

# W I N . P A S

task : control text mode window

author : Michael Tischer / Bruno Jennrich  
 developed on : 5/3/1994  
 last update : 04/6/1995

init WIN;

interface

```

const
    VM_COLOR      = 1;
    VM_MONO       = 2;

    VIOINT        = $10;
    VIOPAGE       = $00;
    VIOETCURSOR   = $02;
    VIOETCURSOR   = $03;
    VIOETPAGE     = $05;
    VIOGETMODE    = $0F;

    WIN_CRLF      = $0001;
    WIN_SCROLL    = $0002;
    WIN_HASCURSOR = $0004;
    WIN_ACTIVE    = $0008;

    OT_INT        = 1;
    OT_BOOL       = 2;
    DT_TRUEFALSE  = 0;
    DT_ONOFF      = 1;
    
```

{ OBJTYPE - INT }

```
DT_YESNO           = 2;
.1 .4 cm 173 Raster
MSG_LOSTFOCUS      = 0;
MSG_GOTFOCUS       = 1;
MSG_KEY            = 2;
MSG_CHANGED        = 3;
MSG_PRECHANGE      = 4;
KBD_ESC            = 27;
KBD_TAB            = 9;
KBD_SHIFTTAB       = 15 * 256;
KBD_BACK           = 8;
KBD_CR             = 13;
KBD_LF             = 10;
KBD_F1             = 59 * 256;
KBD_F2             = 60 * 256;
KBD_F3             = 61 * 256;
KBD_F4             = 62 * 256;
KBD_F5             = 63 * 256;
KBD_F6             = 64 * 256;
KBD_F7             = 65 * 256;
KBD_F8             = 66 * 256;
KBD_F9             = 67 * 256;
KBD_F10            = 68 * 256;
KBD_UP             = 72 * 256;
KBD_LEFT           = 75 * 256;
KBD_DN             = 80 * 256;
KBD_RIGHT          = 77 * 256;
KBD_DEL            = 83 * 256;
KBD_HOME           = 71 * 256;
KBD_END            = 79 * 256;
KBD_PGUP           = 73 * 256;
KBD_PGDN           = 81 * 256;
KBD_CTRLPGUP       = 132 * 256;
```

```
KBD_CTRLPGDN    = 118 * 256;
```

```
KBD_PLUS        = 43;
```

```
KBD_MINUS       = 45;
```

```
type
WINDOW = Record { describes a window }
  iX,iY,           { position to the subordinate window/screen }
  iW,iH,           { width and height }
  iCX,iCY : Integer; { current print position }
  uFlags : Word;     { s. WIN_??? }
  iAttr,          { current attribute }
  iHiAttr,        { active attribute }
  iLoAttr : Byte;   { passive attribute }
end;
WINDOW = ^WINDOW;
```

```
-- Window-Procedure, for ex., for handling ---}
-- keyboard input                                ---}
inproc = Procedure( iNum, msg, iParam : Integer; lParam : Longint );
```

```
NTDATA = Record { describes an integer input field }
  iMin, iMax : Integer;
  pValue      : ^Integer;
  pText       : String;
end;
```

```
BOOLDATA = Record { describes a BOOL input field }
  iDisplay : Integer;
  pValue    : ^Boolean;
  pText     : String;
end;
```

```

OBJ = record { describes an input field }
  X, Y, W, H,
  iType      : Integer;
  pData      : Pointer;
end;
OBJPTR = ^OBJ;

-- Prototypes -----}

procedure win_Init( var Win           : WINDOW;
                   iX, iY, iW, iH : Integer;
                   iLA, iHA       : Byte;
                   uFlags         : Word );

procedure win_LoVideo( var Win : WINDOW );

procedure win_HiVideo( var Win : WINDOW );

procedure win_GetVIOSMEM;

function win_GetMode : Word;

procedure win_GotoXY( var Win : WINDOW; iX, iY : Integer );

procedure win_Clr( var Win : WINDOW );

function win_Save( var Win : WINDOW ) : Pointer;

procedure win_Restore( pMem : Pointer; bFree : Boolean );

procedure win_GetScreenSettings( var Win : WINDOW );

procedure win_ScrollUp( var Win : WINDOW ; NumRows : Integer );

```

```

'rocedure _win_Print( var Win   : WINDOW;
                     Text  : String;
                     iCnt  : Integer );

'rocedure win_Print( var Win : WINDOW; Text : String );

'rocedure win_PrintAt( var Win      : WINDOW;
                      iX, iY : Integer;
                      Text   : String );

'rocedure win_Beep;

'rocedure win_ObjectInitINT( var Obj      : OBJ;
                             var Data    : INTDATA;
                             x, y, w, h : Integer;
                             Text      : String;
                             iMin, iMax : Integer;
                             pValue     : Pointer );

'rocedure win_ObjectInitBOOL( var Obj      : OBJ ;
                              var Data    : BOOLDATA;
                              x, y, w, h : Integer;
                              Text      : String;
                              iDisplay   : Integer;
                              pValue     : Pointer );

'rocedure win_ObjectPrint( var Win : WINDOW; var Obj : OBJ );

'unction win_ObjectQueryValue( var Obj : OBJ ) : Longint;

'rocedure win_ObjectPrintArray( var Win      : WINDOW ;
                               pOArray : pointer;

```

```

                                iNum,
                                iAct    : Integer );

procedure win_ObjectProcessKey( var Win   : WINDOW;
                                Obj     : OBJ ;
                                iKey    : Integer );

function win_ObjectWantsKey( Obj : OBJ; iKey : Integer ) : Boolean;

procedure win_ObjectProcessArray( var Win       : WINDOW;
                                poArray  : pointer;
                                iNum     : Integer;
                                lpFunc   : winproc );

implementation
uses DOS,CRT;

const MAXOBS = 100;

type ObjArray = array [0..MAXOBS-1] of OBJ; { for access to }
type ObjArrayPtr = ^ObjArray;              { OBJ-Arrays }

- global variables -----}
const
lpVIOSMEM : ^Integer = NIL;           { address of the screen memory }
wVIOSMODE : Word     = 0;              { current video mode }
Columns   : Integer  = 0;              { number of lines presented }

*****}
win_Init : initializes WINDOW structure
*-----*}
input : Win      - WINDOW structure to be initialized
       iX, iY    - window position relative to subordinate

```

```

                                window, or screen.
                                origin : ( 0,0 )
                                }
                                {
                                iW, iH   - width and height of the window
                                iLA, iHA - test attributes ( passive, active )
                                uFlags   - Flags
                                }
                                *****
procedure win_Init( var Win           : WINDOW;
                   iX, iY, iW, iH     : Integer;
                   iLA, iHA           : Byte;
                   uFlags              : Word );
begin
    Win.iX      := iX;
    Win.iY      := iY;
    Win.iW      := iW;
    Win.iH      := iH;
    Win.iLoAttr := iLA;
    Win.iHiAttr := iHA;

    Win.uFlags := uFlags;

    Win.iCX := 0;
    Win.iCY := 0;
    win_LoVideo( Win );
end;

*****
win_LoVideo : switch print to lowest intensity
*-----*
input : Win - the window, in which the following print outs
        having the lowest intensity should occur.
*****
procedure win_LoVideo( var Win : WINDOW );

```

```

begin
  Win.iAttr := Win.iLoAttr;
end;

*****}
win_HiVideo : switch print out to high intensity
*-----*
input : Win - the window, in which the followint print outs
        having higher intensity should occur.
*****}
procedure win_HiVideo( var Win : WINDOW );

begin
  Win.iAttr := Win.iHiAttr;
end;

*****}
win_GetVIOSMEM : Determines the screen memory start address
                and current video mode.
*-----*
Info : The screen memory's address is saved in the global
        variable _lpVIOSMEM. The video mode is recorded
        in _iVIOSMODE.
*****}
procedure win_GetVIOSMEM;

var regs : Registers;

begin
  if _lpVIOSMEM = NIL then           { address already determined ? }
    Begin { No, not yet }
      regs.ah := VIOSETPAGE;
      regs.al := VIOPAGE;
    end;
  end;
end;

```



```

intr( VIOINT, regs );

regs.ah := VIOGETMODE;
regs.al := VIOPAGE;
intr( VIOINT, regs );
_Columns := regs.ah;           { number of columns }
_wVIOSMODE := regs.al;
if _wVIOSMODE = $07 then _lpVIOSMEM := ptr( $B000, 0 )
    else _lpVIOSMEM := ptr( $B800, 0 );

End;
Ind;

*****}
win_GetMode : Determines current screen mode.
*-----*
output      : VM_COLOR ( 1 ) - color screen
              VM_MONO  ( 2 ) - Monochrome screen
              otherwise    - not recognized!
*****}

function win_GetMode : Word;

    .1 .4 cm 543 screen
begin
    win_GetVIOSMEM;
    win_GetMode := _wVIOSMODE;
Ind;

*****}
win_GotoXY : set print position within a window
*-----*
input : Win      - the window, in which a new print position
              should be set.
              iX, iY - New position
*****}

```

```

-----
Info : If the flag is set at "WIN_HASCURSОР", this function
      also sets the blinking screen cursor.
*****
procedure win_GotoXY( var Win : WINDOW; iX, iY : Integer );

var regs : Registers;

begin
  if iX < 0 then iX := 0;      { test validity of the new position }
  if iY < 0 then iY := 0;
  if iX > Win.iW then iX := Win.iW - 1;
  if iY > Win.iH then iY := Win.iH - 1;

  Win.iCX := iX; { find new cursor position }
  Win.iCY := iY;

  if( Win.uFlags and WIN_ACTIVE ) <> 0 then { window active ? }
    Begin
      if( Win.uFlags and WIN_HASCURSОР ) <> 0 then
        Begin { window has the cursor }
          regs.dh := Byte ( Win.iY + Win.iCY );
          regs.dl := Byte ( Win.iX + Win.iCX );
        End
      else
        Begin
          regs.dh := $FF;
          regs.dl := $FF;
        End;
      regs.ah := VIOSETCURSOR; { set cursor over BIOS }
      regs.bh := VIOPAGE;
      intr( VIOINT, regs );
    End;
end;

```

```

.na;

*****}
win_Clr : erase window contents
*-----*
input : Win - the window having the contents
        that need to be erased
*****}
procedure win_Clr( var Win:WINDOW );

var iX, iY : Integer;
    lpRow   : ^Word;
    iAdr    : Word;

begin
    win_GetVIOSMEM;
    for iY := 0 to Win.iH - 1 do { run through all lines }
    Begin
        iAdr := ( Win.iX + ( iY + Win.iY ) * _Columns ) * 2;
        lpRow := ptr( seg( _lpVIOSMEM^ ), iAdr );
        iAdr := Win.iAttr * 256 + 32;
        for iX := 0 to Win.iW - 1 do { run through all line columns }
        Begin
            lpRow^ := iAdr; { write empty character }
            Inc( lprow );
        End;
    End;

    Win.iCX := 0; { cursor in upper left window corner}
    Win.iCY := 0;

end;

*****}

```

```

win_Save : save window contents
}
*-----*
input : Win - the window having the contents that should be saved
output : Address for the memory that contains window
        information together with the content, or ZERO
        if no other memory is available.
}
*****
function win_Save( var Win : WINDOW ) : Pointer;

var   pMem,
      pWinData   : pWINDOW;
      lpRows, lpR : ^Word;
      iY, iX      : Integer;
      iAdr        : Word;

begin
  win_GetVIOGMEM;
  GetMem( pMem, ( Win.iW * Win.iH * 2 ) + sizeof( WINDOW ) );
  if pMem <> NIL then
    Begin { sufficient memory could be allocated }
      pWinData := pMem; { window description at the beginning }
      pWinData^ := Win; { copy buffers }

      {-- copy the window contents line by line into the buffer ----}
      lpRows := Pointer ( Longint ( pWinData ) + sizeof( WINDOW ) );
      for iY := 0 to Win.iH - 1 do
        Begin
          iAdr := ( Win.iX + ( iY + Win.iY ) * _Columns ) * 2;
          lpR := ptr( seg( _lpVIOGMEM^ ), iAdr );
          for iX := 0 to Win.iW - 1 do
            Begin
              lpRows^ := lpR^;
              Inc( lpRows );
            end
          end
        end
      end
    end
  end
end

```

```

        Inc( lpR );
    End;
End;
win_Save:=pMem; { return counter to buffer }
end;

*****}
win_Restore : reestablish window contents
*-----*
input : pMem - Address of the memory previously transmitted
          via win_Save which contains window information
          ( position, dimension, cursor ),
          as well as content
          bFree- Should the memory be free after
                  restoration? ( in this case bFree: = TRUE )
*****}
procedure win_Restore( pMem : Pointer; bFree : Boolean );

var lpRows, lpR : ^Word;
    pWinData    : PWINDOW;
    iX, iY      : Integer;
    iAdr        : Word;

begin
    win_GetVIOSMEM;
    if pMem <> NIL then
        Begin { buffer counter shows nothing in the empty space }
            pWinData := pMem;
            lpRows := Pointer ( Longint ( pWinData ) + sizeof( WINDOW ) );

            {-- run through the buffer line by line and restore -----}

```

```

{-- window contents -----
for iY := 0 to pWinData^.iH do
  Begin
    iAdr := ( pWinData^.iX + ( iY + pWinData^.iY ) * _Columns ) * 2;
    lpR := ptr( seg( _lpVIOGMEM^ ), iAdr );
    for iX := 0 to pWinData^.iW - 1 do { run through the line }
      Begin
        { columns }
        lpR^ := lpRows^;
        Inc( lpR );
        Inc( lpRows );
      End;
    End;
  win_GotoXY( pWinData^, pWinData^.iCX, pWinData^.iCY );

  if bFree then dispose( pMem ); { release memory at will }
End;
nd;

*****}
win_GetActScreenSettings : Determines current screen
                          settings
*-----*
input : Win - WINDOW structure, which should accept
        screen settings
*-----*
Info : Current video mode und current video page are not
        saved in WINDOW structure.
*****}
procedure win_GetScreenSettings( var Win : WINDOW );

var regs : Registers;

```

```

begin
    win_GetVIOSMEM;

    win_Init( Win, 0, 0, _Columns, 25,
              $00, $00, WIN_HASCURSOR or WIN_ACTIVE );

    regs.ah := VIOGETCURSOR;
    regs.bh := VIOPAGE;
    intr( VIOINT, regs );

    Win.iCX := regs.dl;
    Win.iCY := regs.dh;
end;

*****}
win_ScrollUp : scroll window contents up
*-----*}
input : Win      - the window
        NumRows - number of lines to scroll up
*-----*}
Info : The area left empty after scrolling is filled
        with spaces in the current text attribute.
*****}
procedure win_ScrollUp( var Win : WINDOW ; NumRows : Integer );

var iX, iY      :Integer;
    lpRow,
    lpRowDest : ^Word;
    Clear      : WINDOW;
    iDAdr,
    iAdr       : Word;

begin
    win_GetVIOSMEM;

```

```

for iY := NumRows to Win.iH - 1 do { run through the lines }
  Begin
    iDAdr := ( Win.iX + ( ( iY-NumRows ) + Win.iY ) * _Columns ) * 2;
    iAdr := ( Win.iX + ( iY + Win.iY ) * _Columns ) * 2 ;
    lpRowDest := ptr( seg( _lpVIOGMEM^ ), iDAdr );
    lpRow      := ptr( seg( _lpVIOGMEM^ ), iAdr );
    for iX := 0 to Win.iW do { run through the columns }
      Begin
        lpRowDest^ := lpRow^;
        Inc( lpRowDest );
        Inc( lpRow );
      End;
    End;
  win_Init( Clear, Win.iX, Win.iY + ( Win.iH - NumRows ),
            Win.iW, NumRows, Win.iLoAttr, Win.iHiAttr,
            WIN_CRLF or WIN_SCROLL );
  win_Clr( Clear );
end;

*****}
_win_PrINT : text print out engine
*-----*
input : Win    - the window in which text should be printed
       Text    - the text to be printed
       iCnt    - number of lines to be printed.
               ( - 1 := entire text )
*****}

procedure _win_Print( var Win : WINDOW; Text : String; iCnt : Integer );

var i      : Integer;
    lpRow  : ^Byte;
    iAdr   : Word;

```



```

    c      : Char;
begin
    i := 1;

    win_GetVIOSMEM;

    iAdr := ( Win.iX + Win.iCX + ( Win.iCY + Win.iY ) * _Columns ) * 2;
    lpRow := ptr( seg( _lpVIOSMEM^ ), iAdr );

    while( ( i <= Length( Text ) ) and ( iCnt <> 0 ) ) do
        Begin
            c := Text[i];
            case c of
                #13:
                    if( ( not ( Win.uFlags and WIN_CRLF ) <> 0 ) ) then
                        Begin
                            Win.iCX := 0;                { at the beginning of the actual line

                            iAdr := ( Win.iX + Win.iCX + ( Win.iCY + Win.iY )
                                * _Columns ) * 2;
                            lpRow := ptr( seg( _lpVIOSMEM^ ), iAdr );
                        End;

                #10:
                    Begin
                        while Win.iCX < Win.iW do                { LF }
                            { erase remainder of the
                                line }
                                Begin
                                    lpRow^:= 32; Inc( lpRow );
                                    lpRow^ := Win.iAttr; Inc( lpRow );
                                    Inc( Win.iCX );
                                End;
                                Inc( Win.iCY );                { Next line }
                    End;
            end;
        End;
    end;

```

```

        if Win.iCY >= Win.iH then          { reached last line? }
            Begin
                Win.iCY := Win.iH - 1;
                if( Win.uFlags and WIN_SCROLL ) <> 0 then
                    win_ScrollUp( Win, 1 );      { scroll contents }
                End;
            Win.iCX := 0;
            iAdr := ( Win.iX + Win.iCX + ( Win.iCY + Win.iY )
                    * _Columns ) * 2;
            lpRow := ptr( seg( _lpVIOGMEM^ ), iAdr );
        End;
    else                                     { print normal character }
        Begin
            lpRow^ := Byte ( c ); Inc( lpRow );
            lpRow^ := Win.iAttr; Inc( lpRow );
            Inc( Win.iCX );
            if Win.iCX >= Win.iW then
                Begin
                    Inc( Win.iCY );                                { Next line }
                    if Win.iCY >= Win.iH then          { reached last line? }
                        Begin
                            Win.iCY := Win.iH - 1;
                            if( Win.uFlags and WIN_SCROLL ) <> 0 then
                                win_ScrollUp( Win, 1 ); { scroll contents }
                            End;
                        Win.iCX := 0;
                        iAdr := ( Win.iX + Win.iCX + ( Win.iCY + Win.iY )
                                * _Columns ) * 2;
                        lpRow := ptr( seg( _lpVIOGMEM^ ), iAdr );
                    End;
                End;
            End;
        End;
    End;
End;

```

```

        Inc( i ); { next character }
        if iCnt > 0 then Dec( iCnt );
                                { decrease counter if positive }
    End;
    win_GotoXY( Win, Win.iCX, Win.iCY );
End;

*****
win_Print : print string in window
*-----*
input : Win   - window, in which the string will be printed
       Text  - string to be printed.
*-----*
Info : This function utilizes _win_Print to print.
*****
procedure win_Print( var Win : WINDOW; Text : String );

begin
    _win_Print( Win, Text,  - 1 );
End;

*****
win_PrintAT : print text at any position in the window
*-----*
input : Win       - window, in which the string will be printed
       iX, iY    - print position
       Text      - string to be printed.
*****
procedure win_PrintAt( var Win       : WINDOW;
                      iX, iY : Integer;
                      Text  : String );

begin
    win_GotoXY( Win, iX, iY );

```

```

_win_Print( Win, Text, -1 );
end;

*****
win_Beep : produce warning sound
*****
}

procedure win_Beep;

const uFrq : Word = 400;
var l : Longint;

begin
    Sound(uFrq);
    for l := 0 to 9999 do
        NoSound;
    end;

    *****
    win_ObjectInitINT - connect object with INT type information
    *-----*
    input: Object      - type free object
           Data        - type information( PINTDATA )
           x, y, w, h  - position, dimension of the object
           Text       - text to be printed
           iMin, iMax - integers marginal values
           pValue     - address of the variables that accept INT
    *****
}

procedure win_ObjectInitINT( var Obj      : OBJ ;
                             var Data    : INTDATA;
                             x, y, w, h : Integer;
                             Text       : String;
                             iMin, iMax : Integer;
                             pValue     : Pointer );

```



```

Obj.X := x;
Obj.Y := y;
Obj.W := w; { are not used ( yet ) }
Obj.H := h;
Obj.iType := OT_BOOL;
Obj.pData := @Data;
Data.pText := Text;
Data.iDisplay := iDisplay;
Data.pValue := pValue;
end;

*****}
win_ObjectPrint - Object display
*-----*
input: Win      - print window
              Object - object to be printed
*-----*
Info: In this function, the OBJect styles are transmitted,
      in order to print the OBJects according to their
      respective style.
*****}

procedure win_ObjectPrint( var Win : WINDOW; var Obj : OBJ );

var pBuffer    : String;
    pIntData   : ^INTDATA ;
    pBoolData  : ^BOOLDATA ;

begin
  case Obj.iType of
    OT_INT:
      Begin      { dereference obj }
        pIntData := Obj.pData;
        str( pIntData^.pValue^:5, pBuffer );

```

```

        pBuffer := pIntData^.pText + pBuffer;
        win_PrintAt( Win, Obj.X, Obj.Y, pBuffer );
    End;

OT_BOOL:
    Begin        { dereference obj }
        pBoolData := Obj.pData;
        case pBoolData^.iDisplay of
            DT_TRUEFALSE:
                Begin
                    pBuffer := pBoolData^.pText;
                    if pBoolData^.pValue^ then pBuffer := pBuffer + ' TRUE'
                                                else pBuffer := pBuffer + 'FALSE';
                End;

            DT_ONOFF:
                Begin
                    pBuffer := pBoolData^.pText;
                    if pBoolData^.pValue^ then pBuffer := pBuffer + ' AN'
                                                else pBuffer := pBuffer + 'OUT';
                End;

            DT_YESNO:
                Begin
                    pBuffer := pBoolData^.pText;
                    if pBoolData^.pValue^ then pBuffer := pBuffer + ' YES'
                                                else pBuffer := pBuffer + 'NO';
                End;
        End;
        win_PrintAt( Win, Obj.X, Obj.Y, pBuffer );
    End;
End;
End;

```

```

*****}
win_ObjectQueryValue - Determine object value
*-----*
input: Object      - object whose value is to be determined
output: value, whose type depends on the OBJect type, and thus
        may need to be changed.
*****}

function win_ObjectQueryValue( var Obj : OBJ ) : Longint;

var pIntData   : ^INTDATA;
    pBoolData  : ^BOOLDATA;

begin
  case Obj.iType of
    OT_INT :
      Begin { dereference obj }
        pIntData := Obj.pData;
        win_ObjectQueryValue := pIntData^.pValue^;
      End;

    OT_BOOL:
      Begin { dereference obj }
        pBoolData := Obj.pData;
        win_ObjectQueryValue := Longint( pBoolData^.pValue^ );
      End;
    else
      win_ObjectQueryValue := 0;
    End;
  End;

  *****}
  win_ObjectPrintArray - print object list
  *****}

```



```

-----^
input: Win      - print window
      pOArray   - counter on the first object in the array
      iNum      - number of OBJECTs
      iAct      - number of the current OBJECT
*****
procedure win_ObjectPrintArray( var Win      : WINDOW ;
                               pOArray     : pointer;
                               iNum, iAct   : Integer );

var i      : Integer;
    OaPtr  : ObjArrayPtr;

begin
    OaPtr := pOArray;
    for i := 0 to iNum - 1 do
        { display everything }
    Begin
        if i = iAct then win_HiVideo( Win )
            else win_LoVideo( Win );
        win_ObjectPrint( Win, OaPtr^[i] );
    End;
end;

*****
win_ObjectProcessKey - manage/process keyboard input
-----*
input: Win      - print window
      Object    - object, which is to process the keyboard input
      iKey      - keyboard code
*****
procedure win_ObjectProcessKey( var Win      : WINDOW;
                               Obj         : OBJ;
                               iKey        : Integer );

var pIntData    : ^INTDATA;

```

```

pBoolData : ^BOOLDATA;

begin
  case Obj.iType of
    OT_INT:
      Begin { dereference object }
        pIntData := Obj.pData;
        case iKey of
          KBD_PLUS: Inc( pIntData.pValue^ );
          KBD_MINUS: Dec( pIntData.pValue^ );
        End;
        if pIntData.pValue^ < pIntData.iMin then
          pIntData.pValue^ := pIntData.iMax;
        if pIntData.pValue^ > pIntData.iMax then
          pIntData.pValue^ := pIntData.iMin;

          win_HiVideo( Win );
          win_ObjectPrint( Win, Obj );
        End;

    OT_BOOL:
      Begin
        pBoolData := Obj.pData; { dereference object }
        case iKey of
          KBD_PLUS, KBD_MINUS:
            if pBoolData.pValue^ then pBoolData.pValue^ := FALSE
            else pBoolData.pValue^ := TRUE;
        End;
        win_HiVideo( Win );
        win_ObjectPrint( Win, Obj );
      End;
    End;
  end;
end;

```

```

*****}
win_ObjectWantsKey - Can the object use the keyboard input?
*-----*
input: Object - object, which is to process the keyboard input
       iKey   - keyboard code
output:= 0    - keyboard code cannot be used
       <> 0    - Object keyboard is usable
*****}

function win_ObjectWantsKey( Obj : OBJ; iKey : Integer ) : Boolean;

begin
  case Obj.iType of
    OT_INT, OT_BOOL:
      case iKey of
        KBD_PLUS, KBD_MINUS:
          win_ObjectWantsKey := TRUE;
        else
          win_ObjectWantsKey := FALSE;
      end;
    End;
  End;
end;

*****}
win_ObjectProcessArray - process object list ( dialog )
*-----*
input: Win      - print window
       pOArray  - Object list
       iNum     - number of objects
       lpFunc   - message function
*-----*

Info: The input FOCUS is shifted with TAB and <SHIFT>+TAB.
      The user function pFunc is called up, in order

```

```

        to give the user the opportunity to respond immediately to
        new OBJECT states.
    }
    *****
procedure win_ObjectProcessArray( var Win      : WINDOW;
                                pOArray : pointer;
                                iNum      : Integer;
                                lpFunc    : winproc );

var iAct      : Integer;
    iLoop     : Boolean;
    ch        : Char;
    c         : Word;
    oldVal    : Longint;
    OaPtr     : ObjArrayPtr;

begin
    OaPtr := pOArray;
    iAct := 0;
    win_ObjectPrintArray( Win, pOArray, iNum, iAct );

    iLoop := TRUE;
    while iLoop do
        Begin
            ch := ReadKey;
            if ch = #0 then c := Word ( ReadKey ) * 256
                else c := Word( ch );
            if win_ObjectWantsKey( OaPtr^[iAct], c ) then
                Begin
                    if @lpFunc <> NIL then
                        lpFunc( iAct, MSG_PRECHANGE, 0, 0 );
                    oldVal := win_ObjectQueryValue( OaPtr^[iAct] );
                    win_ObjectProcessKey( Win, OaPtr^[iAct], c );
                    if @lpFunc <> NIL then

```

```

        lpFunc( iAct, MSG_CHANGED, 0, oldVal );
    End
else
    Begin
        win_LoVideo( Win );
        win_ObjectPrint( Win, OaPtr^[iAct] );
        if @lpFunc <> NIL then
            lpFunc( iAct, MSG_LOSTFOCUS, 0, 0 );
        case c of
            KBD_ESC:
                iLoop := FALSE;

            KBD_UP:
                Begin
                    Dec( iAct );
                    if( iAct < 0 ) then iAct := iNum - 1;
                End;

            KBD_DN:
                Begin
                    Inc( iAct );
                    if( iAct >= iNum ) then iAct := 0;
                End;
        End;
    End;

    win_HiVideo( Win );
    win_ObjectPrint( Win, OaPtr^[iAct] );
    if @lpFunc <> NIL then
        lpFunc( iAct, MSG_GOTFOCUS, 0, 0 );
    End;
    if @lpFunc <> NIL then
        lpFunc( iAct, MSG_KEY, c, 0 );
    End;
End;

```

ind.

ind.