[1] NE 7)) THEN $
      message, 'Invalid widget name.'


    ; If TLB is modal, block in XMANAGER till you are done
    IF (widget_info(id, /Modal)) THEN isRealModal = 1

    IF (keyword_set(cleanup)) THEN widget_control, id, kill_notify=cleanup
    IF (NOT keyword_set(event_handler)) THEN event_handler = name + '_event'

    ; Register new widget
    AddManagedWidget, name, id

    ; Mark the widget for event processing
    widget_control, id, /managed, event_pro=event_handler

    ; Unless the caller set NO_BLOCK to indicate otherwise, mark
    ; this client as requiring XMANAGER to block. This decision is driven
    ; by backward compatibility concerns. During the IDL 5.0 beta we discovered
    ; that many customers have code that depends on the blocking behavior.
    ;
    ; WARNING: Undocumented feature. See RESTRICTIONS above for details.
    if keyword_set(no_block) then WIDGET_CONTROL, /XMANAGER_ACTIVE_COMMAND, id

    ; pass the group_leader keyword through
    IF (keyword_set(group_leader)) THEN $
      widget_control, id, group_leader=group_leader



    ; Modal Widget Registration
    IF (keyword_set(modalList) and (not isFakeModal)) THEN BEGIN

      ; This client is a non-modal widget, being started while a
      ; fake modal is already up. Just add the new widget to the modal
      ; list and return immediately. The fake modal event loop will
      ; dispatch its events as well as the modal clients.
      modalList = [ modalList, ID ]
      just_reg = 1 ; Don't process events. Instead, return immediately
```

```
      ; need to break out of the outer widget_event call so that the
      ; outer xmanager can see that outmodal has changed
      ; WARNING: Undocumented feature. See RESTRICTIONS above for details.
      widget_control, /event_break

    ENDIF   ; modal

  ENDELSE  ; 2 argument case



  ; Event Processing.
  IF (NOT just_reg) THEN BEGIN
    IF (isRealModal) THEN BEGIN
      XMANAGER_EVLOOP_REAL_MODAL, ID
    ENDIF ELSE IF isFakeModal THEN BEGIN
       XMANAGER_EVLOOP_FAKE_MODAL, ID
    ENDIF ELSE BEGIN
      XMANAGER_EVLOOP_STANDARD
    ENDELSE

    ; keep our list clean and up to date
    ValidateManagedWidgets

  ENDIF

END
```